LNCS 8165

ARCoSS

Andreas Brandstädt
Klaus Jansen
Rüdiger Reischuk (Eds.)

Graph-Theoretic Concepts in Computer Science

39th International Workshop, WG 2013
Lübeck, Germany, June 2013
Revised Papers

Springer

Andreas Brandstädt Klaus Jansen
Rüdiger Reischuk (Eds.)

Graph-Theoretic Concepts in Computer Science

39th International Workshop, WG 2013
Lübeck, Germany, June 19-21, 2013
Revised Papers

Springer

Volume Editors

Andreas Brandstädt
University of Rostock, Institute of Computer Science
Albert-Einstein-Straße 22, 18059 Rostock, Germany
E-mail: ab@informatik.uni-rostock.de

Klaus Jansen
University of Kiel, Department of Computer Science
Olshausenstraße 40, 24098 Kiel, Germany
E-mail: kj@informatik.uni-kiel.de

Rüdiger Reischuk
University of Lübeck, Institute of Theoretical Computer Science
Ratzeburger Allee 160, 23538 Lübeck, Germany
E-mail: reischuk@tcs.uni-luebeck.de

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-45042-6 e-ISBN 978-3-642-45043-3
DOI 10.1007/978-3-642-45043-3
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013952713

CR Subject Classification (1998): G.2, F.2, I.2.8, E.1, I.3

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

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

Printed on acid-free paper

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

Preface

The 39th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2013) took place in Lübeck, Germany, June 19–21, 2013.

The WG conference has a long tradition. Since 1975, it has taken place 22 times in Germany, four times in The Netherlands, twice in Austria, the Czech Republic, and France as well as once in Italy, Slovakia, Switzerland, Norway, the UK, Greece, and in Israel.

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

There were 61 submissions. Each submission was carefully reviewed by at least four members of the Program Committee. The committee accepted 34 papers to be presented at the workshop. The program also included three inspiring invited talks: Ola Svensson (EPFL Lausanne, Switzerland) presented "New Approaches for Approximating TSP," Berthold Vöcking (RWTH Aachen, Germany) discussed the problem "Online Independent Set for Graphs with Bounded Inductive Independence," and Feodor F. Dragan (Kent State University, USA) gave a talk about "Tree-like Structures in Graphs: a Metric Point of View."

We would like to thank the authors who submitted their papers, the speakers, the members of the Program Committee, and the external reviewers. Special thanks to the local Organizing Committee; without their performance the conference could not have been such a success.

We are grateful to the Institute of Computer Science at the University of Rostock, the Institute of Computer Science of the Christian-Albrechts-Universität zu Kiel, and the Institute of Theoretical Computer Science at the University of Lübeck.

August 2013

Andreas Brandstädt
Klaus Jansen
Rüdiger Reischuk

Organization

Program Committee

Andreas Brandstädt	University of Rostock, Germany
Artur Czumaj	University of Warwick, UK
Pinar Heggernes	University of Bergen, Norway
Juraj Hromkovic	ETH Zürich, Switzerland
Klaus Jansen	University of Kiel, Germany
Jan Kratochvil	Charles University, Czech Republic
Van Bang Le	University of Rostock, Germany
Bodo Manthey	University of Twente, The Netherlands
Daniel Marx	Hungarian Academy of Sciences, Hungary
Monaldo Mastrolilli	IDSIA Lugano, Switzerland
Ernst W. Mayr	TU München, Germany
Haiko Müller	University of Leeds, UK
Rolf Niedermeier	TU Berlin, Germany
Christophe Paul	LIRMM Montpellier, France
Dieter Rautenbach	University of Ulm, Germany
Rüdiger Reischuk	University of Lübeck, Germany
Heiko Röglin	University of Bonn, Germany
Christian Sohler	TU Dortmund, Germany
Ioan Todinca	Université d'Orleans, France
Dorothea Wagner	University of Karlsruhe, Germany

Additional Reviewers

Barhum, Kfir
Baum, Moritz
Belmonte, Rémy
Bentz, Cédric
Berndt, Sebastian
Berry, Anne
Bläsius, Thomas
Bodlaender, Hans L.
Boehme, Thomas
Bollig, Beate
Bonichon, Nicolas
Bonomo, Flavia
Bonsma, Paul
Bousquet, Nicolas
Brandes, Ulrik
Broersma, Hajo
Böckenhauer, Hans-Joachim
Chang, Maw-Shang
Chapelle, Mathieu
Chaplick, Steven
Chauve, Cedric
Cheilaris, Panagiotis
Crespelle, Christophe
Dibbelt, Julian
Didimo, Walter
Doerr, Carola
Dragan, Feodor
Drechsler, Rolf
Driemel, Anne
Dvorak, Zdenek

Ehsanfar, Ebrahim
Elberfeld, Michael
Englert, Matthias
Erlebach, Thomas
Fabila-Monroy, Ruy
Farzad, Babak
Feldmann, Andreas Emil
Felsner, Stefan
Fernau, Henning
Figueiredo, Celina
Froese, Vincent
Fuchs, Fabian
Fulek, Radoslav
Gargano, Luisa
Gaspers, Serge
Gebauer, Heidi
Gille, Marc
Golovach, Petr
Gonçalves, Daniel
Guo, Jiong
Gurski, Frank
Hartmann, Tanja
Hartung, Sepp
Hellweg, Frank
Hlineny, Petr
Hoefer, Martin
Hoeksma, Ruben
Hüffner, Falk
Jansen, Bart M.P.
Kaklamanis, Christos
Kappes, Andrea
Keller, Lucia
Kern, Walter
Klavík, Pavel
Kobourov, Stephen
Komm, Dennis
Komusiewicz, Christian
Kosirova, Ivana
Kowaluk, Miroslaw
Kratsch, Dieter
Kratsch, Stefan
Krivosija, Amer
Krug, Sacha
Kunold, Tim
Köhler, Ekkehard
Land, Kati
Leveque, Benjamin
Li, Zhentao
Liedloff, Mathieu
Limouzy, Vincent
Lingas, Andrzej
Liskiewicz, Maciej
Lokshtanov, Daniel
Maffray, Frederic
Matamala, Martín
Mchedlidze, Tamara
Meister, Daniel
Mertzios, George
Milanic, Martin
Munteanu, Alexander
Nichterlein, André
Niedermann, Benjamin
Nisse, Nicolas
Ochem, Pascal
Otachi, Yota
Oum, Sang-Il
Paulusma, Daniel
Peng, Sheng-Lung
Pilipczuk, Marcin
Pilipczuk, Michał
Pinkau, Chris
Proskurowski, Andrzej
Protti, Fabio
Prutkin, Roman
Rafiey, Arash
Rutter, Ignaz
Salazar, Gelasio
Sau, Ignasi
Schiermeyer, Ingo
Schmidt, Melanie
Schuster, Martin
Schweitzer, Pascal
Schwiegelshohn, Chris
Serna, Maria
Shah, Chintan
Smula, Jasmin
Sorge, Manuel
Sprock, Andreas
Stamoulis, Georgios
Steffen, Björn

Steiner, George
Stockhusen, Christoph
Strasser, Ben
Suchý, Ondřej
Taeubig, Hanjo
Tantau, Till
Thilikos, Dimitrios
Tholey, Torsten
Thomborson, Clark
Toman, Stefan
Török, L'ubomír
van't Hof, Pim
van Bevern, René
van der Zwaan, Ruben
Villanger, Yngve
Wahlström, Magnus
Walczak, Bartosz
Weihmann, Jeremias
Weller, Mathias
Westermann, Matthias
Whidden, Chris
Wilson, Samuel
Witt, Oliver
Wollan, Paul
Wood, David
Zemmari, Akka

Abstracts

Tree-Like Structures in Graphs: A Metric Point of View[*]

Feodor F. Dragan

Department of Computer Science, Kent State University, Kent, OH 44242, USA
dragan@cs.kent.edu

Abstract. Recent empirical and theoretical work has suggested that many real-life complex networks and graphs arising in Internet applications, in biological and social sciences, in chemistry and physics have tree-like structures from a metric point of view. A number of graph parameters trying to capture this phenomenon and to measure these tree-like structures were proposed; most notable ones being the *tree-stretch*, *tree-distortion*, *tree-length*, *tree-breadth*, Gromov's *hyperbolicity* of a graph, and *cluster-diameter* and *cluster-radius* in a *layering partition* of a graph. If such a parameter is bounded by a constant on graphs then many optimization problems can be efficiently solved or approximated for such graphs. We discuss these parameters and recently established relationships between them for unweighted and undirected graphs; it turns out that all these parameters are at most constant or logarithmic factors apart from each other. We give inequalities describing their relationships and discuss consequences for some optimization problems.

[*] Dedicated to Professor Andreas Brandstädt, on the occasion of his 65th birthday.

Overview of New Approaches for Approximating TSP

Ola Svensson

École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
ola.svensson@epfl.ch

Abstract. In this extended abstract, we survey some of the recent results on approximating the traveling salesman problem on graphic metrics.

We start by briefly explaining the algorithm of Oveis Gharan et al. [1] that has strong similarities to Christofides' famous 3/2-approximation algorithm. We then explain the main ideas behind an alternative approach introduced by Mömke and the author [2]. The new ingredient in our approach is that it allows for the removal of certain edges while simultaneously yielding a connected, Eulerian graph, which in turn leads to a decreased cost. We also overview the exciting developments for TSP on graphic metrics that rapidly followed: an improved analysis of our algorithm by Mucha [3] yielding an approximation guarantee of 1.44, and the recent developments by Sebö and Vygen [3] who gave a 1.4-approximation algorithm.

Finally, we point out some interesting open problems where our techniques currently fall short of applying to more general metrics.

Online Independent Set for Graphs with Bounded Inductive Independence

Berthold Vöcking

Department of Computer Science
RWTH Aachen University
voecking@cs.rwth-aachen.de

In this invited talk, we present techniques and restuls from a joint work with Oliver Göbel, Martin Hoefer, Thomas Kesselheim, and Thomas Schleiden [1]. We study online algorithms for maximum (weight) independent set on graph classes with bounded inductive independence number like interval and disk graphs with applications to, e.g., task scheduling and spectrum allocation. In the considered online setting, it is assumed that nodes of an unknown graph arrive one by one over time. An online algorithm has to decide whether an arriving node should be included into the independent set. We explain that this natural and practically relevant online problem cannot be studied in a meaningful way within a classical competitive analysis as the competitive ratio on worst-case input sequences is lower bounded by $\Omega(n)$. This devastating lower bound holds even for randomized algorithms on unweighted interval graphs and, hence, for the most restricted graph class under consideration.

As a worst-case analysis is pointless, we study online independent set in a stochastic analysis. Instead of focussing on a particular stochastic input model, we present a generic sampling approach that enables us to devise online algorithms achieving performance guarantees for a variety of input models. In particular, our analysis covers stochastic input models like the secretary model, in which an adversarial graph is presented in random order, and the prophet-inequality model, in which a randomly generated graph is presented in adversarial order. Our sampling approach bridges thus between stochastic input models of quite different nature. In addition, we show that the same performance guarantees can be obtained for a period-based input model that is inspired by practical admission control applications.

Using the graph sampling approach, we devise an online algorithm for the unweighted independent set problem with competitive ratio $O(\rho^2)$ for graphs with inductive independence number ρ. This way, we achieve competitive ratio $O(1)$ for independent set on interval and disk graphs. For weighted independent set, we obtain a competitive ratio of $O\left(\rho^2 \log n\right)$ where n denotes the number of nodes. In addition, we show that this bound is almost best possible for interval and disk graphs. In particular, we prove a lower bound on the competitive ratio for weighted independent set on interval graphs of order $\Omega\left(\log n / \log^2 \log n\right)$.

Reference

1. Oliver Göbel, Martin Hoefer, Thomas Kesselheim, Thomas Schleiden, Berthold Vöcking. Online Independent Set Beyond the Worst-Case: Secretaries, Prophets, and Periods. Technical Report. CoRR abs/1307.3192, 2013.

Table of Contents

Tree-Like Structures in Graphs: A Metric Point of View[⋆]

Feodor F. Dragan

Department of Computer Science, Kent State University, Kent, OH 44242, USA
dragan@cs.kent.edu

Abstract. Recent empirical and theoretical work has suggested that many real-life complex networks and graphs arising in Internet applications, in biological and social sciences, in chemistry and physics have tree-like structures from a metric point of view. A number of graph parameters trying to capture this phenomenon and to measure these tree-like structures were proposed; most notable ones being the *tree-stretch*, *tree-distortion*, *tree-length*, *tree-breadth*, Gromov's *hyperbolicity* of a graph, and *cluster-diameter* and *cluster-radius* in a *layering partition* of a graph. If such a parameter is bounded by a constant on graphs then many optimization problems can be efficiently solved or approximated for such graphs. We discuss these parameters and recently established relationships between them for unweighted and undirected graphs; it turns out that all these parameters are at most constant or logarithmic factors apart from each other. We give inequalities describing their relationships and discuss consequences for some optimization problems.

Recent empirical and theoretical work has suggested that many real-life complex networks and graphs arising in Internet applications, in biological and social sciences, in chemistry and physics have tree-like structures from a metric point of view. A number of graph parameters trying to capture this phenomenon and to measure these tree-like structures were proposed; most notable ones being the *tree-stretch* and the *tree-distortion* of a graph, the *tree-length* and the *tree-breadth* of a graph, the Gromov's *hyperbolicity* of a graph, the *cluster-diameter* and the *cluster-radius* in a *layering partition* of a graph.

The *tree-stretch* $\mathsf{ts}(G)$ of a graph $G = (V, E)$ is the smallest number t such that G admits a spanning tree $T = (V, U)$ with $d_T(x, y) \leq t \cdot d_G(x, y)$ for every $x, y \in V$. The *tree-distortion* $\mathsf{td}(G)$ of a graph $G = (V, E)$ is the smallest number α such that G admits a (not necessarily spanning, possibly weighted and having Steiter points) tree $T = (V \cup S, U)$ with $d_G(x, y) \leq d_T(x, y) \leq \alpha \cdot d_G(x, y)$ for every $x, y \in V$. The *tree-length* $\mathsf{tl}(G)$ (resp., *tree-breadth* $\mathsf{tb}(G)$) of a graph G is the smallest number λ such that G admits a Robertson-Seymour's tree-decomposition with bags of diameter (resp., radius) at most λ in G. A graph G is δ-hyperbolic if for any four vertices u, v, w, x, the two larger of the distance

[⋆] Dedicated to Professor Andreas Brandstädt, on the occasion of his 65th birthday.

A. Brandstädt, K. Jansen, and R. Reischuk (Eds.): WG 2013, LNCS 8165, pp. 1–4, 2013.

sums $d(u,v)+d(w,x), d(u,w)+d(v,x), d(u,x)+d(v,w)$ differ by at most 2δ. The hyperbolicity $\mathsf{hb}(\mathrm{G})$ of a graph G is the smallest number δ such that G is δ-hyperbolic.

A *layering* of a graph $G=(V,E)$ with respect to a start vertex s is the decomposition of V into the *spheres* $L^i=\{u\in V : d(s,u)=i\}$, $i=0,1,2,\dots,r$. A *layering partition* $\mathcal{LP}(s)=\{L_1^i,\dots,L_{p_i}^i : i=0,1,2,\dots,r\}$ of G is a partition of each L^i into *clusters* $L_1^i,\dots,L_{p_i}^i$ such that two vertices $u,v\in L^i$ belong to the same cluster L_j^i if and only if they can be connected by a path outside the ball $B_{i-1}(s)$ of radius $i-1$ centered at s. The *cluster-diameter* $\Delta_s(G)$ and the *cluster-radius* $R_s(G)$ in a *layering partition* $\mathcal{LP}(s)$ (with respect to s) of a graph G are defined as follows: $R_s(G)$ is the smallest number r such that for any cluster $C\in\mathcal{LP}(s)$ there is a vertex $v\in V$ with $C\subseteq B_r(v)$; $\Delta_s(G):=\max\{d_G(x,y) : x,y$ belong to the same cluster of $\mathcal{LP}(s)\}$.

Each of these graph parameters provides a measure of how close metrically a given graph is to a tree. If such a parameter is bounded by a constant on a graph G, then many optimization problems on G can be solved or approximated efficiently. Note that for a tree T, $\mathsf{ts}(T)=\mathsf{td}(T)=1$, $\mathsf{tl}(T)=\mathsf{tb}(T)=$ *length of the longest edge in* T, and $\mathsf{hb}(T)=R_s(G)=\Delta_s(G)=0$ (i.e., for trees, one has the smallest possible values for those parameters).

In this talk, we discuss these parameters and recently established relationships between them for unweighted and undirected graphs. It turns out that all these parameters are at most constant or logarithmic factors apart from each other. In particular, the following inequalities hold for any n-vertex unweighted and undirected graph $G=(V,E)$:

1) $\mathsf{tb}(G)\le\mathsf{tl}(G)\le 2\cdot\mathsf{tb}(G)$, $R_s(G)\le\Delta_s(G)\le 2\cdot R_s(G)$ ([folklore]);
2) $\mathsf{hb}(G)\le\mathsf{tl}(G)\le O(\mathsf{hb}(G)\cdot\log n)$ and $\mathsf{hb}(G)\le\Delta_s(G)\le O(\mathsf{hb}(G)\cdot\log n)$ ([4,5]);
3) $\mathsf{ts}(G)\ge\mathsf{td}(G)\ge\frac{1}{3}\Delta_s(G)$ and $\mathsf{td}(G)\le 2\cdot\Delta_s(G)+2$ for every $s\in V$ ([6]);
4) $R_s(G)\le\max\{3\cdot\mathsf{td}(G)-1,\ 2\cdot\mathsf{td}(G)+1\}$ for every $s\in V$ ([6]);
5) $\mathsf{tl}(G)-1\le\Delta_s(G)\le 3\cdot\mathsf{tl}(G)$, $R_s(G)\le 2\cdot\mathsf{tl}(G)$ for every $s\in V$ ([7,8]);
6) $\mathsf{tb}(G)-1\le R_s(G)\le 3\cdot\mathsf{tb}(G)$ ([10]);
7) $\mathsf{tl}(G)\le\mathsf{td}(G)\le\mathsf{ts}(G)$, $\mathsf{tb}(G)\le\lceil\mathsf{ts}(G)/2\rceil$ ([10]);
8) $\mathsf{ts}(G)\le 2\cdot\mathsf{tb}(G)\cdot\log_2 n$ and $\mathsf{ts}(G)\le 2\cdot\mathsf{td}(G)\cdot\log_2 n$ ([10]).

Inequalities in 2) and 3) imply that the tree-distortion $\mathsf{td}(\mathrm{G})$ of a δ-hyperbolic graph G is at most $O(\delta\log n)$. However, a stronger additive version of this result holds [4,5]: Every n-vertex δ-hyperbolic graph $G=(V,E)$ admits an unweighted tree $T=(V,U)$ (without Steiner points), constructible in linear time, such that $d_T(x,y)-2\le d_G(x,y)\le d_T(x,y)+O(\delta\log n)$ for any $x,y\in V$. Furthermore, it is easy to show that any graph G admitting a tree T with $d_G(x,y)\le d_T(x,y)\le d_G(x,y)+r$ for any $x,y\in V$ is r-hyperbolic. So, the hyperbolicity of a graph G is in fact an indicator of an embedabily of G in a tree with an additive distortion. It follows also from the inequalities listed that the tree-stretch $\mathsf{ts}(\mathrm{G})$ of a δ-hyperbolic graph G is at most $O(\delta\log^2 n)$.

While $\mathsf{hb}(\mathrm{G})$, $R_s(G)$, $\Delta_s(G)$ for a given graph G can be computed in polynomial time (in at most $O(n^4)$ time for $\mathsf{hb}(\mathrm{G})$ and in at most $O(nm)$ time for $R_s(G)$ and $\Delta_s(G)$ (see [2,3])) for any n-vertex, m-edge graph G, checking whether ts(G) is at most t and whether tl(G) is at most λ are NP-complete problems in general unweighted graphs for every $t > 3$ [1] and every $\lambda > 1$ [12] (similar NP-completeness results hold also for td(G) and tb(G)). The inequalities listed show that $\Delta_s(G)$ gives a near 3-approximation of tl(G) and of td(G) and an $O(\log n)$-approximation of ts(G), while $R_s(G)$ gives a near 3-approximation of tb(G).

The above inequalities and results provide not only efficiently computable bounds on those parameters but also serve as basis for constructing best approximation algorithms for the corresponding optimization problems which are NP-hard in general. For example, using the relationship between tl(G) and $\Delta_s(G)$ and the fact that a layering partition of a graph G can be constructed in linear time (see [3]), in [8] a linear time algorithm is provided which construct for a given graph G a Robertson-Seymour's tree-decomposition with bags of diameter at most $3 \cdot \mathsf{tl}(G) + 1$. Using the relationship between td(G) and $\Delta_s(G)$, in [6] an efficient 6-approximation algorithm was provided for the problem of minimum distortion embedding of a graph to a tree. The previous approximation bound was 27. Using the relationship between tb(G) and ts(G), in [10] an efficient $(\log_2 n)$-approximation algorithm was provided for the problem of constructing for a given graph G a tree t-spanner with minimum stretch t. Using the relationship between tb(G) and ts(G), [9] discusses also how to "turn", with a slight increase in the number of trees and in the stretch, a multiplicative tree spanner into a small set of collective additive tree spanners (see [11] for the definition).

Acknowledgement. I would like to use this opportunity to thank all people who contributed to this topic.

References

1. Cai, L., Corneil, D.G.: Tree spanners. SIAM J. Discrete Math. 8, 359–387 (1995)
2. Brandstädt, A., Chepoi, V.D., Dragan, F.F.: Distance approximating trees for chordal and dually chordal graphs. Journal of Algorithms 30, 166–184 (1999)
3. Chepoi, V.D., Dragan, F.F.: A note on distance approximating trees in graphs. European Journal of Combinatorics 21, 761–766 (2000)
4. Chepoi, V.D., Dragan, F.F., Estellon, B., Habib, M., Vaxes, Y.: Diameters, centers, and approximating trees of δ-hyperbolic geodesic spaces and graphs. In: Proceedings of the 24th Annual ACM Symposium on Computational Geometry (SoCG 2008), College Park, Maryland, USA, June 9-11, pp. 59–68 (2008)
5. Chepoi, V.D., Dragan, F.F., Estellon, B., Habib, M., Vaxes, Y., Xiang, Y.: Additive Spanners and Distance and Routing Labeling Schemes for δ-Hyperbolic Graphs. Algorithmica 62(3-4), 713–732 (2012)
6. Chepoi, V.D., Dragan, F.F., Newman, I., Rabinovich, Y., Vaxes, Y.: Constant Approximation Algorithms for Embedding Graph Metrics into Trees and Outerplanar Graphs. Discr. & Comput. Geom. 47, 187–214 (2012)
7. Dourisboure, Y., Dragan, F.F., Gavoille, C., Yan, C.: Spanners for bounded tree-length graphs. Theoretical. Computer Science 383, 34–44 (2007)

8. Dourisboure, Y., Gavoille, C.: Tree-decompositions with bags of small diameter. Discr. Math. 307, 2008–2029 (2007)
9. Dragan, F.F., Abu-Ata, M.: Collective Additive Tree Spanners of Bounded Tree-Breadth Graphs with Generalizations and Consequences. In: van Emde Boas, P., Groen, F.C.A., Italiano, G.F., Nawrocki, J., Sack, H. (eds.) SOFSEM 2013. LNCS, vol. 7741, pp. 194–206. Springer, Heidelberg (2013)
10. Dragan, F.F., Köhler, E.: An Approximation Algorithm for the Tree t-Spanner Problem on Unweighted Graphs via Generalized Chordal Graphs. Algorithmica (in print), doi:10.1007/s00453-013-9765-4
11. Dragan, F.F., Yan, C., Lomonosov, I.: Collective tree spanners of graphs. SIAM J. Discrete Math. 20, 241–260 (2006)
12. Lokshtanov, D.: On the complexity of computing tree-length. Discrete Appl. Math. 158, 820–827 (2010)

Overview of New Approaches for Approximating TSP

Ola Svensson

École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
ola.svensson@epfl.ch

Abstract. In this extended abstract, we survey some of the recent results on approximating the traveling salesman problem on graphic metrics.

We start by briefly explaining the algorithm of Oveis Gharan et al. [1] that has strong similarities to Christofides' famous 3/2-approximation algorithm. We then explain the main ideas behind an alternative approach introduced by Mömke and the author [2]. The new ingredient in our approach is that it allows for the removal of certain edges while simultaneously yielding a connected, Eulerian graph, which in turn leads to a decreased cost. We also overview the exciting developments for TSP on graphic metrics that rapidly followed: an improved analysis of our algorithm by Mucha [3] yielding an approximation guarantee of 1.44, and the recent developments by Sebö and Vygen [3] who gave a 1.4-approximation algorithm.

Finally, we point out some interesting open problems where our techniques currently fall short of applying to more general metrics.

Keywords: approximation algorithms, graph theory, traveling salesman problem.

1 The Traveling Salesman Problem and Christofides' Algorithm

In the traveling salesman problem (TSP) we are given n cities with pairwise distances and we wish to find the shortest possible tour that visits each city exactly once. A natural and commonly made assumption is that the distances obey the triangle inequality, i.e., the distance c_{ij} between cities i and j is no longer than the sum of the distances between i and k and between k and j (that is, $c_{ij} \leq c_{ik} + c_{kj}$). The assumption that the distances obey the triangle inequality can be seen to be equivalent to allowing the tour to visit each city at least once instead of exactly once, which in turn is equivalent to finding a connected Eulerian graph of minimum total cost. Recall that an Eulerian graph is a graph where each vertex has even degree. We can thus formulate TSP with metric distances as follows:

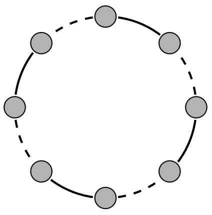
Metric-TSP

Given: A graph $G = (V, E)$ with nonnegative edge costs $\{c_e\}_{e \in E}$.
Find: A connected Eulerian graph $G = (V, E')$ where $E' \subseteq E$ may contain several copies of the same edge so as to minimize $\sum_{e \in E'} c_e$.

Note that with the above definition we can assume that the graph is complete and that the costs satisfy the triangle inequality (as we can always replace an edge by the shortest path between its endpoints).

The metric-TSP is one of the most fundamental NP-hard optimization problems. In spite of a vast amount of research several important questions remain open. While the problem is known to be NP-hard to approximate with a ratio better than 185/184 [4], the best upper bound is still the 1.5-approximation algorithm obtained by Christofides [5] more than three decades ago. Recall that an α-approximation algorithm is a polynomial time algorithm that is guaranteed to find a solution of cost within a factor α of optimum.

We now describe Christofides' beautiful algorithm before continuing to more recent results. While it is deceptively simple, more sophisticated techniques have not yet led to an improved approximation guarantee to date. Christofides' algorithm works in two steps. In the first step, we pick a minimum spanning tree $T = (V, E_T)$ of the graph; this ensures connectivity. In the second step, we find a minimum cost matching M of the vertices of odd degree in T; this ensures that each vertex has even degree. Clearly $E \cup M$ is a feasible solution. We proceed by upper bounding its cost in terms of the optimal cost, denoted by OPT. As removing an edge from the optimal tour gives a spanning tree of cost less than OPT, we have $\sum_{e \in E_T} c_e \leq \mathsf{OPT}$. We shall now finish the analysis by proving that $\sum_{e \in M} c_e \leq \mathsf{OPT}/2$. This follows from observing that the optimal tour induces two perfect matchings (one dashed and one solid) of the odd degree vertices as depicted below:



By using the triangle inequality, one of these matchings must have cost at most $\mathsf{OPT}/2$ and since M was picked to be a minimum matching we have $\sum_{e \in M} c_e \leq \mathsf{OPT}/2$ as required.

In the following sections, we present recent developments that present potential approaches for improving upon the 1.5-approximation guarantee; these approaches are indeed known to perform better on an important special case called graph-TSP. Finally, in Section 4, we point out some interesting open problems where our techniques currently fall short of applying to more general metrics.

2 Graph-TSP and Christofides Algorithm with Sampling

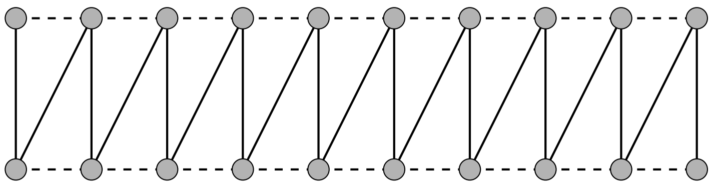
In a major achievement, Oveis Gharan et al. [1] presented an adaptation of Christofides' algorithm with performance guarantee strictly better than 1.5 for the important special case of graph-TSP defined as follows:

Graph-TSP

Given: A graph $G = (V, E)$ with unit edge costs.
Find: A connected Eulerian graph $G = (V, E')$ where $E' \subseteq E$ may contain several copies of the same edge so as to minimize $\sum_{e \in E'} c_e = |E'|$.

Note that the above definition is equivalent to the metric-TSP except that all edges have the same cost. One can see that this is equivalent to the metric-TSP on shortest path metrics of unweighted graphs. In contrast to TSP on Euclidean metrics and to TSP on planar graphs that admit PTASs (see [6,7] for Eucledian metrics and [8,9] for planar graphs), graph-TSP seems to capture the difficulty of the metric-TSP in the sense that, as stated in [9], it is APX-hard and the lower bound 4/3 on the integrality gap of the standard linear programming relaxation is established using a graph-TSP instance. In fact, there is a famous conjecture stating that the standard relaxation approximates metric-TSP within a factor 4/3 (see e.g. [10]). We note that all of the results discussed in this extended abstract also upper bounds the integrality gap of the relaxation for graph-TSP and thus narrows the gap between the upper and lower bounds. To summarize, the motivation for studying graph-TSP stems from the fact that it captures many of the difficulties of metric-TSP but at the same time, it is easier to argue about. The formulation is also more graph theoretic (no weights) and therefore perhaps appeals more to researchers in graph theory.

Let us now return to the approach of Oveis Gharan et al. [1]. As aforementioned, their algorithm is inspired by Christofides' algorithm. To motivate their approach it is instructive to look at instances where Christofides' algorithm fails to achieve a better approximation guarantee than 1.5. Such a family of graphs is depicted below:



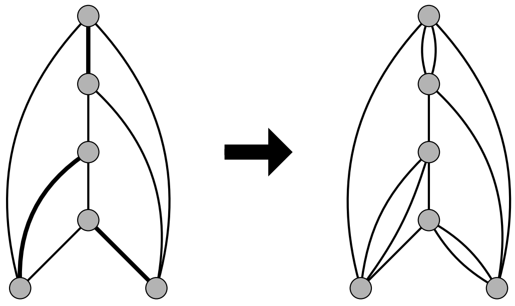
Note that an optimal tour has length n (the number of vertices) by using the edges along the "border". However, if we in the first step of Christofides' algorithm pick the minimum spanning tree consisting of the solid edges then the algorithm will return a tour of a cost that approaches $1.5n$ when the number of vertices tend to infinity.

What went wrong and how can we fix this? By closer inspection, we can see that the tour of high cost was returned because we picked a "bad" spanning tree in the first step. This leads to the natural idea that instead of always picking a minimum spanning tree let us pick one at random. Specifically, Oveis Gharan et al. [1] modify Christofides' algorithm by changing the first step of the algorithm: instead of picking a minimum spanning tree they use the standard linear programming relaxation to sample a random spanning tree according to the maximum entropy distribution. They then use several interesting properties of this distribution (it belongs to a class of measures called strongly Rayleigh) and the structure of near-min cuts. The very sophisticated analysis of the rather simple algorithm shows that it is a $1.5 - \epsilon$-approximation algorithm for graph-TSP, where $\epsilon > 0$ is a small constant. The details of their method is beyond the scope of this extended abstract and we refer the reader to their paper [1]. We would also like to mention that a similar "sampling" idea was previously used to get the current best $O(\log n / \log\log n)$-approximation algorithm for asymmetric TSP [11]. In the next section, we will see an alternative method for approximating TSP that perhaps leads to slightly more complicated algorithms but in return a simpler analysis with better guarantees for graph-TSP.

3 Finding a Tour by Deleting Edges

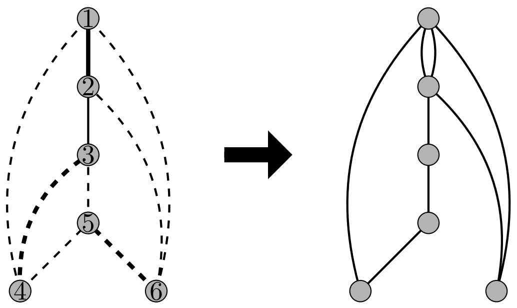
In Christofides' algorithm (and in the modified version of the last section) we first ensure connectivity by picking a spanning tree and then fix the parity of the degrees by adding a perfect matching of the vertices of odd degree. We shall now explain an alternative method that was introduced by Mömke and the author [2] that is inspired by earlier works that related the cost of a 2-edge connected graph to the length of a tour by Fredrickson and Ja'Ja' [12] and Monma, Munson, and Pulleyblank [13].

To explain the main ideas in a simple setting, we restrict our discussion to cubic 2-edge connected graphs; that is, graphs where each vertex has degree 3 and at least 2-edges need to be removed in order to disconnect the graph. It is well known that for such graphs there exists a distribution μ of perfect matchings so that the probability that an edge e is in a randomly picked matching is exactly 1/3. Given this result, a natural way to obtain a connected Eulerian graph given a cubic 2-edge connected graph $G = (V, E)$ is as follows: pick a perfect matching M according to μ and return $E \cup M$. An example of the algorithm is depicted below where the picked perfect matching is depicted in bold and the resulting Eulerian graph is on the right-hand-side:



As we add a perfect matching to a cubic graph, each vertex in $(V, E \cup M)$ will have degree 4 so it is Eulerian and it is connected (as (V, E) is connected). However, the cost of the tour will be $|E| + |M| = 3n/2 + n/2 = 2n$, which might only approximate the optimal tour within a factor of 2.

Once again we can ask ourselves what went wrong and how to improve the algorithm. The key observation is that instead of adding a copy of every edge in the matching we might remove some of them to obtain an Eulerian graph of smaller cost. The simplest algorithm would simply return $(V, E \setminus M)$ where each vertex has degree 2 so it is Eulerian but the problem is that it might not be connected. To ensure connectivity, we introduce the idea of "removable pairing" in [2] which specifies the edges that can be safely removed without disconnecting the graph. Instead of giving the formal definition here, we give the high level idea. Consider a cubic 2-edge connected graph $G = (V, E)$. It is well-known that the vertices of such a graph can be ordered $v_1, v_2, \ldots, v_n$ such that for $i = 2, \ldots, n-1$, v_i has a neighbor v_k to the left (i.e., $k < i$) and a neighbor v_ℓ to the right (i.e., $\ell > i$)[1]. Now direct each edge to the incident vertex that appears later in the ordering and let the set $R \subseteq E$ of removable edges be those edges that are directed to a vertex of indegree at least 2. See the left-hand-side of the figure below for an example; the vertices are numbered according to their ordering and the removable edges are dashed. The algorithm now proceeds by picking a matching according to μ and then it outputs the graph $(V, E \cup (M \setminus R) \setminus (M \cap R))$. An example of the execution is below where the fat edges depict the picked matching:



Again one can see that each vertex has degree 2 or 4 so the graph is Eulerian. Moreover, using that at most one edge incident to every vertex is taken in M one can prove that the graph stays connected. Finally, the expected cost of the tour is $\mathbb{E}_{M \in \mu}[|E| + M \setminus R - M \cap R] = |E| + \mathbb{E}[M \setminus R] - \mathbb{E}[M \cap R] = 4/3|E| - 2/3|R| = 2n - 2/3 \cdot (n+1) = 4n/3 - 2/3$ where we used that $|R| = n+1$ and $\Pr_{M \in \mu}[e \in M] = 1/3$ for every edge e.

In summary, we have used the idea that one can delete edges to correct parity instead of just adding them. Very similar methods yield the tight result that any subcubic (every vertex has degree at most 3) 2-edge connected graph has a tour of length at most $4n/3 - 2/3$ which settles a conjecture by Boyd et al. [14] in

[1] This is called an st-numbering of a graph.

the affirmative. Boyd et al. proved that any cubic 2-edge connected graph has a tour of length $4n/3$ improving upon a previous results by Gamarnik et al. [15].

For general graphs, our algorithm gets more involved and in the original paper we solve a min-cost circulation problem to find a large set of removable edges. This was first proved to give a 1.461-approximation algorithm for graph-TSP and the analysis was later improved by Mucha [3] who showed that our algorithm yields a 1.44-approximation algorithm. The current best 1.4-approximation algorithm for graph-TSP by Sebö and Vygen [16] uses Ear decompositions together with the concept of removable edges in a very clever way. To keep the discussion brief we refer the reader to those papers and also to the recent survey by Vygen [17].

4 Some Open Problems

A major open problem is to give a 4/3-approximation algorithm for metric-TSP. Natural subgoals include that of giving a $1.5 - \epsilon$-approximation algorithm for metric-TSP and that of giving a 4/3-approximation algorithm for graph-TSP. Our algorithm in [2] may actually give a better approximation guarantee for graph-TSP than 1.44. Indeed, for so-called half-integral solutions we show that its approximation guarantee is 4/3 which is tight with respect to the standard linear programming relaxation.

As a final note, let me point out a natural special case of metric-TSP where our current techniques fall short. Given a cubic 3-edge connected graph $G = (V, E)$ with weights $\{w_v\}_{v \in V}$ on the vertices, can you find a connected Eulerian graph $G = (V, E')$ so that $\sum_{\{u,v\} \in E'} \frac{w_u + w_v}{2} \leq \frac{4}{3} \sum_{v \in V} w_v$? If all the vertices have the same weight this is simply graph-TSP but if they have different weights the problem becomes more complex: we are currently unable to give a better guarantee than 1.5, which is simply implied by Christofides' algorithm from 1976.

References

1. Oveis Gharan, S., Saberi, A., Singh, M.: A randomized rounding approach to the traveling salesman problem. In: [18], pp. 550–559
2. Mömke, T., Svensson, O.: Approximating graphic TSP by matchings. In: [18], pp. 560–569
3. Mucha, M.: 13/9-approximation for graphic TSP. In: Dürr, C., Wilke, T. (eds.) STACS. LIPIcs, vol. 14, pp. 30–41. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012)
4. Lampis, M.: Improved inapproximability for TSP. In: Gupta, A., Jansen, K., Rolim, J., Servedio, R. (eds.) APPROX/RANDOM 2012. LNCS, vol. 7408, pp. 243–253. Springer, Heidelberg (2012)
5. Christofides, N.: Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University (1976)
6. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. Journal of the ACM 45(5), 753–782 (1998)

7. Mitchell, J.S.B.: Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. SIAM Journal on Computing 28(4), 1298–1309 (1999)
8. Arora, S., Grigni, M., Karger, D.R., Klein, P.N., Woloszyn, A.: A polynomial-time approximation scheme for weighted planar graph TSP. In: Karloff, H.J. (ed.) SODA, pp. 33–41. ACM/SIAM (1998)
9. Grigni, M., Koutsoupias, E., Papadimitriou, C.H.: An approximation scheme for planar graph TSP. In: Proc. 36th Annual Symposium on Foundations of Computer Science (FOCS 1995), pp. 640–645. IEEE Computer Society (1995)
10. Goemans, M.X.: Worst-case comparison of valid inequalities for the TSP. Mathematics and Statistics 69(1), 335–349 (1995)
11. Asadpour, A., Goemans, M.X., Madry, A., Gharan, S.O., Saberi, A.: An O(log n/ log log n)-approximation algorithm for the asymmetric traveling salesman problem. In: Charikar, M. (ed.) SODA, pp. 379–389. SIAM (2010)
12. Frederickson, G.N., Jájá, J.: On the relationship between the biconnectivity augmentation and travelling salesman problems. Theoretical Computer Science 19(2), 189–201 (1982)
13. Monma, C.L., Munson, B.S., Pulleyblank, W.R.: Minimum-weight two-connected spanning networks. Mathematical Programming 46(1), 153–171 (1990)
14. Boyd, S., Sitters, R., van der Ster, S., Stougie, L.: TSP on cubic and subcubic graphs. In: Günlük, O., Woeginger, G.J. (eds.) IPCO 2011. LNCS, vol. 6655, pp. 65–77. Springer, Heidelberg (2011)
15. Gamarnik, D., Lewenstein, M., Sviridenko, M.: An improved upper bound for the TSP in cubic 3-edge-connected graphs. Operations Research Letters 33(5), 467–474 (2005)
16. Sebő, A., Vygen, J.: Shorter tours by nicer ears: 7/5-approximation for graphic TSP, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. CoRR abs/1201.1870 (2012)
17. Vygen, J.: New approximation algorithms for the TSP. In: OPTIMA, vol. 90, `http://www.mathopt.org/Optima-Issues/optima90.pdf`
18. Ostrovsky, R. (ed.): IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25. IEEE (2011)

Linear Rank-Width and Linear Clique-Width of Trees

Isolde Adler[1,⋆] and Mamadou Moustapha Kanté[2,⋆⋆]

[1] Institut für Informatik, Goethe-Universität, Frankfurt, Germany
iadler@informatik.uni-frankfurt.de
[2] Clermont-Université, Université Blaise Pascal, LIMOS, CNRS, France
mamadou.kante@isima.fr

Abstract We show that for every forest T the linear rank-width of T is equal to the path-width of T, and we show that the linear clique-width of T equals the path-width of T plus two, provided that T contains a path of length three. It follows that both linear rank-width and linear clique-width of forests can be computed in linear time. Using our characterization of linear rank-width of forests, we determine the set of minimal excluded acyclic vertex-minors for the class of graphs of linear rank-width at most k.

1 Introduction

Rank-width [29] is a graph parameter introduced by Oum and Seymour with the goal of efficient approximation of the *clique-width* [9] of a graph. *Linear rank-width* can be seen as the linearized variant of rank-width, similar to path-width, which can be seen as the linearized variant of tree-width. While path-width is a well-studied notion, much less is yet known about linear rank-width. Indeed, any graph of k-bounded path-width has k-bounded linear rank-width, but conversely the difference is unbounded. For example, the class of all complete (bipartite) graphs has linear rank-width at most 1, but unbounded path-width. *Linear clique-width*, a linearized version of clique-width, was introduced independently by several authors when studying the computational complexity of clique-width (see for instance the works by Gurski et al. [11,13,14,15,16], and the paper [27] by Lozin and Rautenbach). The computation of the linear clique-width of some graph classes have been investigated by Heggernes et al. [17,18,19]. Linear rank-width is equivalent to linear clique-width in the sense that any graph class has bounded linear clique-width if and only if it has bounded linear rank-width.

Computing linear rank-width is NP-complete in general. In fact, it is proved in [11] that computing linear clique-width is NP-complete and one can easily reduce the computation of linear clique-width to the computation of linear rank-width. Moreover, very little is known about efficient computation of linear rank-width on restricted graph classes. The only known results are for special types

⋆ Supported by the German Research Council, Project GalA, AD 411/1-1.
⋆⋆ Supported by the French Agency for Research under the DORSO project.

of graphs, such as for complete (bipartite) graphs, and linear clique-width is known to be polynomial time computable on *thickend paths* [19] and *k-path powers* [17]. Even for the very natural class of forests efficient computability was open. In contrast, many classes are known that allow efficient computation of path-width [3,4,5,10,12,24,28,31].

In this paper, we provide the first non-trivial graph class on which linear rank-width can be computed in polynomial (even linear) time. We prove

Theorem 1. *Linear rank-width and linear clique-width of forests can be computed in linear time.*

We obtain Theorem 1 as a corollary of the following theorems.

Theorem 2. *The linear rank-width of any forest equals its path-width.*

Theorem 3. *Let T be a tree. If T contains a path of length* 3*, then* $\operatorname{lcw}(T) = \operatorname{pw}(T) + 2$*. Otherwise,* $\operatorname{lcw}(T) = \operatorname{pw}(T) + 1$*.*

While it was known that the class of all trees has unbounded linear rank-width (see [12] for a combinatorial proof) and unbounded path-width, Theorem 2 is somewhat surprising, because it actually equates the two structurally very different parameters.

It is known that the linear clique-width of any graph is bounded by its path-width plus 2 [11]. Since linear rank-width is bounded by linear clique-width, the same bound carries over to linear rank-width. We show that the linear rank-width of any graph is bounded by its path-width. This is not hard to prove, but it seems it was not written down yet. For forests we show that the converse holds, too. Our proof uses the characterization of path-width by the cops and invisible robber game [23]. Given an ordering of the vertices of a forest T witnessing the linear rank-width of T, we construct a winning strategy for the cops. Here it is not sufficient for the cops to search the vertices according to the given ordering, but a more involved strategy yields the result. Indeed, our proof method is constructive in the sense that it shows how to transform the given ordering into a winning strategy for the cops (and a path decomposition).

It is known that the (linear) rank-width does not increase when taking *vertex-minors*, and, given k, the set of minimal excluded vertex-minors for the class of graphs of rank-width at most k is known to be computable [20]. However, until now, explicit sets of minimal excluded vertex-minors are only known for circle graphs [7], distance-hereditary graphs [20], and for graphs of linear rank-width at most one [1]. For graphs of linear rank-width at most k, some minimal excluded vertex-minors were established in [22]. Using Theorem 2, we determine the set of minimal excluded acyclic vertex-minors for linear rank-width k. It turns out that they coincide with the minimal excluded minors for graphs of path-width at most k that are acyclic [32].

Summary. Section 2 introduces the terminology and the notions of linear rank-width, path-width and the cops and invisible robber game. In Section 3 we prove that linear rank-width and path-width coincide on forests (Theorem 2), and in Section 4 we prove Theorem 3 characterizing the linear clique-width of forests. In Section 5 we give the set of minimal excluded acyclic vertex-minors for the class of graphs of linear rank-width k, and we conclude with Section 6.

2 Preliminaries

For a set A we denote the power set of A by 2^A. We let $A \setminus B := \{x \in A \mid x \notin B\}$ denote the *difference* of two sets A and B. For a subset X of a ground set A let $\overline{X} := A \setminus X$. For two sets A and B let $A \Delta B := (A \setminus B) \cup (B \setminus A)$ denote the *symmetric difference* of A and B. For an integer $n > 0$ we let $[n] := \{1, \ldots, n\}$.

In this paper, graphs are finite, simple and undirected, unless stated otherwise. Let G be a graph. We denote the vertex set of G by $V(G)$ and the edge set by $E(G)$. We regard edges as two-element subsets of $V(G)$. For a vertex $v \in V(G)$ we let $N_G(v) := \{u \in V(G) \mid u \neq v,\ \{v,u\} \in E(G)\}$ denote the set of *neighbors* of v (in G). The *degree* of v (in G) is $\deg_G(v) := |N_G(v)|$. A partition of $V(G)$ into two sets X and Y with $X \dot\cup Y = V(G)$ is called a *cut* in G. We denote it by (X,Y). A *tree* is a connected, acyclic graph. A *leaf* of a tree is a vertex of degree one. A *path* is a tree where every vertex has degree at most two. The *length* of a path is the number of its edges. The *distance* between two vertices $u,v \in V(G)$ is the length of a shortest path from u to v. A *rooted tree* is a tree with a distinguished vertex r, called the *root*. The *height* of a rooted tree is the maximal length of a path from the root to a leaf (counted in terms of edges). Let T be a rooted tree with root r. Let $v \in V(T)$. The tree T^v is the subtree of T induced by those vertices $u \in V(T)$ such that the path from r to u contains v. For a rooted tree T it is sometimes convenient to orient the edges of T in the direction away from the root, thus obtaining an *oriented tree*.

Path-Width. A *path decomposition* of a graph G is a pair (P,B), where P is a path and $B = (B_t)_{t \in V(P)}$ is a family of subsets $B_t \subseteq V(G)$, satisfying

1. For every $v \in V(G)$ there exists a $t \in V(P)$ such that $v \in B_t$.
2. For every $e \in E(G)$ there exists a $t \in V(P)$ such that $e \subseteq B_t$.
3. For every $v \in V(G)$ the set $\{t \in V(P) \mid v \in B_t\}$ is connected in P.

The *width* of a path decomposition (P,B) is defined as $\mathrm{w}(P,B) := \max\{|B_t| \mid t \in V(P)\} - 1$. The *path-width* of G is defined as

$$\mathrm{pw}(G) := \min\{\mathrm{w}(P,B) \mid (P,B) \text{ is a path decomposition of } G\}.$$

Paths have path-width ≤ 1. Indeed, the graphs of path-width ≤ 1 are precisely the disjoint unions of *caterpillars*, i.e. of the graphs that contain a path P such that every vertex has distance at most one to some vertex of P. There is no finite upper bound on the path-width of trees. Indeed, the rooted binary tree T_h of height h satisfies $\mathrm{pw}(T_h) = \lceil h/2 \rceil$ [30].

A path decomposition (P,B) of G is *small* if any two distinct vertices $t,t' \in V(P)$ satisfy $B_t \not\subseteq B_{t'}$. The following lemma is not hard to prove.

Lemma 4. *Any graph G has a small path decomposition of width* $\mathrm{pw}(G)$. $\square$

Linear Rank-Width. For sets R and C an (R, C)*-matrix* is a matrix where the rows are indexed by elements in R and columns indexed by elements in C. (Since we are only interested in the rank of matrices, it suffices to consider matrices up to permutations of rows and columns.) For an (R, C)-matrix M, if $X \subseteq R$ and $Y \subseteq C$, we let $M[X, Y]$ be the submatrix of M where the rows and the columns are indexed by X and Y respectively. If M is an (R, C)-matrix and when the context is clear we will identify the row indexed by $x \in R$ with x (similarly for the column indexed by $y \in C$); hence we will say for instance that a subset X of R is a basis for the rows of M if the rows indexed by X form a basis for the rows of M and similarly for other linear algebra terminologies involving rows (or columns).

Let A_G be the adjacency $(V(G), V(G))$-matrix of G. For a graph G, let $v_1, \ldots, v_n$ be a linear ordering of $V(G)$. Every index $i \in [n]$ induces a cut $(X_i, \overline{X_i})$, where $X_i = \{v_1, \ldots, v_i\}$. The *cutrank* of the ordering $v_1, \ldots, v_n$ is defined as

$$\mathrm{cutrk}_G(v_1, \ldots, v_n) := \max\{\mathrm{rk}(A_G[X_i, \overline{X_i}]) \mid i \in [n]\}.$$

The *linear rank-width* of G is defined as

$$\mathrm{lrw}(G) := \min\{\mathrm{cutrk}_G(v_1, \ldots, v_n) \mid v_1, \ldots, v_n \text{ is a linear ordering of } V(G)\}.$$

Disjoint unions of caterpillars have linear rank-width ≤ 1. Ganian [12] gives an alternative characterization of the graphs of linear rank-width ≤ 1 as *thread graphs*. In addition, he proves that there is no finite upper bound on the linear rank-width of trees.

The Cops and Invisible Robber Game. We now introduce the *cops and invisible robber game* characterizing path-width. Let G be a graph and let $k \geq 0$ be an integer. The *cops and invisible robber game* on G (with game parameter k) is played by two players, the *cop player* and the *robber player*, on the graph G. The cop player controls k cops and the robber player controls the robber. Both the cops and the robber move on the vertices of G. Some of the cops move to at most k vertices and the robber stands on a vertex r not occupied by the cops. At all times, the robber is invisible to the cops. Initially, no cops occupy vertices and the robber chooses a vertex to start playing. In each move, some of the cops fly in helicopters to at most k new vertices. During the flight, the robber sees which position the cops are approaching and before they land she quickly tries to escape by running arbitrarily fast along paths of G to a vertex r', not being allowed to run through a vertex occupied by a cop. Hence, if $X \subseteq V(G)$ is the cops' position, the robber stands on $r \in V(G) \setminus X$, and after the flight, the cops occupy the set $Y \subseteq V(G)$, then the robber can run to any vertex r' within the connected component of $G \setminus (X \cap Y)$ containing r. The cops win if they land a cop via helicopter on the vertex occupied by the robber. The robber wins if she can always elude capture. A *play* is a sequence of cop positions $X_0, X_1, X_2, \ldots$ with $X_0 := \emptyset$ and $|X_i| \leq k$ for all i. At each step of a play, we can describe the set of *cleared* vertices as follows. At the position X_0, the set of cleared vertices is $A_0 := \emptyset$. After the cops' move to X_i (for $i > 0$), the set of cleared vertices is

$$\begin{aligned} A_i := (A_{i-1} \cup X_i) \setminus \{r \in V(G) \mid & \text{ there is a path from } V(G) \setminus A_{i-1} \\ & \text{to } r \text{ in } G \setminus (X_{i-1} \cap X_i)\}. \end{aligned}$$

Winning strategies are defined in the usual way. The *invisible cop-width* of G, $\mathrm{icw}(G)$, is the minimum number of cops having a winning strategy on G.

A winning strategy for the cops is *monotone*, if for any play $X_1, X_2, X_3, \ldots$ played according to the strategy, the sets $A_0, A_i, A_2, \ldots$ form a non-decreasing sequence (with respect to $\subseteq$). The *monotone invisible cop-width* of G, $\mathrm{monicw}(G)$, is the minimum number of cops having a monotone winning strategy on G.

Theorem 5 ([2,26]). *Any graph G satisfies* $\mathrm{pw}(G)+1 = \mathrm{icw}(G) = \mathrm{monicw}(G)$. □

3 Linear Rank-Width and Path-Width

In this section we prove that on forests, linear rank-width and path-width coincide. Due to space constraints some proofs are omitted.

Lemma 6. *Any graph G satisfies* $\mathrm{lrw}(G) \leq \mathrm{pw}(G)$.

Definition 7. *Let G be a graph and let (X, Y) be a cut in G. A vertex $x \in X$ is a* standard vertex (of the cut) *if x has exactly one neighbor in Y.*

Fact 8. *Let T be a tree and let (X, Y) be a cut in T.*

1. *Any two distinct rows of $A_T[X, Y]$ have at most one common non-zero position.*
2. *Let $B \subseteq X$ be a basis of the rows of M. A vertex $x \in X \setminus B$ cannot be generated by less than $|N_T(x) \cap Y|$ elements of B.*

Lemma 9 (Spanning dependent vertices). *Let T be a tree and let (X, Y) be a cut in T. Let $B \subseteq X$ be a basis of the row space of $A_T[X, Y]$. For $x \in X \setminus B$ with $N_T(x) \cap Y \neq \emptyset$ let $B' \subseteq B$ be the (unique) minimal subset of B spanning x. We let T', called B-*basic *tree of x, be the bipartite subgraph of T with vertex set $V(T') = X' \dot\cup Y'$, where $X' := B' \cup \{x\}$ and $Y' := N_T(B' \cup \{x\}) \cap Y$, and with edge set $E(T') := \{\{u, v\} \in E(T) \mid u \in X',\ v \in Y'\}$. Then*

1. *T' is a tree.*
2. *The leaves of T' are standard vertices in X.*
3. *The vertices in Y' have degree two in T'.*
4. *Choose x to be the root of T' and orient the edges of T' away from the root. Let $b\colon B' \to Y'$ where for every $z \in B'$ we let $b(z)$ be the predecessor of z in T' oriented. Then b is a bijection, and hence $|Y'| = |B'|$.*

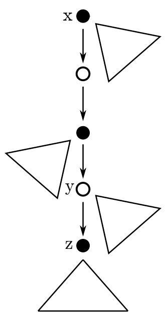


Fig. 1. The tree T' in the proof of Lemma 9. Black vertices are in X', white vertices in Y'.

Lemma 10 (Clearing dependent vertices). *Under the conditions of Lemma 9, suppose that in the $(k+1)$-cops and robber game on T the cops have cleared all vertices in $X \setminus \{x\}$ and the game is in a position where at most k cops are occupying vertices. Furthermore, assume that exactly $|B'|$ cops are occupying vertices of T', and in addition, for each vertex $b \in B'$, either b is occupied by a cop, or $N_T(b) \cap Y$ is occupied by cops. Then there is a sequence of moves of $|B'|+1$ cops, involving only the cops on vertices of T' plus one additional cop, that ends in a position, where*

1. *the vertices in $X \cup V(T') \setminus \{x\}$ are cleared,*
2. *all vertices in $N_T(x) \cap Y$ are occupied,*
3. *exactly $|B'|$ cops occupy vertices of T', and*
4. *the set $N_T(B') \cap Y$ is occupied by cops.*

Theorem 11. *Any forest T satisfies* $\mathrm{pw}(T) \leq \mathrm{lrw}(T)$.

Proof. We may assume that T is a tree. Let $v_1, \ldots, v_n$ be a linear ordering of $V(T)$ witnessing $k := \mathrm{lrw}(T)$. For $i \in [n]$ let $X_i := \{v_1, \ldots, v_i\}$ and $Y_i := \{v_{i+1}, \ldots, v_n\}$, and let M_i be $A_T[X_i, Y_i]$.

We describe a strategy for $k+1$ cops in the invisible robber and cops game. The strategy follows the linear ordering of $V(T)$. For each new vertex v_i that has to be cleared, we describe a *transition* – a finite sequence of cop moves to make sure that v_i is cleared. After the ith transition, the following invariants hold.

1. Every vertex in X_i is cleared.
2. There is a basis $B_i \subseteq X_i$ of the rows of M_i such that each $b \in B_i$ satisfies:

   b is occupied by a cop or $N_T(b) \cap Y_i$ is occupied by cops, and no vertex in the set $X_i \setminus B_i$ is occupied by a cop.

3. The cops occupy exactly $|B_i|$ vertices.

The first $\leq k$ transitions simply consist in placing cops on the vertices $v_1, \ldots, v_\ell$, with $\ell \leq k$, successively, where ℓ is the greatest index $i \leq k$ such that the rank of M_i is equal to i. Obviously, after each such transition the invariants hold.

Suppose we have completed the ith transition, and we want to make the $(i+1)$st transition. Moving from M_i to M_{i+1}, the following cases can occur.

(a) In M_{i+1}, the new vertex v_{i+1} is in the span of B_i.
(b) In M_{i+1}, the new vertex v_{i+1} is linearly independent of B_i.

Observe that B_i can span the rows of M_{i+1}, but may be linearly dependent in M_{i+1}. If it is linearly dependent in M_{i+1}, then the size of a maximum linearly independent subset of B_i is $|B_i|-1$, because deleting a column can only decrease the rank by one.

Claim 1: If the size of a maximum linearly independent subset of B_i in M_{i+1} is $|B_i| - 1$, then there exists a vertex $v_N \in N_T(v_{i+1}) \cap B_i$ such that $B_i \setminus \{v_N\}$ is a maximum linearly independent subset of B_i in M_{i+1}.

Proof of the Claim: If B_i is linearly dependent in M_{i+1} and linearly independent in M_i, there exists a row of M_i corresponding to a vertex $u \in B_i$ that is generated by $B_i \setminus \{u\}$ and that has a 1 at the column corresponding to v_{i+1}, and hence $u \in N_T(v_{i+1})$. $\dashv$

We will complete the $(i+1)$st transition in such a way that the new basis B_{i+1} of the row space of M_{i+1} contains a basis of the rows of M_{i+1} corresponding to B_i, together with the vertex v_{i+1}, if v_{i+1} is linearly independent of B_i in M_{i+1}. For this, let $v_N \in N_T(v_{i+1}) \cap B_i$ be as in Claim 1. If v_{i+1} is in the span of B_i, we let $B_{i+1} := B_i \setminus v_N$. Otherwise, we let $B_{i+1} := (B_i \setminus v_N) \cup \{v_{i+1}\}$. Obviously, B_{i+1} is a basis of M_{i+1}.

For each v spanned by B_{i+1} let T_v denote the B_{i+1}-basic tree of v. The following follows from the fact that T is a tree and the vertex v_N is adjacent to v_{i+1}.

Claim 2: let $v_N \in N_T(v_{i+1}) \cap B_i$ be as in Claim 1.

(i) The B_{i+1}-basic tree of v_N does not contain v_{i+1}.
(ii) $V(T_{v_{i+1}}) \cap V(T_{v_N}) = \emptyset$ and there is no edge other than $\{v_N, v_{i+1}\}$ between a vertex of $T_{v_{i+1}}$ and a vertex of T_{v_N}. $\dashv$

We identify two cases, depending on whether a cop occupies v_{i+1}.

Case 1. After the ith transition, v_{i+1} is not occupied by a cop.
Then by the inductive invariant (1), the set $N_T(v_{i+1}) \cap X_{i+1} = N_T(v_{i+1}) \cap X_i$ is occupied by cops, and hence $N_T(v_{i+1}) \cap X_i \subseteq B_i$ by the inductive invariant (2).

Case 1.1 Vertex v_{i+1} is in the span of B_i in M_{i+1}.
If v_{i+1} has no neighbors in Y_{i+1}, then we use the $(k+1)$st cop to step on v_{i+1} and remove the cop again. Otherwise, let T' be the B_{i+1}-basic tree of v_{i+1}, and let $B' \subseteq B_{i+1}$ be the minimal subset of B_{i+1} spanning v_{i+1}. Since T has no

cycles, $V(T') \cap \big(N_T(v_{i+1}) \cap X_{i+1}\big) = \emptyset$. Hence we can use Lemma 10 to move to $N_T(v_{i+1}) \cap Y_{i+1}$ with at most $|B'| + 1 \leq k + 1$ cops, ending in a position where at most k cops are on $V(T)$. Since $N_T(v_{i+1}) \cap X_{i+1}$ is occupied by cops, we can use the $(k + 1)$st cop to step on v_{i+1} and then lift the $(k + 1)$st cop up again, thus clearing v_{i+1}. We have then cleared X_{i+1}.

It remains to check conditions (2) and (3). By the inductive hypothesis invariant, (2) is already satisfied, and if B_i is linearly independent in M_{i+1}, condition (3) is also satisfied. So assume B_i is linearly dependent in M_{i+1}. If v_N does not have a neighbor in Y_{i+1} we can remove safely the cop from v_N. Otherwise, if it has a neighbor in Y_{i+1}, we can use Lemma 10 to move to $N_T(v_N) \cap Y_{i+1}$, and we then lift up the cop from v_N. By Claim 2, we can do it safely. In this way, we end the transition with a position of $|B_{i+1}|$ cops on $V(T)$. This follows from Lemma 10(3) and Claim 2. Hence all three invariants are satisfied.

Case 1.2. Vertex v_{i+1} is not in the span of B_i in M_{i+1}.
If v_N has no neighbors in Y_{i+1}, we place the $(k + 1)$st cop on v_{i+1} (v_{i+1} is not already occupied by a cop) and we then remove the cops from v_N. After these moves, at most k cops are occupying vertices.

Now, if v_N has a neighbor in Y_{i+1}, take the B_{i+1}-basic tree T_{v_N} of v_N and use Lemma 10 to move cops in $V(T_{v_N}) \setminus \{v_N\}$ to $N_T(v_N) \cap Y_{i+1}$. Claim 2 guarantees the safety of these moves. After these moves, at most k cops are occupying vertices. If v_{i+1} was occupied by a cop, then remove the cop that is still occupying the vertex v_N. If v_{i+1} was not occupied by a cop, then we place the $(k+1)$st cop on v_{i+1} and remove the cop that occupy the vertex v_N. After these moves, v_{i+1} is cleared and since we did not recontaminate X_i, X_{i+1} is cleared. Moreover, exactly $|B_{i+1}|$ vertices of T are occupied by cops (Lemma 10(3) and Claim 2), and since the other cops are not moved, invariant (2) is satisfied. The three invariants are hence satisfied.

Case 2. After the ith transition, v_{i+1} is occupied by a cop.
By the inductive invariant (1), each vertex $b \in N_T(v_{i+1}) \cap X_{i+1} = N_T(v_{i+1}) \cap X_i$ is cleared, hence either b is occupied by a cop, or $N_T(b) \cap Y_{i+1}$ is occupied by cops.

Case 2.1. Vertex v_{i+1} is in the span of B_i in M_{i+1}.
For every $b \in \{v_N, v_{i+1}\}$ such that $V(T_b) \cap Y_{i+1}$ contains an unoccupied vertex, we use Lemma 10 to move cops in $V(T_b) \setminus \{b\}$ to $V(T_b) \cap Y_{i+1}$. This is possible, because the B_{i+1}-basic trees involved are pairwise disjoint and pairwise connected via v_{i+1} only (Claim 2). After that, we remove the cops occupying vertices in $\{v_N, v_{i+1}\}$. Since by induction, the cop moves are monotone, we can conclude that the three inductive invariants are satisfied.

Case 2.2. Vertex v_{i+1} is not in the span of B_i in M_{i+1}.
If $V(T_{v_N}) \cap Y_{i+1}$ contains an unoccupied vertex, we use Lemma 10 to move cops in $V(T_{v_N}) \setminus \{v_N\}$ to $V(T_{v_N}) \cap Y_{i+1}$. After that, we remove the cop occupying the vertex v_N. Since by induction, the cop moves are monotone, we can conclude that the three inductive invariants are satisfied. □

Note that the analogous statement of Theorem 11 fails for C_3, the cycle of length three. While $\mathrm{lrw}(C_3) = 1$, we have $\mathrm{pw}(C_3) = 2$.

Theorem 2 now follows from Lemma 6 and Theorem 11. Theorem 2 combined with [10] gives the following as a corollary.

Theorem 12. *There is a linear time algorithm that computes the linear rank-width of any forest, and an ordering of its vertex set V witnessing its linear rank-width can be computed in time $\mathcal{O}(|V| \cdot \log |V|)$.*

Example 13. *Let T be the graph shown in Figure 2. The ordering b, a, c, d, e is a witness for $\mathrm{lrw}(T) \leq 1$. The strategy for two cops according to the proof of Theorem 11 is as follows: the first cop moves to b and then the second cop moves to a and remains there. Now the first cop moves to c, d, e in this ordering.*

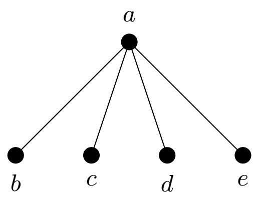


Fig. 2. The tree of Example 13

Example 14. *The tree T in Figure 3 satisfies $\mathrm{lrw}(T) = 2$. The given ordering (attached to the vertices) witnesses $\mathrm{lrw}(T) \leq 2$. The strategy for three cops according to Theorem 11 is* $\{1\}, \{1,2\}, \{2,3\}$, $\{2,4\}, \{4,5\}$, $\{4,5,6\}, \{4,6,7\}$, $\{4,8\}, \{8,9\}$, $\{8,9,10\}, \{8,10,11\}, \{8,10,12\}, \{8,12,14\}$, $\{8,13,14\}, \{8,14,15\}$, $\{8,16\}, \{8,16,17\}$, $\{8,17,18\}, \{8,17,19\}, \{8,19,20\}, \{19,20,21\}, \{21,22\}$.

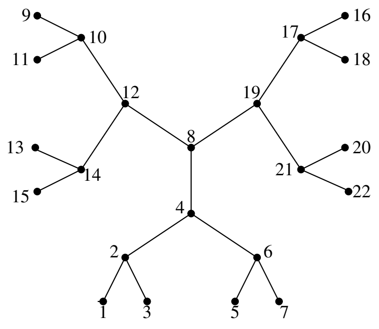


Fig. 3. The tree of Example 14

4 Linear Clique-Width

In this section we prove Theorem 3, characterizing the linear clique-width of forests in terms of their path-width. It follows that the linear clique-width of forests is linear time computable. In [11] it is proved that the linear clique-width of a graph is at most its path-width plus 2. We prove that for forests containing a path of length three, this upper bound is also a lower bound.

Let us recall the definition of linear clique-width [11,15,27]. Let k be a positive integer. A *k-labeled* graph is a pair (G, γ) where G is a graph and $\gamma : V(G) \to [k]$ is a mapping; we will also denote it by $(V(G), E(G), \gamma)$. The k-labeled graph consisting of a single vertex labeled by $i \in [k]$ is denoted by $(\mathbf{i}, \gamma_{\mathbf{i}})$. The set LIN-CW$_k$ of k-labeled graphs is defined inductively with the following operations.

1. For each $i \in [k]$, $(\mathbf{i}, \gamma_{\mathbf{i}})$ is in LIN-CW$_k$.
2. If $i, j \in [k]$ and (G, γ) is in LIN-CW$_k$, then $(\rho_{i\to j}(G), \gamma)$ is in LIN-CW$_k$ and denotes the k-labeled graph $(V(G), E(G), \gamma')$ with
$$\gamma'(x) := \begin{cases} \gamma(x) & \text{if } \gamma(x) \neq i, \\ j & \text{otherwise.} \end{cases}$$
3. If $i, j \in [k]$, $i \neq j$, and (G, γ) is in LIN-CW$_k$, then $(\eta_{i,j}(G), \gamma)$ is in LIN-CW$_k$ and denotes the k-labeled graph $(V(G), E', \gamma)$ with
$$E' := E(G) \cup \{\{x, y\} \mid \gamma(x) = i \text{ and } \gamma(y) = j\}\,.$$
4. If $i \in [k]$ and (G, γ) is in LIN-CW$_k$, then $(G \oplus \mathbf{i}, \gamma')$ is in LIN-CW$_k$ and denotes the graph $(V(G) \cup \{z\}, E(G), \gamma')$ where $z \notin V(G)$ and
$$\gamma'(x) := \begin{cases} \gamma(x) & \text{if } x \in V(G), \\ i & \text{otherwise.} \end{cases}$$

An expression built with the operations $\mathbf{i}, \rho_{i\to j}, \eta_{i,j}$ and $\oplus$ according to the definition of LIN-CW$_k$ is called a *linear k-expression.* The *linear clique-width* of a graph G, denoted by $\mathrm{lcw}(G)$, is the minimum k such that G is isomorphic to a graph in LIN-CW$_k$ (after forgetting the labels). It is worth noticing that if H is an induced subgraph of G, then $\mathrm{lcw}(H) \leq \mathrm{lcw}(G)$. Moreover, any linear k-expression t defining a graph G defines a linear ordering of $V(G)$ witnessing the ordering in which the vertices of G appears in t.

Lemma 15 ([8,11]). *Any graph G satisfies* $\mathrm{lcw}(G) \leq \mathrm{pw}(G) + 2$. □

The proofs of Lemmas 17 and 18 are omitted due to space constraints. For the proof of Lemma 18 we use the following lemma, proved in [10, Theorem 3.1].

Lemma 16. *Let T be a tree and let $k \geq 1$ be an integer. Then* $\mathrm{pw}(T) \leq k$ *if and only if for all $v \in V(T)$ at most two of the trees in $T \setminus v$ have path-width k and all others have path-width less than k.* □

Lemma 17. *Let T be a tree obtained from three trees T_1, T_2 and T_3 by adding a new vertex r adjacent to exactly one vertex in each of the three trees. If* $\mathrm{lcw}(T_i) = k$ *for each* $i \in \{1, 2, 3\}$*, then* $\mathrm{lcw}(T) \geq k + 1$.

Lemma 18. *Any forest T containing a path of length three satisfies* $\mathrm{lcw}(T) \geq \mathrm{pw}(T) + 2$.

Proof of Theorem 3. The first statement follows from Lemmas 15 and 18. For the second statement, if T does not contain a path of length three, then it is a star. Since stars with at least one edge have linear clique-width 2 and path-width 1, we can conclude that $\mathrm{lcw}(T) = \mathrm{pw}(T) + 1$. □

5 Minimal Excluded Acyclic Vertex-Minors

As an application, in this section we identify the minimal excluded *acyclic* vertex-minors for linear rank-width k. For this result we use both Lemma 16 and the fact that linear rank-width and path-width coincide on trees.

For a graph G and a vertex x of G, the *local complementation at x* of G consists in replacing the subgraph induced on the neighbors of x by its complement. The resulting graph is denoted by $G{*}x$. If H can be obtained from G by a sequence of local complementations, then G and H are called *locally equivalent*. A graph H is called a *vertex-minor* of a graph G if H is isomorphic to a graph obtained from G by applying a sequence of local complementations and deletions of vertices. The graph H is a *proper vertex-minor* of G if H is a vertex-minor of G and $|V(H)| < |V(G)|$. A graph G is a *minimal excluded vertex-minor* for the class of graphs of linear rank-width k, if $\mathrm{lrw}(G) > k$ and $\mathrm{lrw}(H) \leq k$ for all proper vertex-minors H of G. It is known that for fixed k, the set of minimal excluded vertex-minors for the class of graphs of linear rank-width at most k is finite [21]. For $k = 1$, the set of minimal excluded vertex-minors consists of three graphs [1]. For $k \geq 2$, a double-exponential lower bound on the number of minimal excluded vertex-minors is known [22] . See also [6,20] for more information on vertex-minors.

We say that a graph G is a *minimal excluded acyclic vertex-minor* for the class of graphs of linear rank-width k, if G is acyclic and every proper acyclic vertex-minor of G has linear rank-width less than k. Note that a minimal excluded acyclic vertex-minor may not be a minimal excluded vertex minor. For example, let R_3 be the the tree obtained from the star with three leaves by subdividing each edge once (cf. Figure 4). Then R_3 is a minimal excluded acyclic vertex-minor for the class of graphs of linear rank-width at most 1, but it contains the net graph (i. e. the graph obtained from a triangle by adding three pendant vertices, one to each of the vertices of the triangle) shown in Figure 4 as a proper vertex minor, which in turn is a minimal excluded vertex-minor for the class of graphs of linear rank-width at most one [1].

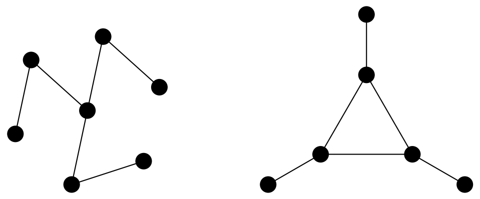


Fig. 4. The subdivided 3-star R_3, and the net graph

We now determine the set of pairwise not locally equivalent minimal excluded acyclic vertex-minors for linear rank-width k. Due to minimality, the minimal excluded (acyclic) vertex-minors for linear rank-width k are necessarily connected. Let $\mathcal{H}_1 := \{R_3\}$. For $k \geq 2$, let $\mathcal{H}_k$ be the set of (pairwise non isomorphic) trees obtained by taking a new vertex r and three trees in $\mathcal{H}_{k-1}$, and by linking this new vertex to one vertex in each of these three trees. Notice that two trees in $\mathcal{H}_k$ have the same size.

Lemma 19. *Let $k \geq 1$ be an integer. Every tree of linear rank-width $k+1$ contains a tree in $\mathcal{H}_k$ as a vertex-minor.*

Theorem 20. *For each $k \geq 1$, the set $\mathcal{H}_k$ is the set of minimal excluded acyclic vertex-minors for linear rank-width k.*

Proof. One can prove by induction, by using Theorem 2 and Lemma 16, that each tree in $\mathcal{H}_k$ has linear rank-width $k+1$ and is minimal with respect to this property. Moreover, by Lemma 19 any tree of linear rank-width $k+1$ contains as a vertex-minor a tree in $\mathcal{H}_k$. So it is enough to prove that two trees in $\mathcal{H}_k$ are not locally equivalent. Bouchet has proved in [6] that two trees are locally equivalent if and only if they are isomorphic. Hence, since no two trees in $\mathcal{H}_k$ are isomorphic, we are done. $\square$

6 Conclusion

We proved that linear rank-width and path-width coincide on forests, and we determined the linear clique-width of forests in terms of their path-width. Our proof method for the first result completely differs from our proof method for the second result. We believe that the second method can be adapted in order to obtain a shorter but non-constructive proof for the first result.

We obtained a linear time algorithm that computes the linear rank-width and the linear clique-width of any forest. Natural questions are: Is there a linear time algorithm that computes the linear rank-width (or linear clique-width) of distance-hereditary graphs or of series-parallel graphs? And, more generally, is there a polynomial time algorithm that computes the linear rank-width (or linear clique-width) of graphs of bounded rank-width?

We used the fact that linear rank-width and path-width coincide in forests to determine the set of minimal excluded acyclic vertex-minors for linear rank-width k. One can probably use the same technique to compute the set of *minimal excluded acyclic induced subgraphs* for linear rank-width and linear clique-width k. The complete set of minimal excluded vertex-minors for linear rank-width k is unknown and a next step could be to determine the set of distance-hereditary excluded vertex-minors for linear rank-width k (we know from [22] that the number is at least doubly exponential in k). In [20] it is proved that the size of the excluded vertex-minors for rank-width k is bounded by $(6^{k+1} - 1)/5$, and similar results exist for tree-width and path-width [25]. Can we get a similar result for linear rank-width?

Clique-width and linear clique-width are not monotone with respect to the vertex-minor inclusion and are only known to be monotone with respect to the induced subgraph inclusion. Characterizing linear clique-width with respect to the induced subgraph inclusion seems to be a hard task and few results have been obtained [13,19]. Can we at least characterize the linear clique-width of co-graphs (which have clique-width at most 2) or in general of distance-hereditary graphs (which have clique-width at most 3) in order to identify the set of distance-hereditary excluded induced subgraphs for linear clique-width k?

Acknowledgement. We thank an anonymous referee for helpful comments.

References

1. Adler, I., Farley, A.M., Proskurowski, A.: Obstructions for linear rankwidth at most 1. J. Discrete Applied Mathematics (to appear, 2013)
2. Bienstock, D., Seymour, P.D.: Monotonicity in graph searching. J. Algorithms 12(2), 239–245 (1991)
3. Bodlaender, H.L., Kloks, T.: Efficient and constructive algorithms for the pathwidth and treewidth of graphs. J. Algorithms 21(2), 358–402 (1996)
4. Bodlaender, H.L., Kloks, T., Kratsch, D.: Treewidth and pathwidth of permutation graphs. SIAM J. Discrete Math. 8(4), 606–616 (1995)
5. Bodlaender, H.L., Möhring, R.H.: The pathwidth and treewidth of cographs. In: Gilbert, J.R., Karlsson, R. (eds.) SWAT 1990. LNCS, vol. 447, pp. 301–309. Springer, Heidelberg (1990)
6. Bouchet, A.: Transforming trees by successive local complementations. J. Graph Theory 12(2), 195–207 (1988)
7. Bouchet, A.: Circle graph obstructions. J. Comb. Theory, Ser. B 60(1), 107–144 (1994)
8. Courcelle, B., Engelfriet, J.: Graph Structure and Monadic Second-Order Logic, A Language-Theoretic Approach. Cambridge University Press (2012)
9. Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. Discrete Applied Mathematics 101(1-3), 77–114 (2000)
10. Ellis, J.A., Sudborough, I.H., Turner, J.S.: The vertex separation and search number of a graph. Inf. Comput. 113(1), 50–79 (1994)
11. Fellows, M.R., Rosamond, F.A., Rotics, U., Szeider, S.: Clique-width is np-complete. SIAM J. Discrete Math. 23(2), 909–939 (2009)
12. Ganian, R.: Thread graphs, linear rank-width and their algorithmic applications. In: Iliopoulos, C.S., Smyth, W.F. (eds.) IWOCA 2010. LNCS, vol. 6460, pp. 38–42. Springer, Heidelberg (2011)

13. Gurski, F.: Characterizations for co-graphs defined by restricted nlc-width or clique-width operations. Discrete Mathematics 306(2), 271–277 (2006)
14. Gurski, F.: Linear layouts measuring neighbourhoods in graphs. Discrete Mathematics 306(15), 1637–1650 (2006)
15. Gurski, F., Wanke, E.: On the relationship between nlc-width and linear nlc-width. Theor. Comput. Sci. 347(1-2), 76–89 (2005)
16. Gurski, F., Wanke, E.: The nlc-width and clique-width for powers of graphs of bounded tree-width. Discrete Applied Mathematics 157(4), 583–595 (2009)
17. Heggernes, P., Meister, D., Papadopoulos, C.: A complete characterisation of the linear clique-width of path powers. In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 241–250. Springer, Heidelberg (2009)
18. Heggernes, P., Meister, D., Papadopoulos, C.: Graphs of linear clique-width at most 3. Theor. Comput. Sci. 412(39), 5466–5486 (2011)
19. Heggernes, P., Meister, D., Papadopoulos, C.: Characterising the linear clique-width of a class of graphs by forbidden induced subgraphs. Discrete Applied Mathematics 160(6), 888–901 (2012)
20. Oum, S.I.: Rank-width and vertex-minors. J. Comb. Theory, Ser. B 95(1), 79–100 (2005)
21. Oum, S.: Rank-width and well-quasi-ordering. SIAM J. Discrete Math. 22(2), 666–682 (2008)
22. Jeong, J., Kwon, O.-J., Oum, S.I.: Excluded vertex-minors for graphs of linear rank-width at most k. In: Portier, N., Wilke, T. (eds.) STACS. LIPIcs, vol. 20, pp. 221–232. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2013)
23. Kirousis, L.M., Papadimitriou, C.H.: Searching and pebbling. Theor. Comput. Sci. 47(3), 205–218 (1986)
24. Kloks, T., Bodlaender, H.L.: Approximating treewidth and pathwidth of some classes of perfect graphs. In: Ibaraki, T., Inagaki, Y., Iwama, K., Nishizeki, T., Yamashita, M. (eds.) ISAAC 1992. LNCS, vol. 650, pp. 116–125. Springer, Heidelberg (1992)
25. Lagergren, J.: Upper bounds on the size of obstructions and intertwines. J. Comb. Theory, Ser. B 73(1), 7–40 (1998)
26. LaPaugh, A.S.: Recontamination does not help to search a graph. J. ACM 40(2), 224–245 (1993)
27. Lozin, V.V., Rautenbach, D.: The relative clique-width of a graph. J. Comb. Theory, Ser. B 97(5), 846–858 (2007)
28. Megiddo, N., Hakimi, S.L., Garey, M.R., Johnson, D.S., Papadimitriou, C.H.: The complexity of searching a graph. J. ACM 35(1), 18–44 (1988)
29. Oum, S.-I., Seymour, P.D.: Approximating clique-width and branch-width. J. Comb. Theory, Ser. B 96(4), 514–528 (2006)
30. Scheffler, P.: Die Baumweite von Graphen als ein Maß für die Kompliziertheit algorithmischer Probleme. Akademie der Wissenschaften der DDR, Berlin, PhD thesis (1989)
31. Suchan, K., Todinca, I.: Pathwidth of circular-arc graphs. In: Brandstädt, A., Kratsch, D., Müller, H. (eds.) WG 2007. LNCS, vol. 4769, pp. 258–269. Springer, Heidelberg (2007)
32. Takahashi, A., Ueno, S., Kajitani, Y.: Minimal acyclic forbidden minors for the family of graphs with bounded path-width. Discrete Mathematics 127(1-3), 293–304 (1994)

Threshold-Coloring and Unit-Cube Contact Representation of Graphs

Md. Jawaherul Alam[1], Steven Chaplick[2], Gašper Fijavž[3], Michael Kaufmann[4], Stephen G. Kobourov[1,⋆], and Sergey Pupyrev[1,5]

[1] Department of Computer Science, University of Arizona, Tucson, AZ, USA
[2] Department of Applied Mathematics, Charles University, Prague, Czech Republic
[3] Faculty of Computer and Information Science, University of Ljubljana, Slovenia
[4] Wilhelm-Schickhard-Institut für Informatik, Universität Tübingen, Tübingen, Germany
[5] Institute of Mathematics and Computer Science, Ural Federal University, Russia

Abstract. We study *threshold coloring* of graphs where the vertex colors, represented by integers, describe any spanning subgraph of the given graph as follows. Pairs of vertices with near colors imply the edge between them is present and pairs of vertices with far colors imply the edge is absent. Not all planar graphs are threshold-colorable, but several subclasses, such as trees, some planar grids, and planar graphs with no short cycles can always be threshold-colored. Using these results we obtain unit-cube contact representation of several subclasses of planar graphs. We show the NP-completeness for two variants of the threshold coloring problem and describe a polynomial-time algorithm for another.

1 Introduction

Graph coloring is among the fundamental problems in graph theory. Typical applications of the problem and its generalizations are in job scheduling, channel assignments in wireless networks, register allocation in compiler optimization and many others [12]. In this paper we consider a new graph coloring problem in which we assign colors (integers) to the vertices of a graph G in order to define a spanning subgraph H of G. In particular, we color the vertices of G so that for each edge of H, the two endpoints are near, that is, their distance is within a given "threshold", and for each edge of $G \setminus H$, the endpoints are far, that is, their distance greater than the threshold; see Fig 1.

The motivation of the problem is twofold. First, such coloring can be used for the Frequency Assignment Problem [8], which asks for assigning frequencies to transmitters in radio networks so that only specified pairs of transmitters can communicate with each other. Second, such coloring can be used in the context of the geometric problem of *unit-cube contact representation* of planar graphs [3]. Suppose a planar graph G has a unit-cube contact representation where one face of each cube is co-planar; see Fig. 1(a). Assume that we can define a spanning subgraph H of G by our particular vertex coloring. We show that it is possible to compute a unit-cube contact representation of H by lifting the cube for each vertex v by the amount equal to the color of v (where the size or side-length of the cubes are roughly equal to the threshold); see Fig. 1(b).

⋆ Research supported in part by NSF grants CCF-1115971 and DEB 1053573.

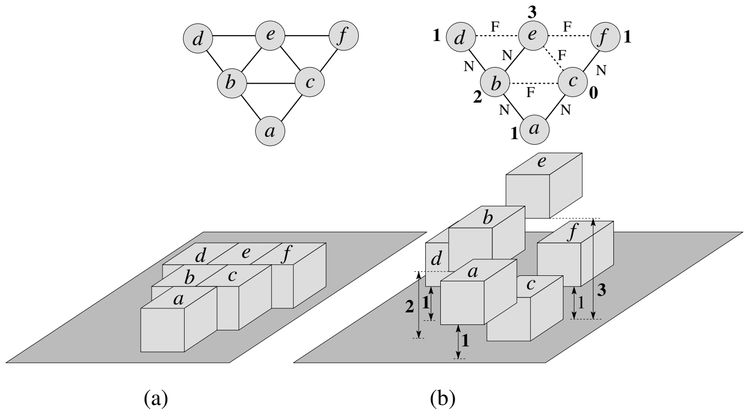


Fig. 1. (a) A planar graph G and its unit-cube contact representation where the bottom faces of all cubes are co-planar, (b) a spanning subgraph H of G with a $(4, 1)$-threshold-coloring and its unit-cube contact representation. Far edges are shown dashed, near edges are shown solid.

Problem Definition: An *edge-labeling* of graph $G = (V, E)$ is a mapping $\ell : E \rightarrow \{N, F\}$ assigning labels N or F to each edge and the pair $\{N, F\}$ defines a partition of the edges into *near* and *far* edges. Let $r \geq 1$ and $t \geq 0$ be two integers and let $[1 \ldots r]$ denote a set of r consecutive integers. For a graph $G = (V, E)$ and an edge-labeling $\ell : E \rightarrow \{N, F\}$, a (r, t)*-threshold-coloring* of G with respect to ℓ is a *coloring* $c : V \rightarrow [1 \ldots r]$ such that for each edge $e = (u, v) \in E$, $e \in N$ if and only if $|c(u) - c(v)| \leq t$. We call r and t the *range* and the *threshold*. Note that the set of near edges defines a spanning subgraph $H = (V, N)$ of G, where H is a *spanning subgraph* of graph G if it contains all vertices of G. H is a *threshold subgraph* of G if there exists such a threshold-coloring.

A graph G is *total-threshold-colorable* if for every edge-labeling ℓ of G there exists an (r, t)-threshold-coloring of G with respect to ℓ for some $r \geq 1, t \geq 0$ (for every partition of edges of G into near and far edges, we can produce vertex colors so that endpoints of near edges receive near colors, and endpoints of far edges receive colors that are far apart). A graph G is (r, t)*-total-threshold-colorable* if it is total-threshold-colorable for the range r and threshold t. We consider the following problem variants.

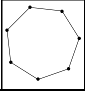
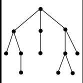
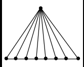
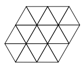
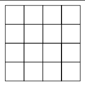
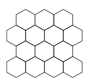
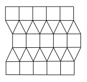
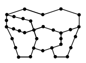
Total-Threshold-Coloring: Given a graph G, is G total-threshold-colorable, that is, is every spanning subgraph of G a threshold subgraph of G?

Threshold-Coloring: Given a graph G and a spanning subgraph H, is H a threshold subgraph of G for some integers $r \geq 1, t \geq 0$?

Exact-Threshold-Coloring: Define H to be an *exact-threshold* graph if H is a threshold subgraph of the complete graph G for some $r \geq 1, t \geq 0$. Given a graph H, is H an exact-threshold graph?

Fixed-Threshold-Coloring: Given a graph G, a spanning subgraph H, and integers $r \geq 1, t \geq 0$, is H (r, t)-threshold-colorable?

Table 1. Results on the **Total-Threshold-Coloring** problem. "No" entries in the last row follow from the fact that graphs with vertices of high degrees cannot have unit-cube representation [3].

graph classes	Cycle	Tree	Fan	Triangular Grid	Square Grid	Hexagonal Grid	Square-Triangle Grid	Planar Graph w/o Cycles of size ≤ 9
								
threshold coloring	$r = 5$, $t = 1$	$r = 2$, $t = 0$	$r = 5$, $t = 1$	No	Open	$r = 5$, $t = 1$	No	$r = 8$, $t = 2$
unit-cube contact	Yes	No	No	Open	Yes	Yes	Open	No

Related Work: Many graph theoretic problems deal with vertex coloring a graph and numerous graph classes are defined based on such coloring; see [2] for a survey. To the best of our knowledge, total-threshold-colorability defines a new class of graphs. Here we mention two related classes: threshold and difference graphs. *Threshold graphs* are ones for which there is a real number S and for every vertex v there is a real weight a_v such that (v, w) is an edge if and only if $a_v + a_w \geq S$ [10]. A graph is a *difference graph* if there is a real number S and for every vertex v there is a real weight a_v such that $|a_v| < S$ and (v, w) is an edge if and only if $|a_v - a_w| \geq S$ [9]. Note that for both classes the threshold (real number S) defines edges between all pairs of vertices, while in our setting the threshold only defines edges of a (not necessarily complete) graph G.

A threshold-coloring of a planar graph can be used to find a contact representation of the graph with cuboids (axis aligned boxes) in 3D. Thomassen [13] shows that any planar graph has a proper contact representation by cuboids in 3D. In a *contact representation* of a graph, the vertices are represented by cuboids (or other polygonal shapes) and the edges are realized by a common boundary of the two corresponding cuboids. A contact representation is *proper* if for each edge the corresponding common boundary has non-zero area. Felsner and Francis [5] prove that any planar graph has a (non-proper) contact representation by cubes. Bremner *et al.* [3] proves that the same result does not hold when using only unit cubes.

Our Contributions: First we show some connections between threshold-coloring and other graph problems. Specifically, we show that the threshold coloring and the fixed threshold coloring problems are NP-complete by reductions from the proper interval graph sandwich problem and standard vertex coloring, and the exact threshold coloring problem can be solved in linear time via equivalence to proper interval graph recognition. We then study the total threshold coloring problem for various planar graph classes. Specifically, we show that trees, hexagonal grids, planar graphs without any cycles of length ≤ 9 are total-threshold-colorable, while the triangular grid and the square-triangle grid are not; these results are summarized in Table 1. Finally we show how to use the threshold-coloring to compute unit-cube contact representations for several subclasses of planar graphs; see the last column of Table 1.

2 Threshold-Coloring and Other Graph Problems

We begin with connections to classical graph coloring problems.

Vertex Coloring Problem: Let $G = (V, E)$ be a graph. We call G k-vertex-colorable if there exists a coloring $c : V \to [1 \dots k]$ such that for any edge $(u, v) \in E$, $c(u) \neq c(v)$, that is, u and v have different colors. Given an input graph G and an integer $k > 0$, the vertex coloring problem asks whether there exists a k-vertex-coloring of G. Lemma 1 immediately follows from the definition.

Lemma 1. *Let $G = (V, E)$ be a graph and let k be a positive integer. Define an edge-labeling $\ell : E \to \{N, F\}$ that assigns each edge the label F, that is, for each edge $e \in E$, $\ell(e) = F$. Then G has a k-vertex-coloring if and only if there exists a $(k, 0)$-threshold-coloring of G with respect to ℓ.*

Proper Interval Representation Problem: An *interval representation* [2] for a graph $G = (V, E)$ is one where each vertex $v \in V$ is represented by an interval $I(v)$ of $\mathbb{R}$ such that for any edge $(u, v) \in E$, the intervals $I(u)$ and $I(v)$ have a non-empty intersection. A *proper interval graph* [2] is one that has an interval representation such that no interval properly contains another. Equivalently, a *proper interval graph* is one that has an interval representation with unit intervals [11]. The problem of proper interval representation for a graph G asks whether G has a proper interval representation.

Lemma 2. *H is an exact-threshold graph if and only if it is a proper interval graph.*

Proof. (sketch) If a graph $H = (V, E)$ is an exact-threshold graph, there are integers $r \geq 1, t \geq 0$ and a mapping $c : V \to [1 \dots r]$ such that for any pair $u, v \in V$, $(u, v) \in E \Leftrightarrow |c(u) - c(v)| \leq t \Leftrightarrow |c(u) - c(v)| < t + \epsilon$ with $0 < \epsilon < 1$ since $c(u)$ and $c(v)$ are integers. Then there is an interval representation of G that contains for each vertex $v \in V$, a unit interval in the range $[c(v)/(t + \epsilon), c(v)/(t + \epsilon) + 1]$. Conversely, if H has a unit interval representation Γ, then scale Γ by an integer factor t, so that each endpoint of each interval has integer coordinate. Then for each vertex v, the coordinate of the left endpoint of the interval for v defines a color $c(v)$ such that for any pair $u, v \in V$, $|c(u) - c(v)| \leq t \Leftrightarrow (u, v) \in E$. □

Graph Sandwich Problem: Given two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ on the same vertex set V, where $E_2 \subseteq E_1$, and a property Π, does there exist a graph $H = (V, E)$ on the same vertex set such that $E_2 \subseteq E \subseteq E_1$ and H satisfies property Π? Here E_1 and E_2 can be thought of as *universal* and *mandatory* sets of edges, with E sandwiched between the two sets. We are interested in a particular property for the graph sandwich problem: "proper interval representability". A graph satisfies *proper interval representability* if it admits a proper interval representation [7].

Lemma 3. *Let $G = (V, E_G)$ and $H = (V, E_H)$ be two graphs on the same vertex set V such that $E_H \subseteq E_G$. Then the threshold-coloring problem for G with respect to the edge partition $\{E_H, E_G - E_H\}$ is equivalent to the graph sandwich problem for the vertex set V, mandatory edge set E_H, universal edge set $E_H \cup (V \times V - E_G)$ and proper interval representability property.*

Proof. (sketch) Define $E_U = E_H \cup (V \times V - E_G)$. Suppose there exists a graph $H^* = (V, E^*)$ such that $E_H \subseteq E^* \subseteq E_U$ and H^* has a proper interval representation. Then by Lemma 2, there exist integers $r \geq 1$ and $t \geq 0$ and a coloring $c : V \rightarrow [1 \ldots r]$ such that for any pair $u, v \in V$, $|c(u) - c(v)| \leq t$ if and only if $(u, v) \in E^*$. Then for any edge $(u, v) \in E_G$, $(u, v) \in E_H \Leftrightarrow |c(u) - c(v)| \leq t$. Conversely, if there exists integers $r \geq 1$ and $t \geq 0$ such that there is an (r, t)-threshold-coloring c' of G with respect to the edge partition $\{E_H, E_G - E_H\}$, then define an edge set $E^* = \{(u, v) \in V \times V : |c'(u) - c'(v)| \leq t\}$. Clearly the graph $H^* = (V, E^*)$ has an exact (r, t)-threshold-coloring and hence by Lemma 2, H^* has a proper interval representation. Furthermore E^* is sandwiched between E_H and E_U. □

The following theorem follows from Lemmas 1, 2 and 3 since the vertex coloring and the graph sandwich problems for proper interval representability are NP-complete [6] and the proper interval recognition can be solved in linear time [4].

Theorem 1. *The* ***Threshold-Coloring*** *and* ***Fixed-Threshold-Coloring*** *problems are NP-complete, while the* ***Exact-Threshold-Coloring*** *problem can be solved in linear time.*

3 Total-Threshold-Coloring of Graphs

In this section we address the **Total-Threshold-Coloring** problem: can every spanning subgraph of a graph G be represented by appropriately coloring the vertices of G?

First note that not every graph (not even every planar graph) is total-threshold-colorable. Suppose that $G = K_4$, and we would like to represent a subgraph where four of the edges remain and span a 4-cycle, while the other two edges are removed (edge-partitioning $\{N, F\}$). Assume that there exists an (r, t)-threshold-coloring with colors c_1, c_2, c_3, c_4 for vertices v_1, v_2, v_3, v_4 respectively. Without loss of generality assume c_4 is the highest color and $(v_1, v_4) \in F$, hence also $(v_2, v_3) \in F$. Also assume $c_3 \geq c_2$ and consequently $c_4 - c_2 \geq c_3 - c_2$. The left side of the inequality should be at most t, and the right side strictly greater than t, which cannot be accomplished by any choice of the range and the threshold.

Paths, Cycles, Trees, Fans: For *paths* and *trees* there is a trivial coloring with threshold $t = 0$ and two colors. Choose an arbitrary vertex as the root and color it 0. Color 1 all vertices with an odd number of far edges on the shortest path to the root. Color 0 all vertices with an even number of far edges to the root. Then all vertices connected by a

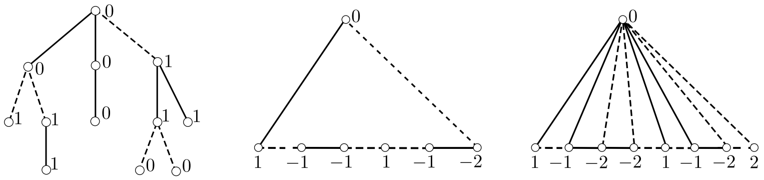
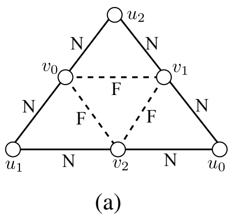


Fig. 2. Threshold-coloring of trees, cycles, and fans



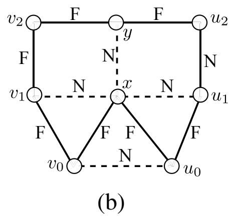


Fig. 3. Graphs which are not threshold-colorable

near edge of G get the same color, and vertices connected by a far edge get different colors; see Fig. 2(a).

For *cycles* and *fans* there is a coloring scheme with threshold $t = 1$ and five colors. A *fan* is obtained from a path P by adding a new vertex v connected to all vertices of the path. We use colors $\{-2, -1, 0, 1, 2\}$ to color a fan. The vertices of P are colored by -1 and 1, and v is colored by 0. After this initial coloring some of the far edges $(u, v), u \in P$ might have $|c(u) - c(v)| = 1$. We fix it by changing the color of u from 1 to 2 or from -1 to -2; see Fig. 2(c). It is easy to see that the same algorithm can be applied to color a cycle; see Fig. 2(b).

Triangular Grid: In a triangular grid all faces are triangles and internal vertices have degree 6. It is easy to show that a triangular grid is not total-threshold-colorable. Consider the graph with vertices $v_0, v_1, v_2, u_0, u_1, u_2$, where each vertex u_i is adjacent to v_{i+1} and v_{i+2} (mod 3); see Fig. 3(a). Let $F = \{(v_0, v_1), (v_1, v_2), (v_2, v_0)\}$, and let N contain the remaining 6 edges. Assume that there exists a (r, t)-threshold-coloring c. Without loss of generality, let $c(v_0) < c(v_1) < c(v_2)$. Now on one hand $c(v_2) - c(v_0) > 2t$ and on the other $c(v_2) - c(v_0) \le |c(v_2) - c(u_1)| + |c(u_1) - c(v_0)| \le 2t$, which is impossible. This also proves that outerplanar graphs are not total-threshold-colorable.

Hexagonal Grid: In a hexagonal grid all faces are 6-sided and internal vertices have degree 3. We show that the grid is total-threshold-colorable with $r = 5$ and $t = 1$.

Lemma 4. *Let $P_2 = \{v_0, v_1, v_2\}$ be a path of length 2. Then for any edge-labeling of P_2 and a fixed color $k \in \{-2, -1, 1, 2\}$, there is a threshold-coloring c of P_2 with threshold $t = 1$, where $c(v_0) = 0$, $c(v_2) = k$ and $c(v_1) \in \{-2, -1, 1, 2\}$.*

Proof. Depending on whether the edge (v_0, v_1) is near or far, choose $c(v_1)$ to be 1 or 2. If the label of (v_1, v_2) disagrees with the colors of v_1 and v_2 then change the sign of $c(v_1)$. □

Lemma 5. *Any hexagonal grid is $(5, 1)$-total-threshold-colorable.*

Proof. The coloring is done in two steps. In the first step we assign color 0 for a set of independent vertices of G as shown in Fig. 4(a), where the colored vertices are white. Note that no two white vertices have a shortest path of length less than 3.

In the second step we find a coloring of the remaining black vertices, using only four colors $\{-2, -1, 1, 2\}$. Let w_1 be a white vertex. We arbitrarily choose one of its

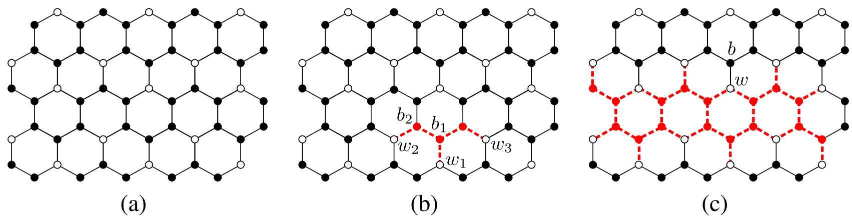


Fig. 4. Total-threshold-coloring of the hexagonal grid. (a) White vertices get color 0, black vertices get one of the colors $-2, -1, 1, 2$. (b) A color assignment to b_1 can be extended to vertices w_2 and w_3 based on the labels of the red dashed edges. (c) Assigning colors to the red vertices.

black neighbors b_1, and assign a color for b_1 based on the label of edge (w_1, b_1). Now vertex b_1 has two white vertices w_2 and w_3 within distance 2. Using Lemma 4 we can (uniquely) extend the coloring of b_1 to w_2 (symmetrically, to w_3) so that additional black vertex b_2 gets a color. Again, the coloring of b_2 can be extended to its nearest white neighbor. We continue such a propagation of colors, see Figs. 4(b) and 4(c) where processed black vertices and edges are shown dashed red. One can see that the process will color a row of hexagons with alternate upper and lower legs. To complete the coloring of G we choose a white vertex in the next row of hexagons and initiate a similar propagation process. For example, one can use vertices w and b shown in Fig. 4(c). □

Square-Triangle Grid: We prove that the graph in Fig. 3(b) is not total-threshold-colorable. Assume to the contrary that c is a (r, t)-threshold-coloring. Without loss of generality let $c(v_0) < c(u_0)$. Since (v_1, u_0) is a far edge and $(v_0, x), (u_0, x)$ are near we have $c(v_0) < c(x) < c(u_0)$. Similar argument shows that $c(v_1) < c(v_0) < c(x) < c(u_0) < c(u_1)$. Then if $x < y$, we have $c(v_1) + t < c(x)$ and $c(x) + t < c(y)$, which implies $c(v_1) + 2t < c(y)$. This makes it impossible to find a color for v_2 near to both v_1 and y. Similarly if $x > y$ then it is impossible to color u_2. Theorem 2 summarizes the results in this section.

Theorem 2. *Paths, cycles, trees, fans, and the hexagonal grid are total-threshold-colorable. The triangular grid and the triangle-square grid are not total-threshold-colorable.*

4 Planar Graphs without Short Cycles

In the counter-examples of total-threshold-colorability (e.g., K_4 and the triangular grid) we have short cycles, which can be used to force groups of vertices to be simultaneously near and far. In this section we show that planar graphs without short cycles are always total-threshold-colorable.

Theorem 3. *Let G be a planar graph without cycles of length ≤ 9. Then G is $(8, 2)$-total-threshold-colorable*[1].

[1] Equivalently, the *girth* (that is, the shortest cycle) of G should be ≥ 10.

The outline of our proof for Theorem 3 is as follows. We first find some small tree structures T that are "reducible", in the sense that for any edge-labeling of T and any given fixed coloring of the leaves of T to the colors $\{0, 1, \ldots, 7\}$, there is a $(8, 2)$-threshold-coloring of T. For a contradiction assume that there is a planar graph with girth ≥ 10 having no $(8, 2)$-threshold-coloring. We consider the minimal such graph G, and by a discharging argument prove that G contains at least one of these reducible tree structures. This contradicts the minimality of G. We start with some technical claims.

Extending a Coloring: Let P_n be a path with vertices $v_0, \ldots, v_n$. Given an edge-labeling of P_n and the color c_0 of v_0 we call a color c_n *legal* if there exists a $(8, 2)$-threshold-coloring c of P_n, so that $c(v_0) = c_0$ and $c(v_n) = c_n$. Due to the space constraints, the proofs of the following 3 claims are in [1].

Claim 1. *Let P_1 be a path of length 1. Then at least one of the colors 1 or 6 is legal (irrespective of the edge label and the color c_0).*

Claim 2. *Let P_2 be a path of length 2. Then 3 is legal unless $c_0 = 3$ and $\{N, F\} = \{\{e_1\}, \{e_2\}\}$, that is, the edges e_1 and e_2 are labeled differently. Symmetrically, 4 is legal unless $c_0 = 4$ and $\{N, F\} = \{\{e_1\}, \{e_2\}\}$.*

Claim 3. *Let P_3 be a path of length 3. Then $1, 3, 4,$ and 6 are all legal (irrespective of the edge label and the color c_0).*

A *star* is a subdivision of the graph $K_{1,n}$, and its *center* is the single vertex of degree ≥ 3. Let T be a star. A *prong* of T is a path from a leaf to the center of T, and a prong with k edges is called a k-prong, we say that it has length k.

Claim 4. *Let T be a subdivision of $K_{1,3}$ with prongs of length $1, 2,$ and 3, respectively. Assume that the leaves of T are assigned colors, so that the leaf u on the 1-prong is colored with either 1 or 6. Then we can extend this partial coloring to the whole T.*

Proof. Let v be the center of T. Given $c(u)$, we can choose $c(v) \in \{3, 4\}$ so that the labeling condition on the 1-prong is satisfied. If this choice cannot be extended to the longer prongs, then the leaf of the 2-prong is also colored with either 3 or 4, see Claim 2. But then the choice $c(v) \in \{1, 6\}$ which satisfies the labeling condition on the 1-prong can be extended to the remaining prongs. □

Reducible Configurations: A *configuration* is a tree T, and is *reducible* if every assignment of colors to the leaves of T can be, for every possible edge-labeling of T, extended to a $(8, 2)$-threshold-coloring c of the whole T.

Claim 5. *A path P_4 of length 4 is a reducible configuration.*

Proof. Let v be a neighbor of a leaf in P_4. By Claim 1 and Claim 3 either $c(v) = 1$ or $c(v) = 6$ extends to the remaining uncolored vertices. □

Claim 5 implies that longer paths are reducible. Let us turn our attention to stars.

Claim 6. *(A) Let T be a star with at most 1 prong of length 1 and the remaining prongs have length 3. Then T is reducible.*

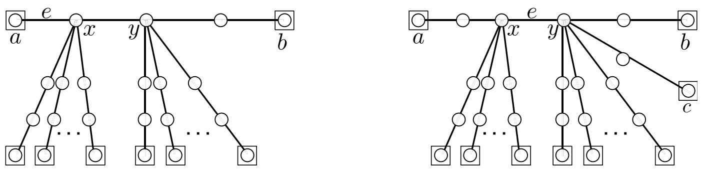


Fig. 5. Two additional types of reducible configurations, T1 and T2

(B) Let T be a star with at most 3 prongs of length 2 and the remaining prongs have length 3. Then T is reducible.

Proof. In both cases let v denote the center of the star. In order to establish (A) let $c(v)$ be either 1 or 6, which is appropriate for the 1-prong (such a choice exists by Claim 1). By Claim 3 the coloring $c(v)$ can be extended to the remaining 3-prongs. For (B) we may assume that neither 3 nor 4 can be extended to all three 2-prongs. By Claim 2 both colors 3 and 4 are used at leaves of the 2-prongs. Now, by Claim 1 at least one of $c(v) = 1$ or $c(v) = 6$ extends to the third 2-prong, and hence also to the remaining 2- and 3-prongs, by Claim 2 and Claim 3. □

Claim 7. *There exist two types T1 and T2 of reducible configurations shown in Fig. 4.*

The proof of Claim 7 is analogous to the proof of Claim 6, see [1].

Discharging: A *minimal counterexample* is the smallest possible (in terms of order) planar graph G without cycles of length ≤ 9 which is not $(8, 2)$-total-threshold-colorable. A minimal counterexample G does not contain reducible configurations. Further G is connected and has no vertices of degree 1. As G is also not a cycle (such a cycle should be of length ≥ 9 and should not contain a P_4), and is therefore homeomorphic to a (multi)graph of minimal degree ≥ 3.

Let us fix its planar embedding determining its set of faces $F(G)$. Let us define *initial charges*: initial charge of a vertex v, $\gamma_0(v)$, is equal to $4\deg(v) - 10$, and the initial charge of a face f, $\gamma_0(f)$, is equal to $\deg(f) - 10$. As all faces have length ≥ 10, every face is initially non-negatively charged, and we shall not alter the charges of faces. A routine application of Euler formula shows that the total initial charge is -20.

The discharging procedure will run in two phases, by $\gamma_i(v)$ we shall denote the charge of vertex v after Phase i of discharging. Informally, Phase 1 shall see that vertices of degree 2 do not have negative charges, and Phase 2 will leave only vertices of degree 3 with a possible negative charge.

Let u, v be vertices of G. We say that u and v are 2-adjacent, if G contains a $u - v$-path whose (possible) internal vertices all have degree 2. In Phase 1 we redistribute charge according to the following rule:

Rule 1: every vertex v of degree ≥ 3 sends charge 1 to every vertex u of degree 2, for which v and u are 2-*adjacent.*

In Phase 2 we shall apply the following rule:

Rule 2: If u and v are adjacent with $\gamma_1(u) > 0, \gamma_1(v) < 0$ then u sends charge 1 to v.

As every vertex u of degree 2 (we also call them *2-vertices*) is 2-adjacent to exactly two vertices of bigger degree, we have $\gamma_1(u) = 0$ in this case. For a vertex v of degree ≥ 3, the discharging in Phase 1 decreases the charge of v by the number of 2-vertices which are 2-adjacent to v.

Let v be a vertex of degree ≥ 3. A *prong at* v is a $v - x$-path whose other end-vertex x is of degree ≥ 3 and has internal vertices of degree 2.

Claim 8. *Let* v *be a vertex of degree* ≥ 3*. Then the number of 2-vertices that are 2-adjacent to* v *is at most* $2 \cdot \deg(v) - 3$.

Proof. By Claim 5 each prong at v contains at most two vertices of degree 2. If the shortest prong at v has length 1, then Claim 6 implies that at least one other prong has length ≤ 2. If the shortest prong at v has length 2, then by Claim 6 we have at least four prongs that are of length ≤ 2, and the result follows. □

Now Claim 8 serves as the lower bound for vertex charges after Phase 1, and in turn prepares us for the Phase 2 of discharging.

Claim 9. *(A) Let* v *be a vertex of degree* 3*. If* $\gamma_1(v) < 0$*, then* $\gamma_1(v) = -1$ *and the prongs at* v *have lengths* $1, 2$ *and* 3*, respectively.*

(B) Let v *be a vertex of degree* 3*. If* $\gamma_1(v) = 0$*, then the prongs at* v *have either lengths* $1, 1, 3$ *or* $1, 2, 2$*.*

(C) Let v *be a vertex of degree* 3 *with its prongs of length* $1, 1,$ *and* 2*. Then* $\gamma_1(v) = 1$*.*

(D) Let v *be a vertex of degree* 3 *with all* 3 *prongs of length* 1*. Then* $\gamma_1(v) = 2$*.*

(E) If v *is a vertex of degree* ≥ 4*, then* $\gamma_2(v) \geq 0$*, and also* $\gamma_2(v)$ *is not smaller than the number of* 1*-prongs at* v*.*

Proof. Let us first prove (E). Choose a vertex v with $\deg(v) \geq 4$. For every prong of length 3, v sends 2 units of charge in Phase 1. For every shorter prong v sends at most 1 unit of charge in either Phase 1 or Phase 2. The total charge sent out of v in both of the phases is by Claim 6 and Claim 8 at most $2\deg(v) - 2$. Hence $\gamma_2(v) \geq (4\deg(v) - 10) - (2\deg(v) - 2) = 2\deg(v) - 8 \geq 0$. The other cases merely stratify vertices of degree 3 according to the number of their 2-neighbors of degree 2. □

Claim 9(E) states that every vertex v of degree ≥ 4 satisfies $\gamma_2(v) \geq 0$. Similarly, if a 3-vertex u is adjacent to a vertex v whose degree is at least 4, then also $\gamma_2(u) \geq 0$. This fact follows from either Claim 9(A) and (E) (in case $\gamma_1(u) < 0$), or from either Claim 9(C) or (D) (if $\gamma_1(v) > 0$) as in this case u cannot send excessive charge in Phase 2.

Claim 10. *No vertex* v *has* $\gamma_2(v) < 0$ *and* $\gamma_1(v) < 0$*.*

Proof. Let v be a vertex satisfying both $\gamma_2(v) < 0$ and $\gamma_1(v) < 0$. By Claim 9 $\deg(v) = 3$ and v has prongs of length $1, 2, 3$. Let u be the only neighbor of v of degree $\neq 2$. Since v has received no charge from u in Phase 2 we have both $\deg(u) = 3$ and $\gamma_1(u) \leq 0$. By Claim 9 the prongs of u are of lengths $1, 2, 3$ or $1, 1, 3$ or $1, 2, 2$. Hence G contains one of configurations shown in Fig. 6(a). Now observe that these are reducible, as each matches one of T1 or T2 types of reducible configurations Claim 7. □

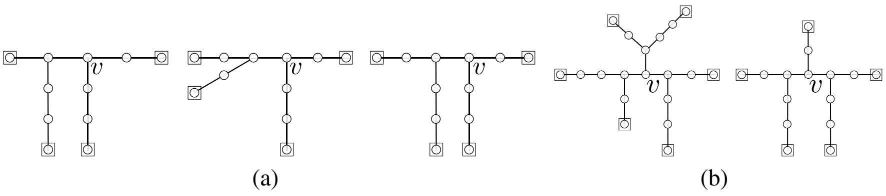


Fig. 6. (a) Negatively charged vertex v after both phases induces a reducible configuration. (b) Negatively charged vertex v after Phase 2 with positive charge after Phase 1.

Claim 11. *No vertex v has $\gamma_2(v) < 0$ and $\gamma_1(v) \geq 0$.*

Proof. If $\gamma_1(v) = 0$, then also $\gamma_2(v) = 0$, as Rule 2 does not reduce charge of a discharged vertex. By Claim 9(E) vertices of degree ≥ 4 do not have negative charge after Phase 2. Hence we may assume that v has degree 3, $\gamma_1(v) > 0$, and $\gamma_2(v) < 0$. By Claim 9(C) and (D) every neighbor u of v satisfies either $\deg(u) = 2$ or $\deg(u) = 3$ and $\gamma_1(u) < 0$. There are exactly two possible cases and they are shown in Fig. 6(b). It is enough to see that there exists a color choice $c(v)$ which can be extended in the 2-prong and/or stars centered at neighbors of v. Consider the option shown in the right. By Claim 1 at least one of $c(v) = 1$ or $c(v) = 6$ extends to the top 2-prong, and this choice also extends to the two copies of T, see Claim 4. In the left case both choices $c(v) = 1$ and $c(v) = 6$ extend to the three copies of T, again by Claim 4. □

Claim 10 and Claim 11 imply that no vertex has negative charge after Phase 2 of the discharging procedure. As the total charge remains negative and the faces cannot have negative charges, we have a contradiction, which completes the proof of Theorem 3.

5 Unit-Cube Contact Representations of Graphs

Lemma 6. *If G has a unit-cube contact representation Γ so that one face of each cube is co-planar in Γ, then any threshold subgraph of G also has a unit-cube representation.*

Proof. Let $H = (V, E_H)$ be a threshold subgraph of $G = (V, E_G)$ and let $c : V \to [1 \ldots r]$ be an (r, t)-threshold-coloring of G with respect to the edge-partition $\{E_H, E_G - E_H\}$. We now compute a unit-cube contact representation of H from Γ using c.

Assume (after possible rotation and translation) that the bottom face for each cube in Γ is co-planar with the plane $z = 0$; see Fig. 1(a). Also assume (after possible scaling) that each cube in Γ has side length $t + \epsilon$, where $0 < \epsilon < 1$. Then we can obtain a unit-cube contact representation of H from Γ by lifting the cube for each vertex v by an amount $c(v)$ so that its bottom face is at $z = c(v)$; see Fig. 1(b). Note that for any edge $(u, v) \in E_H$, the relative distance between the bottom faces of the cubes for u and v is $|c(u) - c(v)| \leq t < (t + \epsilon)$; thus the two cubes maintain contact. On the other hand, for each pair of vertices u, v with $(u, v) \notin E_H$, one of the following two cases occurs: (i) either $(u, v) \notin E_G$ and their corresponding cubes remain non-adjacent as they were in Γ; or (ii) $(u, v) \in (E_G - E_H)$ and the relative distance between the bottom faces of the two cubes is $|c(u) - c(v)| \geq (t + 1) > (t + \epsilon)$, making them non-adjacent. □

We can directly compute a unit-cube contact representation for subgraphs of the square grid and the hexagonal grid, via geometric algorithms, instead of via threshold coloring [1]. To summarize the result of this section:

Theorem 4. *Any subgraph of the square and hexagonal grids has a unit-cube contact representation.*

6 Conclusion and Open Problems

We introduced threshold coloring and studied connections with other graph problems. Many interesting open problems remain, such as characterization and recognition of total-threshold-colorable graphs, and tightening the girth bounds.

Acknowledgments. We thank Torsten Ueckerdt, Carola Winzen and Michael Bekos for discussions about different variants of the threshold-coloring problem.

References

1. Alam, M.J., Chaplick, S., Fijavž, G., Kaufmann, M., Kobourov, S.G., Pupyrev, S.: Threshold-coloring and unit-cube contact representation of graphs. ArXiv report (2013), `http://arxiv.org/abs/1305.0069`
2. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph classes: A survey. Society for Industrial and Applied Mathematics (1999)
3. Bremner, D., et al.: On representing graphs by touching cuboids. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 187–198. Springer, Heidelberg (2013)
4. de Fegueiredo, C.M.H., Meidanis, J., de Mello, C.P.: A linear-time algorithm for proper interval graph recognition. Information Processing Letter 56(3), 179–184 (1995)
5. Felsner, S., Francis, M.C.: Contact representations of planar graphs with cubes. In: Symposium on Computational Geometry, pp. 315–320 (2011)
6. Golumbic, M.C., Kaplan, H., Shamir, R.: On the complexity of DNA physical mapping. Advances in Applied Mathematics 15(3), 251–261 (1994)
7. Golumbic, M.C., Kaplan, H., Shamir, R.: Graph sandwich problems. Journal of Algorithms 19(3), 449–473 (1995)
8. Hale, W.: Frequency assignment: Theory and applications. Proceedings of the IEEE 68(12), 1497–1514 (1980)
9. Hammer, P.L., Peled, U.N., Sun, X.: Difference graphs. Discrete Applied Mathematics 28(1), 35–44 (1990)
10. Mahadev, N.V.R., Peled, U.N.: Threshold Graphs and Related Topics. N. Holland (1995)
11. Roberts, F.S.: Indifference graphs. In: Proof Techniques in Graph Theory, pp. 139–146. Academic Press (1969)
12. Roberts, F.S.: From garbage to rainbows: Generalizations of graph coloring and their applications. Graph Theory, Combinatorics, and Applications 2, 1031–1052 (1991)
13. Thomassen, C.: Interval representations of planar graphs. Journal of Combinatorial Theory, Series B 40(1), 9–20 (1986)

Rolling Upward Planarity Testing of Strongly Connected Graphs

Christopher Auer, Christian Bachmaier,
Franz J. Brandenburg, and Kathrin Hanauer

University of Passau, Germany
{auerc,bachmaier,brandenb,hanauer}@fim.uni-passau.de

Abstract. A graph is upward planar if it can be drawn without edge crossings such that all edges point upward. Upward planar graphs have been studied on the plane, the standing and rolling cylinders. For all these surfaces, the respective decision problem $\mathcal{NP}$-hard in general. Efficient testing algorithms exist if the graph contains a single source and a single sink but only for the plane and standing cylinder.

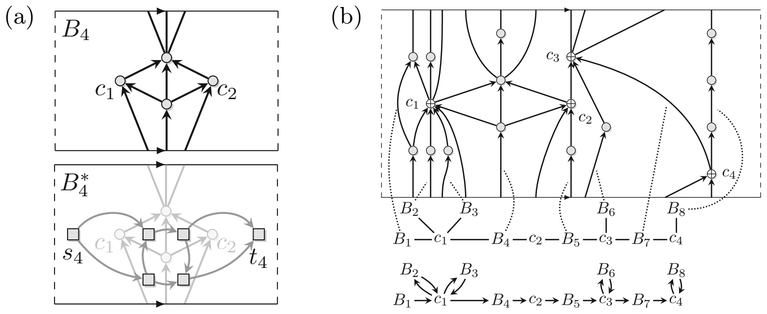
Here we show that there is a linear-time algorithm to test whether a strongly connected graph is upward planar on the rolling cylinder. For our algorithm, we introduce dual and directed SPQR-trees as extensions of SPQR-trees.

1 Introduction

A directed graph is upward planar (**UP**) if it has drawing without edge crossings such that all edge curves monotonically increase in y-direction. Upward planar graphs are of interesting in their own, but also arise in the context of the Sugiyama framework [22], which is the common drawing method for directed graphs. In the Sugiyama framework, the graphs are visualized as hierarchies, where all edges point upward, and it works particularly well for acyclic graphs. If the graph contains cycles, the Sugiyama algorithm is extended to recurrent hierarchies [3], where rolling upward planar (**RUP**) graphs naturally arise in the case of planarity.

The upward direction of the edges induces essential differences between planar and upward planar graphs. For instance, for planar graphs all surfaces of genus 0 are equivalent, such as the plane, the sphere, and the standing and rolling cylinders. The situation is different with upward planarity: There are directed graphs that have no upward drawing in the plane but on the surface of the standing cylinder on which edges may wind around the backside. In fact, there is a strict hierarchy of graph classes upward planar in the plane, on the (truncated) sphere [10,11,16,17] or on the standing cylinder [6,14,19,20,23], and the rolling cylinder [1,6]. Moreover, there are several linear-time planarity test algorithms, whereas upward planarity is $\mathcal{NP}$-complete in general [12].

In [2], we provide an overview on different types of upward planarity and investigate their relationships. There, we use the fundamental polygon representation: Let $I = (-1, 1)$ be the open interval from -1 to 1. The fundamental



$$\begin{tikzcd} B_2 \arrow[dr, shift left] & & B_3 \arrow[dl, shift left] & & & & B_6 \arrow[d, shift left] & & B_8 \arrow[d, shift left] \\ B_1 \arrow[r] & c_1 \arrow[ul, shift left] \arrow[ur, shift left] \arrow[r] & B_4 \arrow[r] & c_2 \arrow[r] & B_5 \arrow[r] & c_3 \arrow[r] \arrow[ur, shift left] & B_7 \arrow[r] & c_4 \arrow[ur, shift left] \end{tikzcd}$$

Fig. 1. **RUP** compounds and the (directed) block cut tree

polygon of the plane is $I \times I$, which is (the interior of) a square. By identifying its left and right (top and bottom) sides, we get the *standing* (*rolling*) cylinder. Fig. 1(a) top shows the fundamental polygon of the rolling cylinder, where the arrows at the bottom and top indicate their identification. A graph is *standing upward planar (**SUP**)* (*rolling upward planar (**RUP**)*) if it has a plane upward drawing on the standing (rolling cylinder). In both cases, the edge curves may wind around the cylinder and reappear at the identified sides of the fundamental polygon. An example of a **RUP** graph is shown in Fig. 1(a) at the top. Note that **RUP** graphs may contain cycles whereas **SUP** graphs are acyclic.

Acyclic dipoles are an important tool to study upward planar graphs on the plane and on cylinders. An acyclic dipole has a single source s and a single sink t and no cycles. A graph is **SUP** if and only if it can be augmented to an acyclic dipole by adding edges such that planarity is preserved [14,17,19]. If additionally the edge (s, t) can be added without destroying planarity, we obtain st-graphs, which characterize **UP** [8,18]. In [1], we use acyclic dipoles to characterize **RUP** graphs by means of their duals and investigate strongly connected graphs with at least one edge, which we call *compounds*. A compound is **RUP** if and only if its directed dual is an acyclic dipole (see bottom of Fig. 1(a)).

The **SUP** decision problem is $\mathcal{NP}$-complete [16]. In contrast, an acyclic dipole is **SUP** if and only if it is planar, which can be checked efficiently, and there is an efficient algorithm for triconnected single-source graphs [10]. Hence, the situations for **SUP** and **UP** are similar, as the latter is also efficiently solvable for single-source graphs [5]. The situation for **RUP** is alike: The general decision problem is $\mathcal{NP}$-complete [7] and in this paper we present an efficient algorithm for compounds that utilizes the dipole structure of the compound's dual. We divide the problem into two parts. First, we derive a characterization of **RUP** compounds by means of their block-cut trees which also yields a decision algorithm (Sect. 3). In the second part (Sect. 4), we tackle the blocks, i. e., the biconnected components, by using SPQR-trees to decide if a block is **RUP**. We conclude in Sect. 5.

2 Preliminaries

We consider connected, planar, directed graphs $G = (V, E)$ with vertices V and edges E. A plane *drawing* of G maps the vertices to distinct points and the edges to non-intersecting Jordan curves in the plane. A plane drawing induces an embedding and, equivalently, a rotation system which is the cyclic counter-clockwise ordering of the edges at every vertex. An embedding is **RUP** if it is obtained from a **RUP** drawing. An embedding of G specifies faces and defines the (directed) dual graph $G^* = (F, E^*)$ [4]. The vertices of G^* are the faces as defined in the embedding. To avoid confusion, we call the elements of V vertices and the elements F faces. There is a one-to-one correspondence between the edges of the *primal* G and the edges of its dual G^*. For each edge e in G, there is an edge e^* between faces f and f' in G^* if and only if e separates f and f'. If e is directed, edge e^* is oriented such that it points from the face left of e to the face right of e, when traversing the edge curve of e in its direction. Fig. 1(a) top shows a directed graph with its dual at the bottom. A face f is enclosed by edges at its *boundary* and by vertices which are *incident* to f. Note that G^* is always connected and G^* is acyclic if G is strongly connected. Whenever a dual graph G^* is given, then the primal is assumed to be embedded accordingly. Moreover, G^* can be computed from G's embedding in linear time.

In [1], we use directed duals to characterize **RUP** graphs.

Proposition 1 ([1]). *An embedded compound is* ***RUP*** *if and only if its dual is an acyclic dipole.*

Hence, we can efficiently decide whether an embeddig is **RUP** by testing if its dual is an acyclic dipole.

Corollary 1. *There is a linear-time algorithm to test whether the embedding of a compound is* ***RUP****.*

The embedding of a triconnected graph is unique up to *flipping*, i. e., inversion of all cyclic orderings in the rotation system.

Corollary 2. *There is a linear-time algorithm to test if a triconnected compound is* ***RUP****.*

Hence, deciding whether a compound is **RUP** can be done efficiently if the embedding is fixed. If no embedding is given, we use block-cut and SPQR-trees to find a **RUP** embedding. For a connected graph $G = (V, E)$, a block B is a subgraph induced by a set of edges such that no biconnected subgraph of G properly contains B. Two distinct blocks $B_i = (V_i, E_i)$ and $B_j = (V_j, E_j)$ may share a *cut vertex* $c \in V_i \cap V_j$. The *block-cut tree* $\mathcal{T}_B = (\mathcal{B}, \mathcal{C}, \mathcal{E}_B)$ is a tree, where $\mathcal{B} = (B_1, \ldots, B_k)$ $(k \geq 1)$ are the blocks, $\mathcal{C}$ are the cut vertices and there is an edge $\{B_i, c\} \in \mathcal{E}_B$ if c is in block B_i. Observe that a connected graph is strongly connected if and only if every block is strongly connected.

In the following, we call a directed path *dipath* and an undirected path simply *path*, e. g., of the underlying undirected graph. A dipath (path) with coinciding start and end is a *cycle* (*circle*). Any of these is called *simple* if it contains no vertex/edge twice.

3 Directed Block-Cut Trees of RUP-Compounds

First, we investigate the block-cut trees of **RUP** compounds and define the directed block-cut tree which we us to find a **RUP** embedding of a compound. As an example, consider the compound G in Fig. 1(b) consisting of blocks $B_1, \dots, B_8$ connected by cut vertices $c_1, \dots, c_4$, which are displayed by $\oplus$. A block B_i of a compound is strongly connected. Hence, its dual B_i^* is an acyclic dipole with source s_i and sink t_i (see also Fig. 1(a)). Source s_i is a face enclosed by the leftmost cycle C_i^l in the embedding of B_i and the vertices in C_i^l are incident to s_i. Accordingly, t_i corresponds to the rightmost cycle in B_i's embedding. A block contains at least one cycle that winds around the cylinder and, thus, divides it into two halves. Hence, a cut vertex shared by two blocks must be incident to the source or sink of the blocks' duals. For instance in Fig. 1(b), cut vertex c_1 is located at the rightmost cycle of block B_1 and at the leftmost cycle of B_2.

Lemma 1. *Let B_i and B_j be two blocks of a **RUP**-embedded compound sharing cut vertex c. Then, c is either incident to both s_i and t_j or to both s_j and t_i.*

A **RUP** embedding of a block B_i is *feasible* if each of the block's cut vertices is incident to the source or sink in B_i^*. Note that a cut vertex can be incident to both the source and the sink, e. g., c_1 in Fig. 1(b). By Lemma 1, a necessary condition for an embedding to be **RUP** is that each block's embedding is feasible.

Below the fundamental polygon in Fig. 1(b), the block-cut tree $\mathcal{T}_B$ of compound G is displayed. $\mathcal{T}_B$ has a linear structure in the following sense: The cut vertices and all blocks which contain two cut vertices, i. e., B_4, B_5, and B_7, form a path called *spine*. All other blocks contain only one cut vertex and "group" around the cut vertices on the spine. In fact, after removing all blocks with degree one from the directed block-cut tree only the spine remains and, hence, the block-cut tree of a **RUP** compound is a *caterpillar* [15], i. e., a tree where the removal of all leaves yields a path:

Lemma 2. *The block-cut tree of a **RUP** graph is a caterpillar.*

From Lemma 2, we immediately obtain that a block contains at most two cut vertices. We now define the *directed block-cut tree*, which describes the embedding of a compound with respect to its blocks and cut vertices. For the definition, we assume that each block is feasibly **RUP**-embedded and that the block-cut tree is a caterpillar. We obtain a total order $c_1, \dots, c_l$ on the set of cut vertices by traversing the spine of the caterpillar in either direction. The *directed block-cut tree* $\overrightarrow{\mathcal{T}_B} = (\mathcal{B}, \mathcal{C}, \overrightarrow{E}_B)$ contains all vertices of the original block-cut tree (see the bottom of Fig. 1(b)). Let B_i be a block which contains only cut vertex c_j. If c_j is incident to the source of B_i^*, we add edge (B_i, c_j) to $\overrightarrow{E}_B$ and if c_j is incident to the sink, we get edge (c_j, B_i). Let B_i be a block on the spine containing cut vertices c_j and c_{j+1}. There is an edge $(c_j, B_i) \in \overrightarrow{E}_B$ if c_j is incident to the source of B_i^* and $(B_i, c_{j+1}) \in \overrightarrow{E}_B$ if c_{j+1} is incident to the sink.

In the **RUP** embedding of B_4, c_1 is located at the leftmost cycle whereas c_2 is located at its rightmost cycle (see Fig. 1(a)). Hence, block B_4 has an incoming edge from c_1 and an outgoing edge to c_2. Note that $\overrightarrow{\mathcal{T}_B}$ is technically not a tree since it may contain antiparallel edge pairs whenever a cut vertex is incident to the source and sink of a block's dual. However, it still has a tree structure. Observe that the in- and outdegree of a block is at most one. Moreover, all "inner" cut vertices, i. e., c_2 and c_3, have the same in- and outdegree. Consider, for instance, c_3 in Fig. 1(b), which is attached to the right side of B_5, to both sides of B_6, and to the left side of B_7. Also, the indegree of c_1 equals its outdegree since B_1 is closing off its left side. Similarly, all "inner" blocks have equal in- and outdegree, i. e., blocks $B_2, \ldots, B_8$. Hence, the directed block cut tree has an Eulerian dipath $B_1, c_1, B_2, c_1, B_4, c_1, B_4, \ldots, c_4$. This observation holds in general and yields a characterization:

Lemma 3. *A compound is* ***RUP*** *if and only if every block has a feasible* ***RUP*** *embedding such that the directed block cut tree has an Eulerian dipath.*

If the directed block-cut tree has an Eulerian dipath, then this dipath visits each block exactly once and, hence, defines a total order on the blocks. The blocks can then be attached in order to each other at their shared cut vertices to obtain a **RUP** embedding for the whole compound.

Algorithm 1 returns a **RUP** embedding of a compound G or `false` if the compound is not **RUP**. As planarity is a necessary condition for a graph to be **RUP**, we assume that the compound is planar. The algorithm uses the linear-time subroutine `TestBiconnected`, which is the topic of Sect. 4. `TestBiconnected`(B_i, V_l, V_r) returns an embedding of block B_i such that all vertices in V_l and V_r are incident to the source and sink of B_i^*, respectively; if no such embedding exists, it returns `false`. First, Algorithm 1 checks whether G is biconnected. If this is the case, it directly calls `TestBiconnected`. Otherwise, it checks whether the block-cut tree $\mathcal{T}_B$ is a caterpillar. The spine of the caterpillar induces the total order $c_1, \ldots, c_l$ on the set of cut vertices. In the remainder of the algorithm, the directed block-cut tree $\overrightarrow{\mathcal{T}_B}$ is derived by testing the **RUP** embeddability of each block. Simultaneously, it stores the start and the end of the Eulerian dipath of $\overrightarrow{\mathcal{T}_B}$ in $\varepsilon_{\text{start}}$ and ε_{end}, respectively, with initial values $\varepsilon_{\text{start}} = c_1$ and $\varepsilon_{\text{end}} = c_l$. If there is a block B_i containing c_1 that has no embedding such that c_1 is incident to both the source and sink of B_i^*, B_i must have an embedding such that c_1 is incident to the sink of B_i^* (e. g., B_1 in Fig. 1(b)). Hence, B_i is the start of the Eulerian dipath and $\varepsilon_{\text{start}}$ is set to B_i. Note that no second block with the same properties can exist if the graph is **RUP** since otherwise $\overrightarrow{\mathcal{T}_B}$ has no Eulerian dipath. Likewise, the end of the Eulerian dipath is obtained. Finally, Algorithm 1 traverses the Eulerian dipath from $\varepsilon_{\text{start}}$ to ε_{end}, which induces a total order on the set of blocks, and assembles a **RUP** embedding of G. Note that `TestBiconnected` is called at most twice for each block and that the block-cut tree can be calculated in time $\mathcal{O}(|V| + |E|) = \mathcal{O}(|V|)$. The embeddings of all blocks can be merged in $\mathcal{O}(|V|)$. Also, all other steps have a running time linear in the size of either G or the block-cut tree. Hence, the overall running time of Algorithm 1 is $\mathcal{O}(|V|)$.

Algorithm 1. `TestCompound`

```
Input: Planar compound $G = (V, E)$
Output: RUP embedding of $G$ or false if $G$ is not RUP
1  if $G$ is biconnected then return TestBiconnected $(G, \emptyset, \emptyset)$
2  $\mathcal{T}_B = (\mathcal{B}, \mathcal{C}, \mathcal{E}_B) \leftarrow$ BlockCutTree $(G)$
3  if $\mathcal{T}_B$ is no caterpillar then return false
4  Order cut vertices from $c_1$ to $c_l$ according to the spine of $\mathcal{T}_B$
5  $\overrightarrow{\mathcal{T}_B} = (\mathcal{B}, \mathcal{C}, \overrightarrow{E}_B)$ with $\overrightarrow{E}_B = \emptyset$
6  $\varepsilon_{\text{start}} \leftarrow c_1$;  $\varepsilon_{\text{end}} \leftarrow c_l$
7  foreach $B_i = (V_i, E_i) \in \mathcal{B}$ do
8      if $B_i$ contains exactly one cut vertex $c_j$ then
9          if TestBiconnected $(B_i, \{c_j\}, \{c_j\}) \neq$ false then
10             $\overrightarrow{E}_B \leftarrow \overrightarrow{E}_B \cup \{(c_j, B_i), (B_i, c_j)\}$
11         else $B_i$ must be either the beginning or the end of the Eulerian dipath
12             if $c_j = \varepsilon_{\text{start}} \wedge$ TestBiconnected $(B_i, \emptyset, \{c_j\}) \neq$ false then
13                 $\varepsilon_{\text{start}} \leftarrow B_i$;  $\overrightarrow{E}_B \leftarrow \overrightarrow{E}_B \cup \{(B_i, c_j)\}$
14             else if $c_j = \varepsilon_{\text{end}} \wedge$ TestBiconnected $(B_i, \{c_j\}, \emptyset) \neq$ false then
15                 $\varepsilon_{\text{end}} \leftarrow B_i$;  $\overrightarrow{E}_B \leftarrow \overrightarrow{E}_B \cup \{(c_j, B_i)\}$
16             else return false
17     else $B_i$ contains two cut vertices $c_j$, $c_{j+1}$
18         if TestBiconnected $(B_i, \{c_j\}, \{c_{j+1}\}) \neq$ false then
19             $\overrightarrow{E}_B \leftarrow \overrightarrow{E}_B \cup \{(c_j, B_i), (B_j, c_{j+1})\}$
20         else return false
21 Construct RUP embedding $\mathcal{E}$ from the RUP embeddings of the blocks in order
   of the Eulerian dipath in $\overrightarrow{\mathcal{T}_B}$ from $\varepsilon_{\text{start}}$ to $\varepsilon_{\text{end}}$
22 return $\mathcal{E}$
```

4 Testing Biconnected Graphs

In the following, we define directed SPQR-trees of the primal and dual of biconnected compounds to store all possible embeddings. The main idea of the second phase of our algorithm is that if each nodes' skeleton of the primal directed SPQR-tree has an embedding whose dual is an acyclic dipole, then the dual of the whole graph is an acyclic dipole and, hence, the primal graph is **RUP**. For the **RUP** testing algorithm of biconnected graphs, we assume that the input graph $G = (V, E)$ contains no anti-parallel pairs of edges $(u, v), (v, u) \in E$, and no self-loops $(u, u) \in E$. Note that both wrap around the cylinder in a **RUP** embedding. All anti-parallel pairs of edges can be replaced by introducing a new vertex w and substituting edge (u, v) by edges (u, w) and (w, v). All self-loops are replaced alike. The so obtained graph is **RUP** if and only if the original graph is **RUP**.

SPQR-trees were introduced by Di Battista and Tamassia [9] and provide a description of how a graph is composed. Let G_u be an undirected biconnected graph, e. g., the underlying undirected graph of G. In the definition we adopt here, the SPQR-tree $\mathcal{T}$ of G_u is unrooted. The nodes of $\mathcal{T}$ either represent a series composition (S), a parallel composition (P), a single edge (Q), or a triconnected component (R). Associated with each node μ of $\mathcal{T}$ is a graph that is homeomorphic to a subgraph of G_u and called the *skeleton* $\text{skel}(\mu)$ of μ. In its original definition, every edge $e = \{u, v\}$ of a skeleton, except for one of a Q-node, is a *virtual edge*, i. e., an edge that represents the subgraph of G_u which connects u and v. This subgraph is also referred to as the *expansion graph* $\text{expg}(e)$ of e. For every virtual edge e in the skeleton of a node μ, there is another node ν that refines the structure of $\text{expg}(e)$. This link is represented by an edge between μ and ν in $\mathcal{T}$. Therefore, every leaf of $\mathcal{T}$ is a Q-node. For simplification, however, we represent edges of the graph directly in the skeleton of an S- ,P-, or R-node, so that we can neglect Q-nodes. If G_u is planar, its SPQR-tree stores all planar embeddings of G_u. They can be obtained by two basic operations: swapping two (virtual) edges in the skeleton of a P-node and flipping the triconnected component represented by an R-node. The embedding of G_u can be obtained by merging all skeletons at their associated virtual edges.

Let $\mathcal{E}$ be a planar embedding of G_u. Then, $\mathcal{E}$ implies an embedding of the skeleton of each node of $\mathcal{T}$. The *dual skeleton* $\text{skel}(\mu)^*$ of a node μ is the dual graph of $\text{skel}(\mu)$. The *dual SPQR-tree* $\mathcal{T}^*$ of G_u is a tree whose nodes' skeletons are the dual skeletons of $\mathcal{T}$ with types S^*, P^*, and R^*, and $\mathcal{T}^*$ inherits the topology of $\mathcal{T}$. Note that the dual graph of the skeleton of a P-node yields a circle of length at least 3, which corresponds to an S-node, and the dual of the skeleton of an S-node is a P-node in turn. Also, the dual of a triconnected and embedded graph is triconnected [21, Thm. 2.6.7] and, hence, the dual of an R-node's skeleton is triconnected as well. Consequently, for the node types of the dual SPQR-tree holds $S^* = P$, $P^* = S$, and $R^* = R$. Moreover, merging all skeletons of $\mathcal{T}^*$ at the associated virtual edges yields the dual graph G_u^* of G_u.

Lemma 4. *The dual SPQR-tree of a graph G_u with embedding $\mathcal{E}$ is the SPQR-tree of the dual graph G_u^* with $S^* = P$, $P^* = S$, and $R^* = R$.*

In the next step, we extend SPQR-trees to directed graphs. Let G be a directed, biconnected graph and $\mathcal{T}$ be the SPQR-tree of its underlying undirected graph. For every skeleton $\text{skel}(\mu)$ of a node μ in $\mathcal{T}$, we obtain the *directed skeleton* $\overrightarrow{\text{skel}}(\mu)$ by directing all non-virtual edges according to their direction in G. See Figs. 2(b) and (c) for an example. In the following, we call an acyclic dipole with source u and sink v a *uv-graph.* Each virtual edge $e = \{u, v\}$ is treated as follows: If its expansion graph $\text{expg}(e)$ is either a uv-graph or a vu-graph, e is represented as a directed edge (u, v) or (v, u), respectively. Otherwise, e remains undirected and is mapped to a subset of the flags $\{\odot, ◀▶, ▶◀\}$, depending on whether $\text{expg}(e)$ contains a source (◀▶) or a sink (▶◀) other than u and v, or a cycle ($\odot$). The *directed SPQR-tree* $\overrightarrow{\mathcal{T}}$ is obtained from $\mathcal{T}$ by replacing each skeleton by its directed counterpart. Likewise, we obtain the *directed dual SPQR-tree* $\overrightarrow{\mathcal{T}^*}$ for a planar embedding $\mathcal{E}$ from $\mathcal{T}^*$, see Figs. 2(c) and (d).

Observe that with the flags, every skeleton in the directed SPQR-tree stores information whether the graph contains cycles, sources, or sinks, and, to a certain

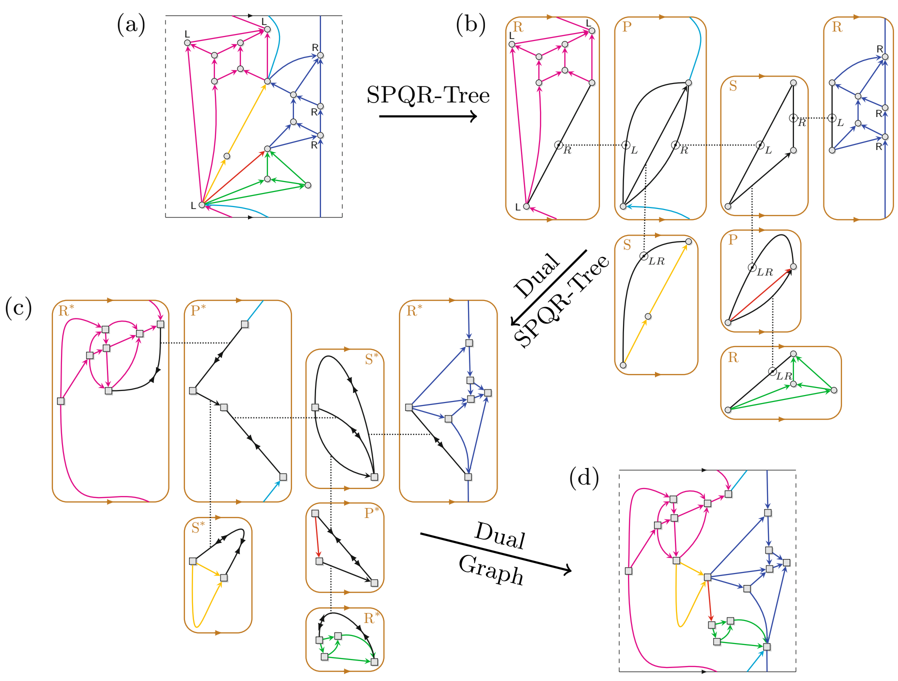


Fig. 2. A compound with triconnected components and its dual together with their SPQR-trees. Black lines are virtual edges. Dotted edges connect the nodes of the SPQR-tree and indicate the associated virtual edges.

extent also how many. This implies transitivity for the flags $\odot$, $\blacktriangleleft\!\blacktriangleright$, $\blacktriangleright\!\blacktriangleleft$: Let μ be a node containing a virtual edge e that is refined by node ν and let e' be the virtual edge in ν that is refined by μ. If any virtual edge $e'' \neq e'$ in ν carries flag $X \in \{\odot, \blacktriangleleft\!\blacktriangleright, \blacktriangleright\!\blacktriangleleft\}$, then e in μ also has flag X.

The planar embedding of a graph is uniquely defined by the planar embeddings of the skeletons in its SPQR-tree and it can be obtained by merging the nodes at the associated virtual edges. Note that the direction or the flags of a virtual edge are insignificant for the merge operations since the vertices that are matched are always those that represent the same vertex in the graph.

Corollary 3. *The directed dual SPQR-tree of a graph G with embedding $\mathcal{E}$ is the directed SPQR-tree of the dual graph G^* of G.*

From now on, we only consider the biconnected and embedded compound G with acyclic dual G^*. Note that for a strongly connected graph, all virtual edges in the skeletons of its directed SPQR-tree are either directed or carry flag $\odot$, whereas for acyclic graphs, a virtual edge may only have one or both of the flags $\blacktriangleleft\!\blacktriangleright$ and $\blacktriangleright\!\blacktriangleleft$.

We introduce optional labels L and R for vertices of G and refine the flag $\odot$ on all virtual edges as follows: If the expansion graph of a virtual edge e with

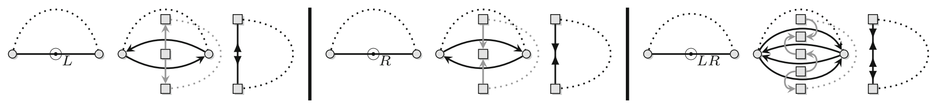


Fig. 3. Auxiliary skeleton

flag $\odot$ contains a vertex with label L, then e's flag is refined to $\odot_L$. Analogously, if $\text{expg}(e)$ contains a vertex with label R, then e's flag is refined to $\odot_R$. In case that $\text{expg}(e)$ contains vertices with both labels, e carries both flags. We say that a directed SPQR-tree is *labeled*, if every $\odot$-flag has been refined. Observe that this implies at least one vertex with label L or R in G. A directed SPQR-tree of an acyclic graph is trivially labeled.

Consider graph G^*, which is acyclic, and contains at least one source and one sink. Every source and every sink in G^* is a face which is bounded by a cycle in G. We assign the label L to every vertex that is incident to a source, and the label R to every vertex that is incident to a sink. Note that a vertex may have both labels.

Lemma 5. *There is a labeled directed SPQR-tree for G such that exactly the vertices incident to a source in G^* are labeled* L *and those incident to a sink are labeled* R.

Lemma 5 enables us to obtain the directed dual SPQR-tree $\overrightarrow{\mathcal{T}^*}$ of G directly from the directed SPQR-tree $\overrightarrow{\mathcal{T}}$ that is labeled according to its embedding. If a virtual edge in the skeleton of a node in $\overrightarrow{\mathcal{T}}$ has flag $\odot_L$ or $\odot_R$, we can assign to its dual edge the flag ◀▶ or ▶◀, respectively. The dual of a virtual edge with flag ◀▶ or ▶◀ then is a virtual edge with flag $\odot_R$ or $\odot_L$, respectively. Observe that for directed graphs, the dual of a dual graph is the *converse* of the primal graph, i. e., G with all edges reversed.

Let $\overrightarrow{\text{skel}}(\mu)$ be the skeleton of a node μ in a labeled directed SPQR-tree $\overrightarrow{\mathcal{T}}$ of a graph G. The *auxiliary skeleton* $\text{aux}(\overrightarrow{\text{skel}}(\mu))$ is derived from $\overrightarrow{\text{skel}}(\mu)$ by substituting each undirected virtual edge $\{u, v\}$ according to its flags by a *proxy* as depicted in Fig. 3. For instance, the proxy of a virtual edge with flag $\odot_L$ is an anti-parallel pair of edges, such that it induces a source in the dual graph, which is in turn the proxy of a virtual edge with flag ◀▶. If G has a planar embedding, the new edges inherit the position of $\{u, v\}$ in the rotation systems at u and v. Observe that all constructions preserve planarity.

By Proposition 1, G is **RUP** if and only if its dual G^* is an acyclic dipole. Consider the directed SPQR-tree $\overrightarrow{\mathcal{T}^*}$ of G^*. Then, the skeleton of every node represents the structure of G^* either directly or via flags on its virtual edges.

Lemma 6. *G^* is an acyclic dipole if and only if every auxiliary skeleton of its directed SPQR-tree is an acyclic dipole.*

Algorithm 2. `TestBiconnected`

```
Input: Planar biconnected compound $G = (V, E)$, $V_l, V_r \subseteq V$
Output: RUP embedding of $G$ such that all $v \in V_l$ ($V_r$) are incident to the
        source (sink) of $G^*$; or false if no such RUP embedding exists
1  $\mathcal{T} \leftarrow$ ComputeSPQRTree $(G)$ and keep edge directions according to $G$
2  Obtain $\overrightarrow{\mathcal{T}}$: Root the tree arbitrarily, direct all edges towards root, and let
   $\mu_1, \dots, \mu_k$ be a topological sorting of $\mathcal{T}$'s nodes
3  foreach $\mu = \mu_1, \dots, \mu_k$ do ignoring virtual edge associated with parent
4      if $\overrightarrow{\text{skel}}(\mu)$ contains virtual edge with flag $\odot$ or a cycle then
5          Set flag $\odot$ on virtual edge of parent associated with $\mu$
6      else $\overrightarrow{\text{skel}}(\mu)$ is $uv$-graph
7          Direct virtual edge of parent associated with $\mu$ from $u$ to $v$
8  Propagate completive information from root to children
9  if $\exists$ skeleton with $> 2$ $\odot$-flags then return false
10 foreach node $\mu$ and vertex $v$ in $\overrightarrow{\text{skel}}(\mu)$ do if $v \in V_l$ ($v \in V_r$) then
11     label $v$ with $\mathsf{L}$ ($\mathsf{R}$) in $\overrightarrow{\text{skel}}(\mu)$
12 $\overrightarrow{\text{skel}}(\mu) \leftarrow$ skeleton with maximum number of virtual edges with flag $\odot$
13 if $\overrightarrow{\text{skel}}(\mu)$ contains $\geq 1$ virtual edges with flag $\odot$ then
14     foreach $f \in \{\odot_L, \odot_R, \odot_L \,\&\, \odot_R\}$ do
15         Refine $\odot$-flag on one virtual edge to $f$
16         Refine $\odot$-flag on other virtual edge, if existent, to complement of $f$
17         Establish transitivity
18         if TestRUPLabeledSPQRTree $(\overrightarrow{\mathcal{T}}) \neq$ false then goto 22
19     return false
20 else no virtual edges with $\odot$-flag
21     if TestRUPLabeledSPQRTree $(\overrightarrow{\mathcal{T}}) =$ false then return false
22 Build and return the RUP embedding of $G$ by merging all skeletons
```

Algorithm 2 takes as input a planar, biconnected compound $G = (V, E)$ with two sets $V_l, V_r \subseteq V$. It returns a **RUP** embedding of G such that all vertices in V_l (V_r) are incident to the source (sink) of G^* or `false` if no such embedding exists. First, the SPQR-tree of G is computed which can be done in linear time [13]. From lines 1–8 the directed SPQR-tree $\overrightarrow{\mathcal{T}}$ is obtained, i. e., each virtual edges is either directed or the $\odot$ flag is set. Note that for each skeleton an algorithm is executed whose running time is linear in the number of vertices in the skeleton. Since the number of edges represented in the skeletons is in $\mathcal{O}(|V|)$ [9], we obtain a running time linear in the size of G. By Lemma 6, the dual of every skeleton must be an acyclic dipole. Hence, the primal skeleton can contain at most one virtual edge with $\odot_L$- and one with $\odot_R$-flag and, thus, at most two $\odot$-edges (cf. line 9). Next, in the skeletons, the representatives of the vertices in V_l and V_r are labeled accordingly. In lines 13–22, the algorithm tries all transitive refinements of the $\odot$-flag. Note that if there is a skeleton with two cylic edges e_1, e_2, either e_1 gets $\odot_L$ and e_2 gets $\odot_R$ or vice versa.

In a skeleton with one $\odot$-edge, the three refinements $\odot_L$,$\odot_R$, and $\odot_L$ & $\odot_R$ are possible. If all $\odot$-edges in one skeleton have been refined, the refinements of all other $\odot$-edges are determined uniquely by transitivity. Thus, the algorithm has to test at most three different refinements altogether. The obtained labeled SPQR-tree is passed to `TestRUPLabeledSPQRTree`. This routine verifies that each skeleton has an embedding such that its dual is an acyclic dipole and the vertices with labels L and R are incident to the source and sink, respectively. `TestRUPLabeledSPQRTree` proceeds for every node μ as follows: If μ is of type R, its auxiliary skeleton is obtained. A planarity test yields the (up to flipping) unique embedding of the auxiliary skeleton. If one dual is an acyclic dipole, then so is the other. If in one of them the vertices with label L (R) are incident to the source (sink), this embedding is retained. Otherwise, the test aborts and returns `false`. If μ is a P-node, we define the rotation system at any of the two vertices in the skeleton as follows: Denote by $e_1^+, \ldots, e_m^+$ and by $e_1^-, \ldots, e_n^-$ all outgoing and incoming edges of the vertex in arbitrary order. The rotation system is then $e_1^+, \ldots, e_m^+, e_{\odot L}, e_1^-, \ldots, e_n^-, e_{\odot R}$. Observe that if the $\odot$-edge carries both flags, then either $m = 0$ or $n = 0$. Because of planarity, this rotation system also defines the rotation system at the other vertex. If one or both of the $\odot$-edges are missing, they are skipped. In all cases, the dual is an acyclic dipole. Note that whenever an anti-parallel pair of edges is adjacent in the rotation system, this always results in either a source or sink in the dual, and the duals of the proxies of $e_{\odot L}$ and $e_{\odot R}$ are a source and a sink, respectively. From the perspective a P-node provides on the whole graph, the split pair can be incident to both the source and sink of the dual. Hence, no further tests are needed. If μ is an S-node, its skeleton has a unique embedding and, hence, also its auxiliary skeleton. The algorithm computes the dual and verifies that the vertices with L and R are incident to the source and sink, respectively. If all checks succeed, `TestRUPBiconnected` returns a **RUP** embedding of G. For every skeleton, the test can be performed in time linear in the size of the skeleton. Finally, we obtain:

Theorem 1. *There is a linear-time algorithm to decide whether a strongly connected graph is* ***RUP****.*

5 Conclusion

We presented a linear time algorithm to check whether a strongly connected graph is drawable upward planar on a rolling cylinder. We were also able to extend the **RUP**-test to graphs without sources and sinks, i. e., graphs consisting of multiple compounds. The proof is left as future work.

References

1. Auer, C., Bachmaier, C., Brandenburg, F.J., Gleißner, A., Hanauer, K.: The duals of upward planar graphs on cylinders. In: Golumbic, M.C., Stern, M., Levy, A., Morgenstern, G. (eds.) WG 2012. LNCS, vol. 7551, pp. 103–113. Springer, Heidelberg (2012)

2. Auer, C., Bachmaier, C., Brandenburg, F.J., Gleißner, A.: Classification of planar upward embedding. In: van Kreveld, M., Speckmann, B. (eds.) GD 2011. LNCS, vol. 7034, pp. 415–426. Springer, Heidelberg (2011)
3. Bachmaier, C., Brandenburg, F.J., Brunner, W., Fülöp, R.: Drawing recurrent hierarchies. J. Graph Alg. App. 16(2), 151–198 (2012)
4. Bang-Jensen, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications, 1st edn. Springer (2000)
5. Bertolazzi, P., Battista, G.D., Mannino, C., Tamassia, R.: Optimal upward planarity testing of single-source digraphs. SIAM J. Comput. 27(1), 132–169 (1998)
6. Brandenburg, F.J.: On the curve complexity of upward planar drawings. In: Mestre, J. (ed.) Computing: The Australasian Theory Symposium, CATS 2012. CRPIT, vol. 128, pp. 27–36. Australian Computer Society (2012)
7. Brandenburg, F.J.: Upward planar drawings on the standing and the rolling cylinders. Comput. Geom. Theory Appl. (to appear, 2013)
8. Di Battista, G., Tamassia, R.: Algorithms for plane representations of acyclic digraphs. Theor. Comput. Sci. 61(2-3), 175–198 (1988)
9. Di Battista, G., Tamassia, R.: On-line planarity testing. SIAM J. Comput. 25(5), 956–997 (1996)
10. Dolati, A., Hashemi, S.M.: On the sphericity testing of single source digraphs. Discrete Math. 308(11), 2175–2181 (2008)
11. Foldes, S., Rival, I., Urrutia, J.: Light sources, obstructions and spherical orders. Discrete Math. 102(1), 13–23 (1992)
12. Garg, A., Tamassia, R.: On the computational complexity of upward and rectilinear planarity testing. SIAM J. Comput. 31(2), 601–625 (2001)
13. Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 77–90. Springer, Heidelberg (2001)
14. Hansen, K.A.: Constant width planar computation characterizes ACC^0. Theor. Comput. Sci. 39(1), 79–92 (2006)
15. Harary, F., Schwenk, A.J.: The number of caterpillars. Discrete Math. 6(4), 359–365 (1973)
16. Hashemi, S.M., Rival, I., Kisielewicz, A.: The complexity of upward drawings on spheres. Order 14, 327–363 (1998)
17. Hashemi, S.M.: Digraph embedding. Discrete Math. 233(1-3), 321–328 (2001)
18. Kelly, D.: Fundamentals of planar ordered sets. Discrete Math. 63, 197–216 (1987)
19. Limaye, N., Mahajan, M., Sarma M.N., J.: Evaluating monotone circuits on cylinders, planes and tori. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 660–671. Springer, Heidelberg (2006)
20. Limaye, N., Mahajan, M., Sarma, J.M.N.: Upper bounds for monotone planar circuit value and variants. Comput. Complex. 18(3), 377–412 (2009)
21. Mohar, B., Thomassen, C.: Graphs on Surfaces. Johns Hopkins Series in the Mathematical Sciences. Johns Hopkins University Press, Baltimore (2001)
22. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE-SMC 11(2), 109–125 (1981)
23. Thomassen, C.: Planar acyclic oriented graphs. Order 5(1), 349–361 (1989)

Towards a Provably Resilient Scheme for Graph-Based Watermarking*

Lucila Maria Souza Bento[1,2], Davidson Boccardo[2],
Raphael Carlos Santos Machado[1,2], Vinícius Gusmão Pereira de Sá[1],
and Jayme Luiz Szwarcfiter[1,2,3]

[1] Instituto de Matemática, Universidade Federal do Rio de Janeiro,
Rio de Janeiro, Brasil
`lucilabento@ppgi.ufrj.br`, `vigusmao@dcc.ufrj.br`

[2] Instituto Nacional de Metrologia, Qualidade e Tecnologia, Rio de Janeiro, Brasil
`{drboccardo,rcmachado}@inmetro.gov.br`

[3] COPPE-Sistemas, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil
`jayme@nce.ufrj.br`

Abstract. Techniques of watermarking/fingerprinting concern the embedding of identification data into a digital object, allowing for later claims of authorship/ownership and therefore discouraging piracy. Graph-based watermarking schemes comprise an encoding algorithm, which translates a given number (the identifier, usually a positive integer) onto some appropriately tailored graph (the watermark), and a decoding algorithm, which extracts the original identifier from a given watermark. Collberg, Kobourov, Carter and Thomborson (*Error-correcting graphs for software watermarking*, WG'03) introduced one such scheme, meant for software watermarking, in which an integer key was encoded onto a reducible permutation graph. A number of interesting ideas have further improved the original scheme, including the formulation of a particularly promising linear-time codec by Chroni and Nikolopoulos. We extend the work of these authors in various aspects. First, we characterize the class of graphs constituting the image of Chroni and Nikolopoulos's encoding function. Furthermore, we formulate a novel, linear-time decoding algorithm which detects and recovers from ill-intentioned removals of $k \leq 2$ edges. Finally, our results also include the detection of $k \leq 5$ edge modifications (insertions/deletions) in polynomial time and a proof that such bound is tight, so the resilience of the considered watermarking scheme is fully determined. Our proof that graphs of a well-characterized class can detect and recover from bounded-magnitude distortive attacks reinforces the interest in regarding those graphs as possible watermarking solutions to numerous applications.

Keywords: algorithms, reducible permutation graphs, graph-based watermarking, software security.

* Work partially supported by Eletrobrás — Distribuição Rondônia, DR/069/2012.

1 Introduction

Preventing the theft of intellectual property has always been a highly sought after goal. In particular, the criminal reproduction of software known as software piracy has become a big concern in recent years. Watermarking/fingerprinting an object is the act of embedding an identifier of authorship/ownership within that object, so to discourage illegal copying.

Different approaches to software watermarking have been devised to date, still none of them was ever proved to be sufficiently resilient, let alone immune, to the numerous forms of program transformation attacks. Naturally, a lot of research towards strengthening such methods has been endeavored. The pioneering graph-based watermarking algorithm was formulated by Davidson and Myrhvold [10]. It inspired the publication, by Venkatesan, Vazirani and Sinha [14], of the first watermarking algorithm where a positive integer—the *key*—was encoded as a special digraph that could be disguised into the control flow graph (CFG) of a program. Other graph-based watermarking schemes include [6,8,9,13].

In this paper, we consider the ingenious graph-based watermarking scheme introduced by Collberg, Kobourov, Carter and Thomborson [7], and afterwards developed and improved upon by Chroni and Nikolopoulos [1].[1] These latter authors proposed a linear-time *codec* (an encoding/decoding procedure) to obtain watermarks that are particular instances of the reducible permutation graphs introduced in [7]. Chroni and Nikolopoulos's watermarks possess important structural properties and are also meant to be embedded into the CFG of the software to be protected.[2] Though the mechanics of the proposed codec is well described in [1], the class of graphs that constitute the generated watermarks has not been fully characterized. Moreover, not much was known thus far about the resilience of Chroni and Nikolopoulos's graphs to malicious attacks, even though their ability to withstand attacks in the form of a single edge modification has been suggested without proofs. A thorough scrutiny of the structural properties of the aforementioned graph class allowed us to give it a formal characterization, as well as to introduce a linear-time decoding algorithm that retrieves the correct, untampered with encoded key even when $k \leq 2$ edges of the watermark are missing. Such algorithm allows for the determination of the exact resilience level of such graphs against distortive attacks.

The paper is organized as follows. In Section 2, we recall the codec from [1], formulating and proving a number of properties of the employed structures. In Section 3, we characterize the class of canonical reducible permutation graphs. In Section 4, we tackle the edge-removal attack model, proving that, for keys of size $n > 2$, it is always possible to identify and recover from attacks that remove $k \leq 2$ edges. Finally, in Section 5, we formulate a linear-time algorithm that reconstructs the original digraph, in case $k \leq 2$ edges are missing, recovering

[1] A series of papers on watermarking by the same authors include, but is not limited to, [2–5].

[2] It is not in the scope of this paper the discussion of techniques to embed the watermark graph into a CFG.

the encoded key thereafter. As a corollary of the results hitherto presented, we fully determine the resilience of the considered watermarking scheme. Section 6 concludes the paper with our final remarks and future directions. Throughout the paper, many proofs were omitted due to space constraints, and shall be included in a future extended version of the text.

2 The Watermark by Chroni and Nikolopoulos

We start by briefly recalling the watermarking codec described in [1].

Let ω be a positive integer key, with n the size of the binary representation B of ω. Let also n_0 and n_1 be the number of 0's and 1's, respectively, in B, and let f_0 be the index[3] of the leftmost 0 in B. The extended binary B^* is obtained by concatenating n digits 1, followed by the one's complement[4] of B and by a single digit 0. We let $n^* = 2n + 1$ denote the size of B^*, and we define $Z_0 = (z^0_i)$, $i = 1, \ldots, n_1 + 1$, as the ascending sequence of indexes of 0's in B^*, and $Z_1 = (z^1_i)$, $i = 1, \ldots, n + n_0$, as the ascending sequence of indexes of 1's in B^*.

Let S be a sequence. We denote by S^R the sequence formed by the elements of S in backward order. If $S = (s_i)$, $i = 1, \ldots, t$, is a sequence of size t, and there is an integer $k \leq t$ such that the subsequence consisting of the elements of S with indexes less than or equal to k is ascending, and the subsequence consisting of the elements of S with indexes greater than or equal to k is descending, then we say S is *bitonic*. If all t elements of a sequence S are distinct and belong to $\{1, \ldots, t\}$, then S is a *permutation*. If S is a permutation of size t, and, for all $1 \leq i \leq t$, the equality $i = s_{s_i}$ holds, then we say S is *self-inverting*. In this case, the unordered pair (i, s_i) is called a 2-*cycle* of S, if $i \neq s_i$, and a 1-*cycle* of S, if $i = s_i$. If S_1, S_2 are sequences, we denote by $S_1||S_2$ the sequence formed by the elements of S_1 followed by the elements of S_2.

Back to Chroni and Nikolopoulos's codec, the bitonic permutation $P_b = Z_0||Z_1^R = (b_i)$, $i = 1, \ldots, n^*$, is obtained by appending to Z_0 the elements of Z_1 in backward order, and, finally, the self-inverting permutation P_s is obtained from P_b as follows: for $i = 1, \ldots, n^*$, index b_i in P_s is assigned to element $s_{b_i} = b_{n^*-i+1}$, and index b_{n^*-i+1} in P_s is assigned to element $s_{b_{n^*-i+1}} = b_i$. In other words, the 2-cycles of P_s correspond to the n unordered pairs of distinct elements of P_b that are equidistant from the extremes of P_b, namely the pairs $(p, q) = (b_i, b_{n^*-i+1})$, for $i = 1, \ldots, n$. Since the central index $i = n + 1$ of P_b is the solution of equation $n^* - i + 1 = i$, element b_{n+1}—and no other—will constitute a 1-cycle in P_s. We refer to such element of P_s as its *fixed element*, and we let f denote it.

The watermark generated by the codec from [1] belongs to the class of reducible permutation graphs first defined in [7]. It is a directed graph G whose vertex set is $\{0, 1, \ldots, 2n + 2\}$, and whose edge set contains $4n + 3$ edges, to wit:

[3] The index of the leftmost element in all sequences considered in the text is 1.

[4] The *one's complement* of a binary R is obtained by swapping all 0's in R for 1's and vice-versa.

a *path edge* $(u, u-1)$ for $u = 1, \ldots, 2n+2$, constituting a Hamiltonian path that will be unique in G, and a *tree edge* from u to $q(u)$, for $u = 1, \ldots, n^*$, where $q(u)$ is defined as the vertex $v > u$ with the greatest index in P_s to the left of u, if such v exists, or $2n + 2$ otherwise[5]. A graph so obtained is called a *canonical reducible permutation graph.*

Let us glance at an example. For $\omega = 43$, we have $B = 101011$, $n = 6$, $n_0 = 2$, $n_1 = 4$, $f_0 = 2$, $B^* = 1111110101000$, $n^* = 13$, $Z_0 = (7, 9, 11, 12, 13)$, $Z_1 = (1, 2, 3, 4, 5, 6, 8, 10)$, $P_b = (7, 9, 11, 12, 13, 10, 8, 6, 5, 4, 3, 2, 1)$, $P_s = (7, 9, 11, 12, 13, 10, 1, 8, 2, 6, 3, 4, 5)$ and $f = 8$. The generated watermark G will have, along with the path edges in Hamiltonian path $14 \to 13 \to \cdots \to 0$, also the tree edges $(1, 10), (2, 8), (3, 6), (4, 6), (5, 6), (6, 8), (7, 14), (8, 10), (9, 14), (10, 13), (11, 14), (12, 14)$ and $(13, 14)$, as illustrated in Figure 1.

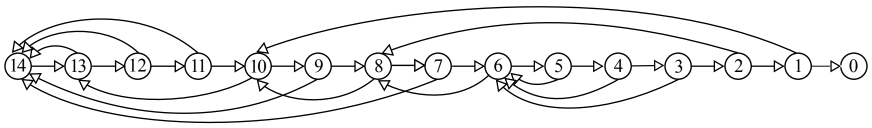


Fig. 1. Watermark for key $\omega = 43$

Canonical reducible permutation graphs are certainly not the only graphs that could be used to encode an integer, and they are certainly not the only graphs that could be disguised into a software's CFG. Yet they do have such properties, hence they are an appropriate choice. Moreover, they present some structural, encoding-related redundancy that grants them some resilience against attacks, as we shall see. We now state a number of properties of these graphs.

Let G still be the canonical reducible permutation graph associated to a key ω of size n, and P_b and P_s, respectively, the bitonic and the self-inverting permutations dealt with during the construction of G.

Property 1. *For $1 \leq i \leq n$, element b_{n+i+1} in P_b is equal to $n - i + 1$, that is, the n rightmost elements in P_b are $1, 2, \ldots, n$ when read from right to left.*

Property 2. *The elements whose indexes are $1, 2, \ldots, n$ in P_s are all greater than n.*

Property 3. *The fixed element f satisfies $f = n + f_0$, unless the key ω is equal to $2^k - 1$ for some integer k, whereupon $f = n^* = 2n + 1$.*

Property 4. *In self-inverting permutation P_s, elements indexed $1, 2, \ldots, f - n - 1$ are respectively equal to $n + 1, n + 2, \ldots, f - 1$, and elements indexed $n + 1, n + 2, \ldots, f - 1$ are respectively equal to $1, 2, \ldots, f - n - 1$.*

Property 5. *The first element in P_s is $s_1 = n + 1$, and the central element in P_s is $s_{n+1} = 1$.*

[5] The rationale behind the name *tree edge* is the fact that such edges induce a spanning tree of $G \setminus \{0\}$.

Property 6. *If $f \neq n^*$, then the index of element n^* in P_s is equal to $n_1 + 1$, and vice-versa. If $f = n^*$, then the index of element n^* in P_s is also n^*.*

Property 7. *The subsequence of P_s consisting of elements indexed $1, 2, \ldots, n{+}1$ is bitonic.*

Property 8. *For $u \neq 2n+1$, $(u, 2n+2)$ is a tree edge of watermark G if, and only if, $u - n$ is the index of a digit 1 in the binary representation B of the key ω represented by G.*

Property 9. *If (u, k) is a tree edge of watermark G, with $k \neq 2n+2$, then (i) element k precedes u in P_s; and (ii) if v is located somewhere between k and u in P_s, then $v < u$.*

3 Canonical Reducible Permutation Graphs

This section is devoted to the characterization of the class of canonical reducible permutation graphs. After describing some terminology, we define the class using purely graph-theoretical predicates, then we prove it corresponds exactly to the set of all watermark instances possibly produced by Chroni and Nikolopoulos's encoding algorithm [1]. Finally, we characterize it in a way that suits the design of a new, resilient, linear-time decoding algorithm.

Given a graph G, we let $V(G)$ and $E(G)$ denote, as usual, the vertex set and edge set of G, respectively. Also, we let $N^+_G(v)$ and $N^-_G(v)$ respectively denote the set of out-neighbors and in-neighbors of vertex $v \in V(G)$. A *reducible flow graph* [11, 12] is a directed graph G with source $s \in V(G)$, such that, for each cycle C of G, every path from s to C reaches C at a same vertex. It is well known that a reducible flow graph has at most one Hamiltonian cycle.

Definition 10. *A* self-labeling reducible flow graph *relative to $n > 1$ is a directed graph G s.t.*

(i) $|V(G)| = 2n + 3$;
(ii) G presents exactly one directed Hamiltonian path, hence there is a unique labeling function $\sigma : V(G) \to \{0, 1, \ldots, 2n+2\}$ of the vertices of G such that the order of the labels along the Hamiltonian path is precisely $2n+2, 2n+1, \ldots, 0$;
(iii) considering the labeling σ as in the previous item, $N^+_G(0) = \emptyset$, $N^-_G(0) = \{1\}$, $N^+_G(2n+2) = \{2n+1\}$, $|N^-_G(2n+2)| \geq 2$, and, for all $v \in V(G) \setminus \{0, 2n+2\}$, $N^+_G(v) = \{v-1, w\}$, for some $w > v$.

From now on, without loss of generality, we shall take σ for granted and assume the vertex set of any self-labeling reducible flow graph G *is* the very set $V(G) = \{0, 1, \ldots, 2n+2\}$, yielding the unique Hamiltonian path $2n+2, 2n+1, \ldots, 0$ in G.[6]

[6] By doing so, we may simply compare two vertices, e.g. $v > u$ (or v greater than u, in full writing), whereas we would otherwise need to compare their images under σ, e.g. $\sigma(v) > \sigma(u)$.

Let G be a self-labeling reducible flow graph and H its unique Hamiltonian path. We define a tree T with vertex set $V(T) = V(G) \setminus \{0\}$, and edge set $E(T)$ comprising all edges in $E(G) \setminus E(H)$ deprived of their orientation. We call T the *representative tree* of G, and we regard it as a *rooted* tree whose root is $2n+2$, and where the children of each $v \in V(T)$, denoted $N_T(v)$, are exactly the in-neighbors of v in $G \setminus E(H)$. In addition, we regard T as an *ordered* tree, that is, for each $v \in V(T)$, the children of v are always considered according to an ascending order of their labels. Finally, for $v \in T$, we denote by $N^*_T(v)$ the set of descendants of v in T. Figure 2 depicts two representative trees.

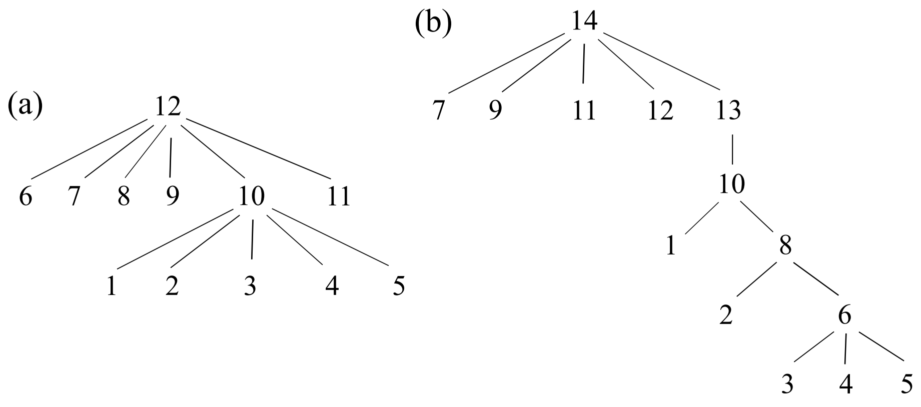


Fig. 2. Representative trees of the watermarks for keys (a) $\omega = 31$ and (b) $\omega = 43$

Observation 11. *The representative tree T of a self-labeling reducible flow graph G satisfies the max-heap property, that is, if vertex u is a child of vertex v in T, then $v > u$.*

We convey the idea that a representative tree T satisfies the max-heap property by saying that T is a *descending*, ordered, rooted tree.

Definition 12. *A self-inverting permutation S of size $2n+1$ is said to be* canonical *if:*

(i) *there is exactly one 1-cycle in S;*
(ii) *each 2-cycle (s_i, s_j) of S satisfies $1 \leq i \leq n$, for $s_i > s_j$;*
(iii) *$s_1, \ldots, s_{n+1}$ is a bitonic subsequence of S starting at $s_1 = n+1$ and ending at $s_{n+1} = 1$.*

Lemma 13. *In any canonical self-inverting permutation of $\{1, \ldots, 2n+1\}$, the fixed element f satisfies $f \in [n+2, 2n+1]$.*

Let T be some representative tree, therefore a descending, ordered, rooted tree. The *preorder traversal* P of T is a sequence of its vertices that is recursively defined as follows. If T is empty, P is also empty. Otherwise, P starts at the root

r of T, followed by the preorder traversal of the subtree whose root is the first (i.e. smallest) child of r, followed by the preorder traversal of the subtree whose root is the second smallest child of r, and so on. The last (rightmost) element of P is referred to as the rightmost element of T as well.

Lemma 14. *The preorder traversal of a representative tree T is unique. Conversely, a representative tree T is uniquely determined by its preorder traversal.*

If we remove the first element of P, the remaining sequence is said to be the *root-free* preorder traversal of T.

Definition 15. *A canonical reducible permutation graph G is a self-labeling reducible graph such that the root-free preorder traversal of the representative tree of G is a canonical self-inverting permutation.*

Theorem 16. *A digraph G is a watermark instance produced by Chroni and Nikolopoulos's encoding algorithm [1] if, and only if, G is a canonical reducible permutation graph.*

Let T be the representative tree of some canonical reducible permutation graph G, and P a canonical self-inverting permutation corresponding to the root-free preorder traversal of T. We refer to the fixed element f of P also as the fixed element (or vertex) of both G and T. Similarly, the 2-cyclic elements of P correspond to *cyclic* elements (or vertices) of both G and T. The concepts we describe next will be employed in the characterization of canonical reducible permutation graphs.

A vertex $v \in V(T) \setminus \{2n+2\}$ is considered *large* when $n < v \leq 2n+1$; otherwise, $v \leq n$ and v is dubbed as *small.* Denote by X, Y, respectively, the subsets of large and small vertices in T, so $|X| = n+1$ and $|Y| = n$. By Lemma 13, $f \in X$. We then define $X_c = X \setminus \{f\} = \{x_1, \ldots, x_n\}$ as the set of large cyclic vertices in T.

We say that T is a *type*-1 tree—please see Figure 3(a)—when

(i) $n+1, n+2, \ldots, 2n+1$ are children of the root $2n+2$ in T; and
(ii) $1, 2, \ldots, n$ are children of $2n$.

Elseways, we say that T is a *type*-2 tree relative to f— please see Figure 3(b)— when

(i) $n+1 = x_1 < x_2 < \ldots < x_\ell = 2n+1$ are the children of $2n+2$, for some $\ell \in [2, n-1]$;
(ii) $x_i > x_{i+1}$ and x_i is the parent of x_{i+1}, for all $i \in [\ell, n-1]$;
(iii) $1, 2, \ldots, f-n-1$ are children of x_n;
(iv) $x_i = n+i$, for $1 \leq i \leq f-n-1$;
(v) f is a child of x_q, for some $q \in [\ell, n]$ satisfying $x_{q+1} < f$ whenever $q < n$; and
(vi) $N_T^*(f) = \{f-n, f-n+1, \ldots, n\}$ and $y_i \in N_T^*(f)$ has index $x_{y_i} - f + 1$ in the preorder traversal of $N_T^*[f]$.

Lemma 17. *If y_r is the rightmost vertex of a type-2 representative tree T relative to some $f \neq 2n+1$, then $y_r = \ell$.*

The following theorem characterizes canonical reducible permutation graphs.

Theorem 18. *A digraph G is a canonical reducible permutation graph if, and only if, G is a self-labeling reducible graph and*

(i) the fixed element of G is $2n+1$ and G has a type-1 representative tree; or
(ii) the fixed element of G belongs to $[n+2, 2n]$ and G has a type-2 representative tree.

4 Restoring a Damaged Watermark

In this section, we analyze the effects of a distortive attack against a canonical reducible permutation graph G whereby two edges $e_1, e_2 \in E(G)$ were removed. Let $G' = G \setminus \{e_1, e_2\}$.

4.1 Reconstructing the Hamiltonian Path

The algorithm given in pseudocode as Procedure 1 reconstructs the Hamiltonian path H of G, in case e_1 or e_2 belonged to H, and classifies each missing edge as either a path edge or a tree edge. The input is the damaged watermark graph G'. If $v \in V(G')$, we denote by $H(v)$ the subsequence of the Hamiltonian path of G that ends at v and starts as far as possible in G'. Also, we denote by $first(H(v))$ the first vertex of the subsequence $H(v)$.

The mechanics of Procedure 1 is that of sewing together the $k' \leq 3$ maximal directed paths resulting from the deletion of $k \leq 2$ edges from G. In short, each such directed path is reassembled by placing vertices, one by one in backwards fashion, starting at a vertex with out-degree 0 among those which have not yet been placed. The proof that it is always possible to restore the Hamiltonian path

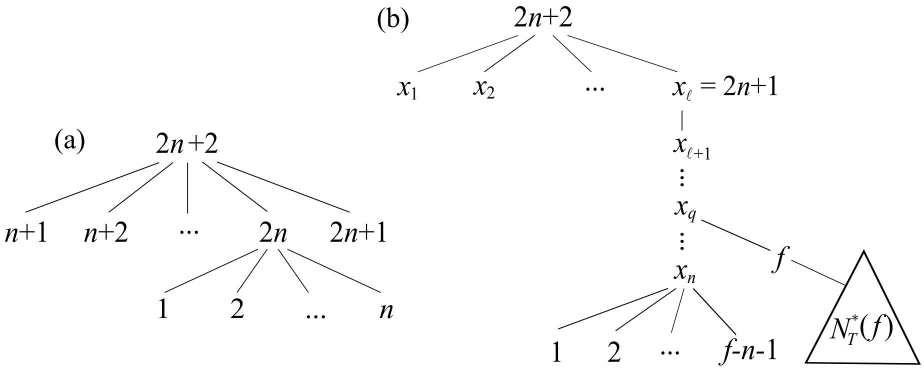


Fig. 3. (a) A type-1 representative tree. (b) A type-2 representative tree.

```
Procedure 1: Reconstructing the Hamiltonian path
$V_0 \leftarrow \{v \in V(G')$ s.t. $|N^+_{G'}| = 0\}$;   $V_1 \leftarrow \{v \in V(G')$ s.t. $|N^+_{G'}| = 1\}$
if $|V_0| = 1$ then
    let $v_0$ be the unique element in $V_0$
    if $|H(v_0)| = 2n+3$ then $H \leftarrow H(v_0)$, return $H$ and edge types as in Figure 4(a)
    else if $\exists\ v_1 \in V_1$ such that $|H(v_0)| + |H(v_1)| = 2n+3$ then
        $H \leftarrow H(v_1)||H(v_0)$, return $H$ and edge types as in Figures 4(b,c)
    else
        let $v_1, v_1' \in V_1$ be such that
            $|H(v_0)| + |H(v_1)| + |H(v_1')| = 2n+3$ and $N^+_{G'}(first(H(v_1)) \cap H(v_1') \neq \emptyset$
        $H \leftarrow H(v_1')||H(v_1)||H(v_0)$, return $H$ and edge types as in Figure 4(d)
else
    let $v_0, v_0'$ be the elements in $V_0$
    if $|H(v_0)| + |H(v_0')| = 2n+3$ then
        let $v_0$ be such that $N^+_{G'}(first(H(v_0))) \cap H(v_0') \neq \emptyset$
        $H \leftarrow H(v_0')||H(v_0)$, return $H$ and edge types as in Figures 4(e,f)
    else
        let $v_0' \in V_0$ and $v_1 \in V_1$ be such that $v_0' \in N^+_{G'}(first(H(v_1))$
        $H \leftarrow H(v_0')||H(v_1)||H(v_0)$, return $H$ and edge types as in Figures 4(g,h)
```

this way relies on the characterization of representative trees and is not overly complicated. It is, however, quite lengthy, since each possible case (see Figure 4) resulting from such an attack must be tackled separately. It has therefore been omitted.

As for the time complexity of the algorithm, note that, in general, $1 \leq |V_0| \leq 2$ and $1 \leq |V_1| \leq 3$. The latter follows from the definition of a self-labeling reducible graph and from the fact that two edges were removed from G. Moreover, each path $H(v_i)$ can be computed in $\mathcal{O}(|H(v_i)|)$ time. Consequently, the entire algorithm has complexity $\mathcal{O}(n)$.

4.2 Determining the Fixed Element

Suppose the watermark G has been attacked, which resulted in a damaged watermark G', where two unknown edges are missing. Now we shall recognize the fixed element of the original watermark, given the damaged one. Getting to know the fixed element of G will play a crucial role in retrieving the missing tree edges and consequently restoring the original key w.

The two following theorems, whose proofs we omit, characterize the fixed element f of G when $f = 2n+1$ and when $f < 2n+1$, respectively. Let T be the representative tree of the original watermark G. We consider the case where the two edges that have been removed belong to T. Denote by F the forest obtained from T by the removal of two edges.

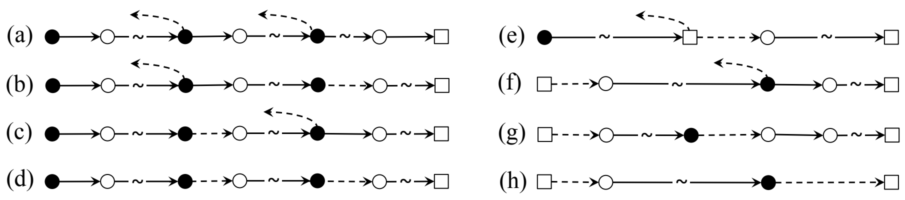


Fig. 4. Possible scenarios for the Hamiltonian path of a damaged watermark G. Dashed arrows indicate missing edges. Broken arrows (with a tilde in the middle) indicate paths of arbitrary size $d \geq 0$ (i.e., both ends may coincide). Squares, solid circles and hollow circles represent vertices whose out-degrees in G' are, respectively, 0, 1 and 2. The tree edges of G' are not being shown (unless those removed by the attacker, represented by backward arrows).

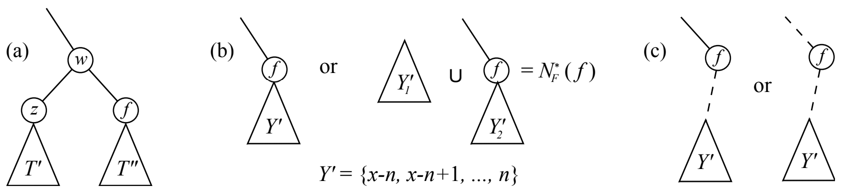


Fig. 5. (a–c) Conditions (i), (ii) and (iii) of Theorem 20, respectively

Theorem 19. *Let F be a forest obtained from the representative tree T by removing two edges, where $n > 2$. The fixed element of T is $f = 2n + 1$ if, and only if,*

(i) vertex $2n + 1$ is a leaf of F; and
(ii) the n small vertices of G' are children of $2n$ in F, with the possible exception of at most two of them, in which case they must be isolated vertices.

Theorem 20. *Let F be a forest obtained from the representative tree T of watermark G by removing two of its edges, and let $x \leq 2n$ be a large vertex of T which is not a child of $2n + 2$. Vertex x is the fixed element f of G if, and only if,*

(i) the large vertex x has a sibling z in F, and $x > z$; or
(ii) the subset of small vertices $Y' \subset Y$, $Y' = \{x - n, x - n + 1, \ldots, n\}$ can be partitioned into at most two subsets Y'_1, Y'_2, such that $\emptyset \neq Y'_1 = N^+_F(x)$ and Y'_2 is the vertex set of one of the trees which form F; or, whenever the previous conditions do not hold,
(iii) the large vertex x is the rightmost vertex of one of the trees of F, while the rightmost vertices of the remaining trees are all small vertices.

The above theorems lead to an algorithm that finds the fixed element of G in linear time. The input is the forest F, obtained from the representative tree T of

```
Procedure 2: Finding $f \neq 2n+1$
 1. If $F$ contains a large vertex $x$ having a sibling $z$
    then let $f \leftarrow max\{x, z\}$ and terminate the algorithm. Otherwise,
 2. For each large vertex $x$ of $F$ satisfying $N_F(x) \neq \emptyset$ and each small $y \in N_F(x)$,
    let $Y' = \{x-n, x-n+1, \ldots, n\}$. If $N_F^*(x) = Y'$ or $N_F^*(x) \subset Y'$,
    and $Y' \setminus N_F^+(x)$ is the vertex set of one of the trees of $F$,
    then let $f \leftarrow x$ and terminate the algorithm. Otherwise,
 3. Find the preorder traversals of the three trees of $F$, and
    then let $f$ be the unique vertex that is both large and the rightmost element
    of the preorder traversal of some tree of $F$.
```

G by the removal of two edges. The algorithm is as follows: first, check whether $f = 2n+1$. This is a direct task, applying Theorem 19: just verify if $2n+1$ is a leaf of F and whether all small vertices are children of $2n$, except possibly two, which must be isolated vertices. If both conditions are satisfied then set $f = 2n+1$ and terminate the algorithm. Otherwise, proceed to determining $f < 2n+1$ by checking the conditions described in Theorem 20 (see also Figure 5).

4.3 Determining the Root's Children

After having identified the fixed element of the watermark, we are almost in a position to determine the tree edges that have been removed.

Observe that, when $f = 2n+1$, the task is trivial, since, in this case, by Theorem 18, there can be only one canonical reducible permutation graph G relative to n. Such a graph is precisely the one with a type-1 representative tree T, which is unique for each $n > 2$ (cf. Property 3 of canonical reducible permutation graphs, in Section 2). By definition, the root-free preorder traversal of a type-1 representative tree, when $f = 2n+1$, is $n+1, n+2, \ldots, 2n, 1, 2, \ldots, n, 2n+1$.

We therefore want to determine the children of $2n+2$ restricted to the case where $f < 2n+1$. Let G be a watermark, T its representative tree and F the forest obtained from T by the removal of two edges. As usual, f stands for the fixed element of T, X is the set of large vertices other than $2n+2$, and $X_c = X \setminus \{f\}$. Finally, denote by $A \subseteq X_c$ the subset of ascending large cyclic vertices of T, which we shall refer to simply as the *ascending* vertices, and denote by D the set $D = X_c \setminus A$ of descending large cyclic vertices of T, or simply the *descending* vertices. Given the forest F and its fixed element f, Procedure 3 computes the set A, which, due to the representative tree properties of T, corresponds precisely to the children of its root $2n+2$. The notation still employs $N_F(v)$ and $N_F^*(v)$ for the children and the descendants of vertex v in F, respectively. We denote by $F[X_c]$ the subgraph of F induced by the vertices in X_c.

Theorem 21. *Procedure 3 correctly computes the set of ascending vertices of T in linear time.*

```
Procedure 3: Constructing the set of large ascending vertices
  1. If $F[X_c] \cup \{2n+2\}$ is connected then $A \leftarrow N_F(2n+2)$
     and terminate the algorithm. Otherwise,
  2. If $F[X_c] \cup \{2n+2\}$ contains no isolated vertices then $A \leftarrow N_F(2n+2) \cup \{2n+1\}$
     and terminate the algorithm. Otherwise,
  3. If $F[X_c] \cup \{2n+2\}$ contains two isolated vertices $x, x'$ then $A \leftarrow N_F(2n+2) \cup \{x, x'\}$
     and terminate the algorithm. Otherwise,
  4. If $F[X_c] \cup \{2n+2\}$ contains a unique isolated vertex $x$ then
                 if $|N_F^*(f)| = 2n - f + 1$ then
                         let $y_r$ be the rightmost vertex of $N_F^*(f)$
                         if $|N_F(2n+2)| < y_r$ then $A \leftarrow N_F(2n+2) \cup \{x, 2n+1\}$
                         else $A \leftarrow N_F(2n+2)$
                 else $A \leftarrow N_F(2n+2) \cup \{x\}$
```

Once we manage to recover the set A of children of the root $2n+2$ in T, the decoding algorithm proposed in the next section can be run, retrieving the encoded key ω. This suffices to prove that the original, undamaged watermark can be fully restored: a simple possibility is to run the (linear) encoding algorithm from scratch, with ω as input. We have, however, an even simpler algorithm that restores the watermark based on the reconstitution of its preorder traversal, but we shall not present it in the present paper due to space constraints.

5 A New Decoding Algorithm

We can now formulate our new decoding algorithm. If the input watermark presents $k \leq 2$ missing edges, the algorithm is able to fix it, prior to running the decoding step. The decoding step itself is absolutely straightforward, and relies on the following theorem.[7]

Theorem 22. *Let ω be a given key and G the watermark corresponding to ω. Let $A' = x_1, \ldots, x_{\ell-1}$ be the ascending sequence of children of $2n+2$, in the representative tree T of G, that are different from $2n+1$. Then $\omega = \sum_{x_i \in A'} 2^{2n-x_i}$.*

As a consequence of the above theorem, whenever the input watermark has *not* been tampered with, the proposed algorithm simply retrieves the encoded key in a faster, less complicated fashion than the original decoding algorithm from [1].[8]

Theorem 23. *Algorithm 4 retrieves the correct key $\omega \geq 4$, encoded in a watermark with up to two missing edges, in linear time.*

Corollary 24. *Distortive attacks in the form of k edge modifications (insertions/deletions) against canonical reducible permutation graphs G, with $|V(G)| = 2n+3$, $n > 2$, can be detected in polynomial time, if $k \leq 5$, and also recovered from, if $k \leq 2$. Such bounds are tight.*

[7] Though some reconstitution steps were written as if *exactly* two edges were missing, the case where a single edge is missing is at least as easy, since removing an arbitrary edge transforms the latter case into the former.

[8] Note that, in this case, it is straightforward to determine the set A, as $A = N_G^-(2n+2)$.

Algorithm 4: Obtaining the key from a possibly damaged watermark

```
1. Let $k \leftarrow |E(G)| - (4n + 3)$.
2. If $k > 2$, report the occurrence of $k$ edge removals and halt.
3. If $0 < k \leq 2$, proceed to the reconstitution (Procedures 1–3).
4. Calculate and return the key $\omega$ as indicated by Theorem 22.
```

6 Final Considerations

After characterizing the class of canonical reducible permutation graphs, we formulated a linear-time algorithm that restores a member of that class presenting up to two missing edges. Our results therefore have proved that canonical reducible permutation graphs are always able to detect and recover from malicious attacks in the form of $k \leq 2$ edge removals[9] in linear time. Moreover, we have shown that attacks in the form of $k \leq 5$ edge modifications (insertions/deletions) can be detected in polynomial time. Such level of resilience, we remark, is a very important feature of the original watermarking scheme.

Future directions. A necessary condition for a watermark G_1 to recover from the removal of a subset of edges $E'_1 \subset E(G_1)$, with $|E'_1| = k$, is that $G'_1 = G_1 \setminus E'_1$ is not isomorphic to some graph G'_2 obtained from watermark $G_2 \neq G_1$ by the removal of k edges. For $k \leq 2$, we have shown this condition is always satisfied, provided $n > 2$, and we have proved this is not always true for $k \geq 3$. An interesting open problem is therefore to characterize the set of keys $\Omega(k)$ whose corresponding watermarks can always recover from the removal of $k \geq 3$ edges.

Future research focusing on the development of watermarking schemes resilient to attacks of greater magnitude may consider extending the concept of canonical reducible permutation graphs by allowing permutations with h-cycles, with $h > 2$, as well as multiple fixed elements.

References

[1] Chroni, M., Nikolopoulos, S.D.: Efficient encoding of watermark numbers as reducible permutation graphs. arXiv:1110.1194v1 [cs.DS] (2011)

[2] Chroni, M., Nikolopoulos, S.D.: Encoding watermark numbers as cographs using self-inverting permutations. In: 12th Int'l Conference on Computer Systems and Technologies, CompSysTech 2011, vol. 578, pp. 142–148. ACM ICPS (2011) (Best Paper Award)

[3] Chroni, M., Nikolopoulos, S.D.: An efficient graph codec system for software watermarking. In: 36th IEEE Conference on Computers, Software, and Applications (COMPSAC 2012). IEEE Proceedings, pp. 595–600 (2012)

[9] The sole exceptions are two very small ($n = 2$) instances G_1, G_2 corresponding to keys $\omega_1 = 2$ (binary $B = 10$) and $\omega_2 = 3$ (binary $B = 11$), respectively, which become isomorphic to one another when edges $(1, 5), (4, 5)$ are removed from G_1 and edges $(1, 4), (4, 6)$ are removed from G_2.

[4] Chroni, M., Nikolopoulos, S.D.: Multiple encoding of a watermark number into reducible permutation graphs using cotrees. In: 13th Int'l Conference on Computer Systems and Technologies (CompSysTech 2012). ACM ICPS Proceedings, pp. 118–125 (2012)

[5] Chroni, M., Nikolopoulos, S.D.: An embedding graph-based model for software watermarking. In: 8th Int'l Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIH-MSP 2012. IEEE Proceedings, pp. 261–264 (2012)

[6] Collberg, C., Huntwork, A., Carter, E., Townsend, G., Stepp, M.: More on graph theoretic software watermarks: implementation, analysis and attacks. Information and Software Technology 51, 56–67 (2009)

[7] Collberg, C., Kobourov, S., Carter, E., Thomborson, C.: Error-correcting graphs for software watermarking. In: Bodlaender, H.L. (ed.) WG 2003. LNCS, vol. 2880, pp. 156–167. Springer, Heidelberg (2003)

[8] Collberg, C., Thomborson, C.: Software watermarking models and dynamic embeddings. In: Proc. 26th ACM SIGPLAN-SIGACT on Principles of Programming Languages, POPL 1999, pp. 311–324 (1999)

[9] Collberg, C., Thomborson, C., Townsend, G.M.: Dynamic graph-based software fingerprinting. ACM Transactions on Programming Languages and Systems 29, 1–67 (2007)

[10] Davidson, R.L., Myhrvold, N.: Method and system for generating and auditing a signature for a computer program, US Patent 5.559.884. Microsoft Corporation (1996)

[11] Hecht, M.S., Ullman, J.D.: Flow graph reducibility. SIAM J. Computing 1, 188–202 (1972)

[12] Hecht, M.S., Ullman, J.D.: Characterizations of reducible flow graphs. Journal of the ACM 21, 367–375 (1974)

[13] Zhu, J., Liu, Y., Yin, K.: A novel dynamic graph software watermark scheme. In: 1st Int'l Workshop on Education Technology and Computer Science, vol. 3, pp. 775–780 (2009)

[14] Venkatesan, R., Vazirani, V.V., Sinha, S.: A graph theoretic approach to software watermarking. In: Moskowitz, I.S. (ed.) IH 2001. LNCS, vol. 2137, pp. 157–168. Springer, Heidelberg (2001)

The Normal Graph Conjecture for Classes of Sparse Graphs

Anne Berry and Annegret Wagler

LIMOS UMR CNRS 6158[*]
Université Blaise Pascal (Clermont-Ferrand II)
Ensemble Scientifique des Cézeaux, 63 173 Aubière, France
{berry,wagler}@isima.fr

Abstract. Normal graphs are defined in terms of cross-intersecting set families: a graph is normal if it admits a clique cover $\mathcal{Q}$ and a stable set cover $\mathcal{S}$ s.t. every clique in $\mathcal{Q}$ intersects every stable set in $\mathcal{S}$.

Normal graphs can be considered as closure of perfect graphs by means of co-normal products and graph entropy. Perfect graphs have been recently characterized as those graphs without odd holes and odd antiholes as induced subgraphs (Strong Perfect Graph Theorem, Chudnovsky et al. 2006). Körner and de Simone observed that C_5, C_7, and $\overline{C}_7$ are minimal not normal and conjectured, in analogy to the Strong Perfect Graph Theorem, that every $(C_5, C_7, \overline{C}_7)$-free graph is normal (Normal Graph Conjecture, Körner and de Simone 1999).

We conclude from a result of Prömel and Steger that the Normal Graph Conjecture is asymptotically true and verify it for two classes of sparse graphs, 1-trees and cacti. In addition, we provide algorithms to recognize normal graphs within these two classes in linear time.

Keywords: Normal Graph Conjecture, 1-trees, cacti.

1 Introduction

Normal graphs come up in a natural way in an information theoretic context by studying co-normal products [5] or graph entropies [3]. A graph G is *normal* if G admits cross-intersecting clique and stable set covers, called a *valid pair* $(\mathcal{Q}, \mathcal{S})$: a clique cover $\mathcal{Q}$ and a stable set cover $\mathcal{S}$ s.t. every clique in $\mathcal{Q}$ intersects every stable set in $\mathcal{S}$. Figure 1 presents three normal graphs and their valid pairs.

The class of normal graphs includes the well-studied perfect graphs, and the interest in normal graphs is caused by the fact that they form, in different ways, a weaker variant of perfect graphs. Berge [1] introduced *perfect graphs* as those graphs G, where the clique number $\omega(G')$ equals the chromatic number $\chi(G')$ for each induced subgraph $G' \subseteq G$ ($\omega(G)$ denotes the size of a maximum clique in G, $\chi(G)$ the least number of stable sets covering all nodes of G).

[*] Research partially supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010.

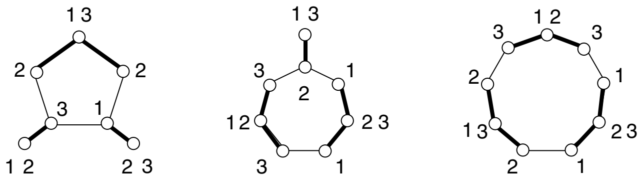


Fig. 1. Three normal graphs and their valid pairs: bold edges indicate the clique covers, labels the cross-intersecting stable set covers

Berge observed that chordless odd cycles C_{2k+1} with $k \geq 2$, the *odd holes*, and their complements, the *odd antiholes* $\overline{C}_{2k+1}$ with $k \geq 2$, are graphs G with $\omega(G) < \chi(G)$, see Figure 2.

This motivated Berge's Strong Perfect Graph Conjecture: a graph G is perfect if and only if G is odd hole- and odd antihole-free. In a sequence of remarkable results, Chudnovsky et al. [2] finally turned the above conjecture into the Strong Perfect Graph Theorem.

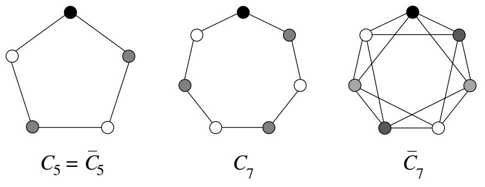


Fig. 2. Small odd holes and odd antiholes

Since normal graphs are a closure of perfect graphs in terms of graph entropy [3,6,12] and taking co-normal products [4,5], Körner and de Simone [7] asked for a similarity of the two classes in terms of forbidden subgraphs. Körner [4] showed that an odd hole C_{2k+1} is normal if and only if $k \geq 4$. By the invariance of normality under taking complements, an odd antihole $\overline{C}_{2k+1}$ is also normal if and only if $k \geq 4$. C_5, C_7 and $\overline{C}_7$ are *minimally* not normal since all of their proper induced subgraphs are perfect and, hence, normal. Körner and de Simone conjectured that there are no other minimally not normal graphs:

Conjecture 1 (Normal Graph Conjecture [7]). All graphs without a C_5, C_7, and $\overline{C}_7$ as induced subgraph are normal.

The Normal Graph Conjecture would imply a sufficient condition for normality which could be checked in polynomial time. However, the non-existence of C_5, C_7, and $\overline{C}_7$ in a graph is not necessary to be normal (see the first two graphs in Figure 1), and a characterization of normal graphs by forbidden subgraphs is not possible (see [14]). So far, the Normal Graph Conjecture has been verified for triangle-free graphs [7], webs [13] and line graphs [10].

In order to treat the Strong Perfect Graph Conjecture from a probabilistic point of view, Prömel and Steger [9] asked for the relation of perfect and odd hole, odd antihole-free graphs with the same number of nodes. For that, they proved that almost all C_5-free graphs are perfect.

This verified the Strong Perfect Graph Conjecture asymptotically as every odd hole, odd antihole-free graph is in particular C_5-free. Since every $(C_5, C_7, \overline{C}_7)$-free graph is C_5-free, also almost all $(C_5, C_7, \overline{C}_7)$-free graphs are perfect and, therefore, normal. Consequently, we obtain:

Corollary 1. *The Normal Graph Conjecture is asymptotically true.*

Moreover, it is known from [9] that a random graph is with high probability perfect only if it is very sparse (i.e., if it has less edges than nodes) or, due to the invariance of perfectness under taking complements, very dense. This motivates our study of two classes of sparse graphs w.r.t. normality.

We start with a class of graphs having as many edges as nodes: A *1-tree* is a connected graph $G = (V, E)$ with $|V| = |E|$ (obtained from a tree by adding one edge since trees are precisely the connected graphs with $|V| - 1 = |E|$). Hence, G contains exactly one cycle C (due to this property, 1-trees are also called *unicyclic graphs*). In other words, G can be obtained from the cycle C and certain trees by a sequence of node-identifications.

In Section 3, we characterize the normal 1-trees with the help of two results: a connected triangle-free graph is normal if and only if it has a so-called nice edge cover [7] and the identification of two normal graphs in one node yields a normal graph again. The latter is a consequence from results on clique identification and normality in [14], see Section 2. We obtain that the only not normal 1-trees are either equal to a C_7 or contain a C_5. This verifies the Normal Graph Conjecture for the class of 1-trees (note: no $\overline{C}_7$ can occur in a 1-tree, so it suffices to show that (C_5, C_7)-free 1-trees are normal) and leads to a linear time recognition algorithm for normal 1-trees.

We further extend this result to the larger class of cacti in Section 4. A *cactus* is a connected graph whose cycles are all edge-disjoint. Thus, a cactus $G = (V, E)$ with k cycles can be considered as a graph obtained from a tree by adding k edges in a certain way (thus cacti admit $|V| - 1 + k$ edges) or, alternatively, as a graph obtained from k 1-trees by a sequence of node-identifications.

As the class of normal graphs is closed under node identification (see Section 2), a cactus is normal if it can be obtained by identifying *normal* 1-trees in nodes; this verifies the Normal Graph Conjecture for cacti, as in particular all (C_5, C_7)-free cacti are normal.

As there also exist normal cacti containing a C_5 or a C_7, our further goal is to recognize normal cacti. For that, we design a linear time algorithm that decomposes a cactus into 1-trees, determines the normality status of each 1-tree, and decides whether the recomposed graph is normal or not.

We close with some concluding remarks and discuss consequences and some future lines of research.

2 Clique Identification and Normal Graphs

A graph G arises by *identification* of two disjoint graphs G_1 and G_2 in a clique if there are cliques $Q_1 \subseteq G_1$ and $Q_2 \subseteq G_2$ with $|Q_1| = |Q_2|$ and a bijection $\phi : Q_1 \to Q_2$ identifying every node $v \in Q_1$ with $\phi(v) \in Q_2$, see Figure 3.

In this section, we discuss how normal graphs can be obtained by clique identification. The first two graphs in Figure 1 are examples of normal graphs, constructed by identifying a *non*-normal C_5 and C_7 with two edges and one edge, respectively. Figure 3 shows another normal graph obtained by identifying two *non*-normal graphs in an edge.

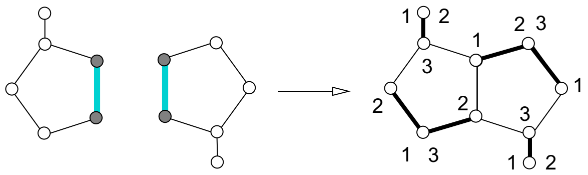


Fig. 3. Identifying two non-normal graphs in an edge: bold edges indicate the clique cover, labels the cross-intersecting stable set cover of the resulting normal graph

However, the non-normal building blocks in these examples are not too far from being normal which shows in particular that it obviously suffices if the nodes in the common clique are covered in one of the two building blocks.

This suggests to relax the condition of normality for the building blocks as follows. Let G be a graph such that G has a stable set cover $\mathcal{S}$, $G - Q'$ has a clique cover $\mathcal{Q}_{Q'}$, for some clique Q', and $\mathcal{S}$ and $\mathcal{Q}_{Q'}$ are cross-intersecting. We call such a graph G *nearly normal*, $(\mathcal{Q}_{Q'}, \mathcal{S})$ a *nearly valid pair* of G, and Q' an *unnormal clique* of G.

Figure 4 shows examples of nearly normal graphs together with their nearly valid pairs and unnormal cliques.

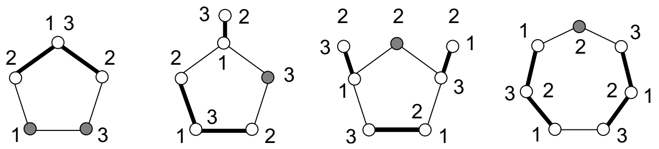


Fig. 4. Nearly normal graphs: the gray-shaded nodes induce the unnormal cliques Q', bold edges indicate the clique covers $\mathcal{Q}_{Q'}$, labels the cross-intersecting stable set covers

It was shown in [14] that this weaker form of normality suffices for constructing a normal graph by clique identification, provided the involved stable set covers are suitable.

Lemma 1. [14] *Construct G by identifying two nearly normal graphs G_1 and G_2 in a clique Q^* and let $Q_1, Q_2 \subseteq Q^*$ be disjoint unnormal cliques. The resulting graph G is normal if there exist nearly valid pairs $(\mathcal{Q}_{Q_i}(G_i), \mathcal{S}(G_i))$ for $i = 1, 2$ satisfying at least one of the following conditions.*

(1) *$\mathcal{S}(G_i)$ contains a stable set S with $S \cap Q^* = \emptyset$ for $i = 1, 2$;*
(2) *$\mathcal{S}(G_i)$ contains no stable set S with $S \cap Q^* = \emptyset$ for $i = 1, 2$;*
(3) *$\mathcal{S}(G_1)$ contains a stable set S with $S \cap Q^* = \emptyset$ but $\mathcal{S}(G_2)$ does not, and Q_1 is non-empty (or vice versa).*

Lemma 1 provides sufficient conditions to obtain a normal graph via clique identification, but it is not yet clear whether these conditions are also *necessary.* However, condition (1) of Lemma 1 is certainly satisfied if Q^* is a non-maximal clique in G_i for $i = 1, 2$: In order to cover a common neighbor v of all nodes in Q^*, $\mathcal{S}(G_i)$ has to contain a stable set S with $v \in S$ and, thus, $S \cap Q^* = \emptyset$.

Corollary 2. *The class of normal graphs is closed under identification in non-maximal cliques.*

In particular, this is the case if Q^* consists in a single (but not isolated) node only. Thus, the class of normal graphs is closed under node-identification.

As we have seen that we can construct normal graphs by identifying two unnormal ones in a clique (as in Figure 3), a natural question is whether there exist further ways to construct normal graphs by clique identification. The next lemma from [14] gives an answer showing that the normality or near-normality of the building blocks is required if the resulting graph is supposed to be normal.

Lemma 2. [14] *Let G be a normal graph obtained by identifying two graphs G_1 and G_2 in a clique Q^*.*

(1) *If G admits a valid pair $(\mathcal{Q}, \mathcal{S})$ such that $\mathcal{S}$ contains no stable set avoiding Q^*, then G_1 and G_2 are normal;*
(2) *If G admits a valid pair $(\mathcal{Q}, \mathcal{S})$ such that $\mathcal{S}$ contains a stable set avoiding Q^*, then G_1 and G_2 are nearly normal with unnormal cliques Q_1^* and Q_2^* such that $Q_i^* \subseteq Q^*$ and $Q_1^* \cap Q_2^* = \emptyset$.*

However, we cannot obtain normal graphs by identifying two non-normal graphs in a node (as the unnormal cliques $Q_1^*, Q_2^* \subseteq Q^*$ cannot be disjoint if $|Q^*| = 1$). We call a nearly normal graph *almost normal* if its unnormal clique consists in one node only. Except the C_5 (whose only possible unnormal clique has size two), all the graphs in Figure 4 are examples of almost normal graphs. We finally obtain the following result which is crucial for our purpose:

Theorem 1. *Construct a graph G by identifying two graphs G_1 and G_2 in a node q^*, G is normal if and only if at least one of the following conditions holds:*

(1) *G_1 and G_2 are normal;*
(2) *G_1 is normal and G_2 almost normal with unnormal node q^*, or vice versa.*

3 The Normal 1-Trees

In this section we characterize the normal 1-trees, verify the Normal Graph Conjecture for such graphs, and provide a linear time recognition algorithm for normal 1-trees.

Recall that a 1-tree G is a connected graph containing exactly one cycle C, i.e., G can be obtained from the cycle C and certain trees by a sequence of node-identifications. In order to characterize the normal 1-trees we use Theorem 1 and the result that a connected triangle-free graph is normal if and only if it has a so-called nice edge cover [7], as defined below.

Let $G = (V, E)$ be a graph and $\mathcal{F}$ be a minimal edge cover of G, i.e., an inclusion-wise minimal set $\mathcal{F} \subseteq E$ s.t. every node in V is the endnode of some edge in $\mathcal{F}$. Consider a (not necessarily chordless) odd cycle C in G and the distribution of the edges of $\mathcal{F}$ alongside the cycle C. We say that a node v of C is *even* w.r.t. $\mathcal{F}$ if v is the endnode of either none or two edges in $\mathcal{F} \cap E(C)$. Since C is an odd cycle, C has an odd number of even nodes. An edge cover of a graph G is called *nice* if it is minimal and every odd cycle in G has at least three even nodes.

For example, the bold edges of all three graphs in Figure 1 form nice edge covers, whereas the graphs in Figure 4 do not admit any nice edge cover.

Körner and de Simone showed in [7] that the occurrence of nice edge covers is sufficient for a graph to be normal. In particular, they characterized the triangle-free normal graphs as follows:

Theorem 2. [7] A connected triangle-free graph is normal if and only if it has a nice edge cover.

Let $G_1 +_v G_2$ denote the graph obtained from G_1 and G_2 by identification in the node v. For instance, the second graph in Figure 4 equals $C_5 +_v K_2$, obtained from a C_5 and a K_2 by node identification; the third graph in Figure 4 equals $(C_5 +_v K_2) +_{v'} K_2$, obtained from $C_5 +_v K_2$ and K_2 by identification in another node of the C_5.

We characterize the normal 1-trees with the help of Theorem 1 and Theorem 2 as follows.

Theorem 3. *A 1-tree G is* not *normal if and only if one of the following conditions holds.*

(1) $G = C_5$.
(2) $G = C_5 +_v T$ *where T is a tree.*
(3) $G = (C_5 +_v T) +_{v'} T'$ *where T, T' are trees and v, v' are two non-consecutive nodes of the C_5.*
(4) $G = C_7$.

That the non-normal 1-trees are all not $(C_5, C_7, \overline{C}_7)$-free implies:

Corollary 3. *The Normal Graph Conjecture is true for 1-trees.*

As there are obviously normal 1-trees admitting a cycle of length 5 or 7, we next address the question of recognizing normal 1-trees.

To find the cycle in a 1-tree $G = (V, E)$ as first step, we will use Algorithm LexBFS [11][1]. LexBFS is a specialized Breadth-First Search, which numbers the nodes from $n = |V|$ to 1; each node x obtains a label, denoted by $\text{lex}(x)$, which is the decreasing list of the neighbors of x with a higher number than that of x. At each step, a node of lexicographically largest label is chosen to be numbered.

When the input graph has only one cycle, all the nodes obtain a label of size 1, except the node which 'closes' a cycle, whose label is of size 2. To find the corresponding cycle C, we can use the double label, which yields the next 2 nodes on C; we then go on to take the lowest numbered node between these two: its label yields the next node on C, and so on, until we find a node already in C, which will be the 'root' of C in the tree induced by the LexBFS numbering. The algorithm can stop as soon as more than 7 nodes of the cycle have been found, as the graph is certain to be normal.

Algorithm 1 (Test for normality of a 1-tree)
Input: 1-tree $G = (V, E)$;
Output: decision whether or not G is normal.

```
(1) Graph traversal.
    Choose a root node $r \in V$.
    Starting in $r$, apply LexBFS until a node $x$ receives a double label $\text{lex}(x) = \{y, z\}$.
(2) Normality test.
    Go from $y$ and $z$ 4 steps back towards $r$ (passing from a node to its "father"):
    IF no cycle of length 5 or 7 has been found, output "$G$ is normal" and STOP.
    ELSE test the conditions of Theorem 3 and decide normality accordingly.
```

Theorem 4. *Given a 1-tree $G = (V, E)$, Algorithm 1 decides in $O(|V|)$ whether or not G is normal.*

4 The Normal Cacti

In this section we study cacti, i.e., graphs with edge-disjoint cycles. Since a cactus G with k cycles can be considered as a graph obtained from a tree by adding k edges, we can alternatively obtain G from k 1-trees by a sequence of node-identifications. As we have characterizations of both normal graphs obtained by node-identification (Theorem 1) and normal 1-trees (Theorem 3) from the previous sections, it is natural to decompose a cactus G accordingly: we have to choose a set of articulation points in G s.t. each building block contains exactly one cycle. We denote the building block of cycle C by $B(C)$.

[1] Although a simple DFS with a 'father' function could be used with the same time complexity, LexBFS provides us an elegant way to directly determine the studied cycle as well as its nodes.

If all these blocks $B(C)$ of G are normal, then G is clearly normal due to Theorem 1. This holds particularly if none of the cycles C in G has length 5 or 7 by Theorem 3; hence we have:

Corollary 4. *The Normal Graph Conjecture is true for cacti.*

As there are obviously normal cacti admitting a cycle of length 5 or 7, our further goal is to find a way to decide whether or not a cactus is normal.

For that, we introduce the block-tree of a cactus as follows. For a cactus $G = (V, E)$ with k cycles, we call a set $A \subseteq V$ of articulation points *valid* if the graph obtained from G by removing A has k components and the resulting building blocks $B(C)$, i.e., the components together with the respective articulation points, contain exactly one cycle C each.

Hence, using a valid set of articulation points decomposes a cactus in as many 1-trees as it has cycles. We obtain the *block-tree* $T(G, A) = (A \cup \mathcal{B}, L)$ of G by taking as nodes a valid set A of articulation points and the set $\mathcal{B}$ of the resulting building blocks $B(C)$ of G and joining two nodes of $T(G, A)$ if and only if one corresponds to an articulation point $q \in A$ and the other one to a block $B(C)$ with $q \in B(C)$.

By construction, $T(G, A)$ is a tree as it is bipartite and has no cycle. We call a leaf of $T(G, A)$ an *endblock* of G and say that two blocks of G are *adjacent* if they share an articulation point in A. Figure 5 shows a cactus G and a block-tree $T(G, A)$ (the black nodes of G are used as articulation points).

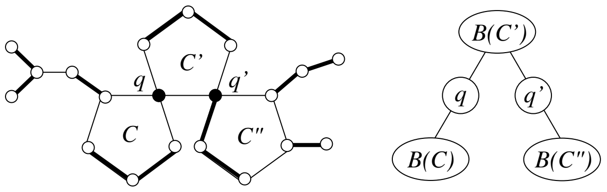


Fig. 5. A cactus and a block-tree

The main idea is to design an algorithm that, given a cactus G,

– finds A and constructs the block-tree $T(G, A)$,
– decides the normality status of each block $B(C)$,
– shrinks $T(G, A)$ iteratively by deleting an endblock $B(C)$ and deciding how normal the remaining graph is.

We next explain in detail a procedure to construct A and the block-tree $T(G, A)$, how to decide the normality status of each block $B(C)$, and of the remaining graph after deleting an endblock.

Constructing a block-tree. To construct a block tree, we will again use algorithm LexBFS. As is the case for Algorithm 1, all the nodes obtain a label of size 1, except the nodes which close a cycle C. Each time a new cycle is found, a new block is started, and the block tree is updated accordingly (by inserting the block and its root node as articulation point).

Algorithm 2 (Constructing a block-tree)
Input: a cactus $G = (V, E)$*;*
Output: a block-tree $T(G, A) = (A \cup \mathcal{B}, L)$*.*

```
(1) Choose a root $r \in V$.
(2) Apply LexBFS until a node $x$ receives a double label $\text{lex}(x) = \{y, z\}$;
    Define the corresponding cycle $C$ and its root $r_c$;
    Start a new block $B(C)$ which contains $C$;
    Update $T(G, A)$ accordingly: add $r_c$ to $A$, $B(C)$ to $\mathcal{B}$, update $L$;
(3) Repeat (2) until all nodes are processed.
```

Theorem 5. *Given a cactus* $G = (V, E)$*, Algorithm 2 constructs a block-tree of* G *in* $O(|V|)$*.*

Deciding the normality status of blocks. Since each block $B(C)$ is a 1-tree by construction, Theorem 3 characterizes whether it is normal or not. In view of Theorem 1, we have to distinguish the different not normal blocks: it is important to determine whether a not normal block $B(C)$ is almost normal and to specify its unnormal nodes.

We denote the set of possible unnormal nodes of an almost normal block $B(C)$ by $U(C)$ and notice that all of them lie on C: Any valid minimal clique cover of a cactus G uses edges and triangles only, maintaining the notion of even nodes for odd cycles C with length ≥ 5. An unnormal node q of $B(C)$ is not covered by the clique cover $\mathcal{Q}_q$ of a nearly valid pair $(\mathcal{Q}_q, \mathcal{S})$ of $B(C)$, but it has to be covered by a clique outside $B(C)$ to become an even node of C.

As an immediate consequence of Theorem 3, we obtain:

Proposition 1. *A block* $B(C)$ *with cycle* C *is*

(1) *not (almost) normal if* $B(C) = C_5$*;*
(2) *almost normal with* $U(C) = C$ *if* $B(C) = C_7$*;*
(3) *almost normal with* $U(C) = N(v) \cap C$ *if* $C = C_5$ *and* $B(C) = C_5 +_v T$ *where* T *is a tree;*
(4) *almost normal with* $U(C) = C \setminus \{v, v'\}$ *if* $C = C_5$ *and* $B(C) = (C_5 +_v T) +_{v'} T'$ *where* T, T' *are trees and* v, v' *are two non-consecutive nodes of the* C_5*;*
(5) *normal otherwise.*

Shrinking a cactus by removing endblocks. To iteratively shrink (the block-tree of) a cactus G by removing an endblock $B(C)$, we use the following idea based on Theorem 1: If $B(C)$ does not satisfy any of the theorem's conditions, then

G is not normal. Otherwise, G is normal if and only if the graph is normal that results from G by either removing $B(C)$ or collapsing a normal endblock $B(C)$ into a K_2 (i.e., into a smaller normal graph).

Consider a cactus G and a block-tree $T(G, A)$. For an endblock $B(C)$ of $T(G, A)$ and (one of) its adjacent block(s) $B(C')$ with common node $q \in A$, we denote by $G - B(C)$ (resp. $G - B(C) + e$) the graph obtained by removing $B(C)$ maintaining q (resp. by replacing $B(C)$ by an edge e attached to q).

As an immediate consequence of Theorem 1 combined with Proposition 1, we infer the normality of G after removing $B(C)$:

Corollary 5. *Let G be a cactus with block-tree $T(G, A)$, $B(C)$ an endblock and $B(C')$ an adjacent block with common node $q \in A$.*

(1) *If $B(C)$ and $B(C')$ are normal, then G is normal if and only if $G - B(C)$ is normal.*
(2) *If $B(C)$ is normal and $B(C')$ not, then G is normal if and only if $G - B(C) + e$ is normal.*
(3) *If $B(C)$ is almost normal and $q \in U(C)$, then G is normal if and only if $G - B(C)$ is normal.*
(4) *If $B(C)$ is almost normal and $q \notin U(C)$ or if $B(C) = C_5$, then G is not normal.*

The above corollary enables us to shrink a cactus starting from endblocks, keeping normality of the input graph or deciding that G is not normal. Thereby, the cases (1) and (3) allow us to simply remove $B(C)$, (4) provides sufficient conditions that G is not normal, and (2) allows us to maintain or change the normality status of the remaining block $B(C') + e = K_2 +_q B(C')$ as follows:

Lemma 3. *Let G be a cactus with block-tree $T(G, A)$, $B(C)$ a* normal *endblock and $B(C')$ an adjacent block with common node $q \in A$.*

(1) *If $B(C')$ is almost normal and $q \in U(C')$, then $B(C') + e$ is normal in $G - B(C) + e$.*
(2) *If $B(C')$ is almost normal and $q \notin U(C')$, then $B(C') + e$ remains almost normal with $U(C')$ in $G - B(C) + e$.*
(3) *If $B(C') = C_5$, then $B(C') + e$ is almost normal with $U(C') = N(q) \cap C'$ in $G - B(C) + e$.*

For the algorithmic process, however, it suffices to shrink the block-tree of a cactus by removing an endblock $B(C)$ and, if necessary, updating the normality status of $B(C')$ according to Lemma 3.

Example 1. Reconsider the cactus G and its block-tree $T(G, A)$ depicted in Figure 5. Initially, we have the following normality status for its blocks:

– $B(C)$ is almost normal with $q \in U(C)$ (Proposition 1(3)),
– $B(C') = C_5$ (Proposition 1(1)),
– $B(C'')$ is normal (Proposition 1(5)).

Algorithm 3 (Test for normality of a cactus)
Input: a cactus G;
Output: decision whether or not G is normal.

```
(1) Constructing a block-tree.
    Construct a set $A$ and the block-tree $T(G,A) = (A \cup \mathcal{B}, L)$ by Algorithm 2,
    test the normality status of each block $B(C) \in \mathcal{B}$ by Proposition 1.
(2) Test for sufficient conditions.
    IF all blocks of $T(G,A)$ are normal THEN output "$G$ is normal" and STOP.
    IF all blocks of $T(G,A)$ are not normal OR an endblock of $T(G,A)$ satisfies Corollary 5 (4) THEN output "$G$ is not normal" and STOP.
(3) Shrinking $T(G,A)$. Choose an endblock $B(C)$ of $T(G,A)$ and an adjacent block $B(C')$ with common node $q \in A$.
      • IF $B(C)$ is normal and $B(C')$ is almost normal with $q \in U(C')$, THEN update $B(C')$ as normal.
      • IF $B(C)$ is normal and $B(C') = C_5$, THEN update $B(C')$ as almost normal with $U(C') = N(q) \cap C'$.
    Now shrink $T(G,A)$ as follows:
      • remove $B(C)$ from $\mathcal{B}$ and the edge $B(C)q$ from $L$;
      • IF now $q$ has degree 1 THEN remove $q$ from $A$.
    Continue with step (1).
```

We next apply step (2) of Algorithm 3 and notice that none of the sufficient conditions for G being (not) normal is satisfied. Hence, we proceed with step (3) and select one of the two endblocks $B(C)$ and $B(C'')$ of $T(G,A)$.

If $B(C)$ is selected, then $B(C')$ is its only adjacent block with common node $q \in A$. None of the two conditions to update the normality status of $B(C')$ is satisfied, so we only remove $B(C)$ and q from $T(G,A)$ in step (3). The subsequent test for sufficient conditions in step (2) reveals that $B(C') = C_5$ is now an endblock, hence the decision is "not normal" according to Corollary 5 (4).

On the other hand, if $B(C'')$ is selected, then $B(C')$ is its only adjacent block with common node $q' \in A$. The second condition is satisfied, hence $B(C')$ is updated as almost normal with $U(C') = N(q') \cap C'$; $B(C'')$ and q' are removed from $T(G,A)$. The test for sufficient conditions in step (2) reveals that now all blocks are not normal, hence the decision is "not normal" due to Theorem 1.

In both cases, the algorithm finds the correct answer "not normal".

Theorem 6. *Given a cactus $G = (V,E)$, Algorithm 3 decides in $O(|V|)$ whether or not G is normal.*

5 Concluding Remarks

In this work, we verify the Normal Graph Conjecture asymptotically as well as for two classes of sparse graphs, 1-trees and cacti. Since the class of normal graphs is closed under complementation, we also conclude:

Corollary 6. *The Normal Graph Conjecture is true for complements of 1-trees or cacti.*

Moreover, we solve the problem of deciding whether such a graph is normal even when it does contain a C_5 or a C_7.

It is well-known that a random graph $G_{n,m}$ with n nodes and $m = (1 - \varepsilon)\frac{n}{2}$ edges consists of many small 1-trees (of order $\log(n)$) as components. Theorem 3 shows that most of such graphs are normal whereas only half of them are perfect. Thus, Theorem 3 indicates that there are many more normal than perfect random graphs $G_{n,m}$ with edge density $m = (1-\varepsilon)\frac{n}{2}$. In addition, we can expect that there are many more normal than (C_5, C_7)-free sparse graphs.

Furthermore, it would be interesting to generalize our techniques and results to larger graph classes. Canonical candidates are superclasses of cacti, e.g., chordless graphs (whose cycles are all chordless) or outerplanar graphs (who admit an embedding into the plane such that no edges cross and all nodes lie on the outer face). The question is to find suitable decompositions for such graphs into blocks such that the normality status of both the blocks and the recomposed graphs can be determined.

References

1. Berge, C.: Färbungen von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind. Wiss. Zeitschrift der Martin-Luther-Universität Halle-Wittenberg 10, 114–115 (1961)
2. Chudnovsky, M., Robertson, N., Seymour, P., Thomas, R.: The Strong Perfect Graph Theorem. Annals of Mathematics 164, 51–229 (2006)
3. Cziszár, I., Körner, J., Lovász, L., Marton, K., Simonyi, G.: Entropy splitting for antiblocking corners and perfect graphs. Combinatorica 10, 27–40 (1990)
4. Körner, J.: An Extension of the Class of Perfect Graphs. Studia Math. Hung. 8, 405–409 (1973)
5. Körner, J., Longo, G.: Two-step encoding of finite memoryless sources. IEEE Trans. Inform. Theory 19, 778–782 (1973)
6. Körner, J., Marton, K.: Graphs that split entropies. SIAM J. Discrete Math. 1, 71–79 (1988)
7. Körner, J., de Simone, C.: On the Odd Cycles of Normal Graphs. Discrete Appl. Math. 94, 161–169 (1999)
8. Lovász, L.: Normal Hypergraphs and the Weak Perfect Graph Conjecture. Discrete Math. 2, 253–267 (1972)
9. Prömel, H.J., Steger, A.: Almost all Berge graphs are perfect. Combinatorics, Probability, and Computing 1, 53–79 (1992)
10. Schülzke, H.O.: The Normal Graph Conjecture for line graphs. Diplomarbeit, TU Berlin (2006)
11. Rose, D.J., Tarjan, R.E., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. 5, 266–283 (1976)
12. Simonyi, G.: Perfect Graphs and Graph Entropy: An Updated Survey. In: Ramirez-Alfonsin, J.L. (ed.) Perfect Graphs, pp. 293–328. Wiley (2001)
13. Wagler, A.K.: The Normal Graph Conjecture is true for circulants. In: Bondy, A., et al. (eds.) Graph Theory in Paris. Trends in Mathematics, pp. 365–374. Birkhäuser, Basel (2007)
14. Wagler, A.K.: Constructions for Normal Graphs and Some Consequences. Discrete Applied Mathematics 156, 3329–3338 (2008)

On the Parameterized Complexity of Computing Graph Bisections

René van Bevern[1], Andreas Emil Feldmann[2], Manuel Sorge[1], and Ondřej Suchý[3]

[1] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany[⋆]
{rene.vanbevern,manuel.sorge}@tu-berlin.de
[2] Combinatorics & Optimization, University of Waterloo, Canada
andreas.feldmann@uwaterloo.ca
[3] Faculty of Information Technology, Czech Technical University in Prague, Czech Republic[⋆⋆]
ondrej.suchy@fit.cvut.cz

Abstract. The Bisection problem asks for a partition of the vertices of a graph into two equally sized sets, while minimizing the *cut size*. This is the number of edges connecting the two vertex sets. Bisection has been thoroughly studied in the past. However, only few results have been published that consider the parameterized complexity of this problem.

We show that Bisection is FPT w.r.t. the minimum cut size if there is an optimum bisection that cuts into a given constant number of connected components. Our algorithm applies to the more general Balanced Biseparator problem where vertices need to be removed instead of edges. We prove that this problem is W[1]-hard w.r.t. the minimum cut size and the number of cut out components.

For Bisection we further show that no polynomial-size kernels exist for the cut size parameter. In fact, we show this for all parameters that are polynomial in the input size and that do not increase when taking disjoint unions of graphs. We prove fixed-parameter tractability for the distance to constant cliquewidth if we are given the deletion set. This implies fixed-parameter algorithms for some well-studied parameters such as cluster vertex deletion number and feedback vertex set.

1 Introduction

We consider the NP-hard Bisection problem for which the n vertices of a graph $G = (V, E)$ need to be partitioned into two sets A and B of size at most $\lceil n/2 \rceil$ each ((A, B) is a *bisection* of G). At the same time the *cut size* needs to be minimized. This is the number of edges connecting vertices in A with vertices in B. Throughout this paper it will be convenient to consider the decision problem corresponding to Bisection, which is defined as follows.

[⋆] Supported by DFG project DAPA, NI 369/12.

[⋆⋆] The main work was done while O. Suchý was at TU Berlin, supported by the DFG, research project AREG, NI 369/9.

BISECTION
Input: A graph G and a positive integer k.
Question: Does G have a bisection with cut size at most k?

BISECTION is of importance both in theory and practice, and has applications in divide-and-conquer algorithms [26], computer vision [25], and route planning [12]. We study BISECTION from the point of view of parameterized complexity, and consider several parameters (Table 1) that naturally arise from the known results for BISECTION. That is, we consider a given parameter p of an input instance and ask whether an algorithm with running time $f(p) \cdot n^{O(1)}$ exists that optimally solves the problem. Here $f(p)$ is a function that only depends on p. If there is such an algorithm we say that the problem is *fixed-parameter tractable* (or FPT for short) with respect to p.

BISECTION has been thoroughly studied in the past. It is known that it is NP-hard in general [20] and the minimum cut size can be approximated within a factor of $O(\log n)$ [32]. Assuming the Unique Games Conjecture, no constant factor approximations exist [23]. For special graph classes such as trees [27] and solid grids [17] the optimum cut size can be computed in polynomial time. For planar graphs it is still open whether BISECTION is NP-hard, but it is known to be FPT with respect to the cut size [8].

In this paper we show that for general graphs one can find an optimal bisection in FPT-time with respect to the cut size if there is an optimal bisection that cuts the graph into a given constant number of connected components. This result is motivated by the fact that in practice the solutions are typically cut into very few connected components [2]. Also for random regular graphs the sets A and B of the optimum bisection are connected with high probability [9]. Our algorithm is presented for the more general BALANCED BISEPARATOR problem, in which vertices instead of edges need to be removed in order to bisect the graph. To achieve our result, we generalize the *treewidth reduction* technique for separation problems that has been recently introduced by Marx et al. [29]. By adapting it to the global balancedness constraint, we address an open question by Marx et al. [30] of whether this is possible. We furthermore observe that BALANCED BISEPARATOR is W[1]-hard with respect to the cut size and the number of cut out components. Hence, BALANCED BISEPARATOR is unlikely to be FPT even when combining these parameters. This means that to obtain a fixed-parameter algorithm it is unavoidable to impose some additional constraint. We chose our condition on having a constant number of cut out connected components in the optimum solution as a natural candidate, as argued above.

Whether BISECTION is FPT with respect to the cut size alone, though, remains open. However, we show that no polynomial-size problem kernels exist for this parameter, unless coNP $\subseteq$ NP/poly. Hence, it is unlikely that there is a polynomial-time algorithm that computes an instance of size polynomial in the cut size and equivalent to the original instance. We prove this by giving a corresponding result for all parameters that are polynomial in the input size and that do not increase when taking the disjoint union of graphs. This includes parameters such as treewidth, minimum cut size, cliquewidth, and more generally bandwidth (see [5, Theorem 44]).

Table 1. Overview of known and new parameterized results

Parameter	Results for Bisection
cut size	FPT for planar graphs [8] FPT for constant cut out components (Theorem 2) No poly-size kernel (Theorem 3) W[1]-hard for Balanced Biseparator (deferred)
treewidth	FPT [33, 34] No poly-size kernel (Theorem 3)
cliquewidth	XP (Lemma 3) No poly-size kernel (Theorem 3)
bandwidth	No poly-size kernel (Theorem 3)
feedback vertex set	FPT (Corollary 1)
cluster vertex deletion number	FPT (Corollary 1)

Some of these parameters have been considered for the Bisection problem before. For instance, we already mentioned the cut size, and it was shown that the problem is FPT with respect to treewidth [33, 34]. However, although treewidth is probably the most widely used graph parameter for sparse graphs, it is not suitable for dense graphs, although they can also have simple structure. For that purpose, Courcelle and Olariu [11] introduced the parameter *cliquewidth* [14]. For this parameter we present an XP algorithm, i.e. an algorithm finding the optimum solution in time $n^{O(q)}$ if a cliquewidth-q expression for the graph is given. In fact we obtain an algorithm that shows that Bisection is FPT with respect to the *cliquewidth-q vertex deletion number*:[1] the number of vertices that has to be deleted to obtain a graph of constant cliquewidth q. To the best of our knowledge this parameter has not been considered in the past. The cliquewidth-q deletion number is a generalization of several well-studied graph parameters like vertex cover ($q = 1$) [10], cluster vertex deletion number and cograph vertex deletion number ($q = 2$) [11], or feedback vertex set ($q = 3$) [24] and treewidth-t vertex deletion set ($q = 2^{t+1} + 1$) [11, 18].

In this paper we use standard terminology of graph theory [13]. Due to space constraints, many proofs are deferred to the full version of the paper.

2 An FPT Algorithm for Cut Size and Constant Number of Cut Out Components

This section shows that Bisection is FPT with respect to the cut size if there is an optimum bisection that cuts into at most some given constant number of connected components. To this end, we show an FPT-algorithm for the more general problem Balanced Biseparator; to formally define it, we need some terminology. Let G be a graph and $S \subseteq V(G)$. We call S an *A-B-separator* if there are vertex sets $A, B \subseteq V(G)$ such that S, A, B form a partition of $V(G)$, and there are no edges between A and B in G. Moreover, we call S *balanced* if $||A|-|B|| \leq 1$.

[1] Precisely, we need the vertex deletion set to be given to obtain FPT for this parameter.

We say that G has a *balanced separator* S if there are sets A, B such that S is a balanced A-B-separator for G. We say that S is an s-t-separator for vertices s, t if there are vertex sets $A, B \subseteq V(G)$ such that S is an A-B-separator and $s \in A$ and $t \in B$. We say that an s-t-separator S is *inclusion-wise minimal*, or just *minimal*, if there is no s-t-separator $S' \subsetneq S$. Finally, we say that S is a c-component separator for G, if there are c connected components in $G - S$. Balanced Biseparator is the following problem:

Balanced Biseparator
Input: A graph G and a positive integer k.
Question: Does G have a balanced separator of size at most k?

Using a reduction from the W[1]-hard problem Cutting ℓ Vertices [28], one can show that Balanced Biseparator is W[1]-hard with respect to k and the number c of cut out components. Hence, an additional constraint like c being constant is unavoidable to get an FPT-algorithm. Our algorithm for Balanced Biseparator transfers also to Bisection:

Proposition 1. *There is a polynomial-time many-one reduction from* Bisection *to* Balanced Biseparator *such that the desired separator size is at most one larger than the desired cut size. Furthermore, each bisection with c connected components for the* Bisection *instance yields a balanced separator for the* Balanced Biseparator *instance whose removal leaves at most* $c + 2$ *connected components and vice-versa.*

Proposition 1 implies that Balanced Biseparator is a more general problem than Bisection. We now outline an FPT algorithm for Balanced Biseparator: we first observe that a balanced separator consists of minimal s-t-separators between a collection of "terminal" vertices s, t. The terminal vertices are chosen one from each of the connected components of the input graph without the separator. Guessing the terminals, we can reduce Balanced Biseparator to finding an "almost balanced" separator consisting of vertices contained in minimal separators. To find such a separator, we generalize the "treewidth reduction" technique introduced by Marx et al. [29, 30] to graphs with vertex weights. We obtain an algorithm that constructs a weighted graph G' that preserves all inclusion-wise minimal vertex cuts of size at most k between some given terminals, preserves the weight of the cut out parts, and has treewidth bounded by some function $g(k, c)$ where c is the number of terminals. Moreover the algorithm runs in time $f(k, c) \cdot n^{O(1)}$. We then show that Balanced Biseparator is fixed-parameter tractable with respect to treewidth when fixing the number of components of the separated graph. The final algorithm guesses the terminals, reduces the treewidth and then solves the bounded-treewidth problem.

The main ingredient in our FPT algorithm for Balanced Biseparator is a generalization of the treewidth reduction technique of Marx et al. [30] to graphs with vertex weights: we aim to construct a graph of bounded treewidth that preserves all inclusion-wise minimal s-t-separators of a given size. To this end, we define trimmers. Let $G = (V, E)$ be a graph, k an integer and $T \subseteq V$. A tuple (G^*, ϕ) of a graph $G^* = (V^*, E^*)$ and a total, surjective, but not necessarily injective mapping $\phi\colon V \to V^*$ is called a (k,T)*-trimmer* of G if the following holds. (Here, we extend $\phi(V) := \bigcup_{v\in V}\phi(v)$ and $\phi^{-1}(v) := \{v' \mid \phi(v') = v\}$.)

(i) Let $S \subseteq V^*$. A set $C \subseteq V^*$ is a connected component in $G^* - S$ if and only if $\phi^{-1}(C)$ is a connected component in $G - \phi^{-1}(S)$ (i. e., ϕ implies a one-to-one mapping between the connected components of $G^* - S$ and $G - \phi^{-1}(S)$).

(ii) If S is an inclusion-wise minimal s-t-separator for G with $|S| \le k$ and $s,t \in T$, then $\phi(S) = S$ and $\phi(S)$ is an inclusion-wise minimal $\phi(s)$-$\phi(t)$-separator for G^*.

We obtain the following.

Theorem 1. *Let $G = (V,E)$ be a graph. For every constant $k \in \mathbb{N}$ and constant-size $T \subseteq V$, we can compute a (k,T)-trimmer (G^*,ϕ) for G in $O(|V|+|E|)$ time such that the treewidth of G^* is at most $g(k,|T|)$ for some function g depending only on k and $|T|$. Moreover, both ϕ and ϕ^{-1} are linear-time computable with respect to their output length.*

We can now state an algorithm for finding a c-component balanced separator of a given size. To this end, we first note that Balanced Biseparator is FPT with respect to treewidth and, thus, gathering the final ingredient for the algorithm.

Lemma 1. *Let G be a graph with treewidth ω and integer vertex-weights λ. Let Λ be the sum of all vertex weights and let $c \ge 2$ be an integer. We can find in $\omega^{O(\omega)} \cdot c^2 \cdot \Lambda^2 \cdot n$ time, for all integers $1 \le s \le \Lambda$, a minimum weight c-component A-B-separator S with $\lambda(A) = s$, or reveal that no such separator exists.*

We now arrive at the main theorem of this section.

Theorem 2. *Let G be a graph. Given non-negative integers c and k, in $h(c,k) \cdot n^{c+3}$ time we can find a c-component balanced separator for G of size at most k if it exists. Here, $h(c,k)$ is a function depending only on c and k.*

Proof. The algorithm proceeds as follows. For each $T \subseteq V(G)$ of size c we compute a (k,T)-trimmer (G^*,ϕ) using Theorem 1. We create a vertex weight function λ for G^* by letting $\lambda(v) = |\phi^{-1}(v)|$. Then, for each s, $|V(G)|/2-1-k \le s \le |V(G)|/2+k$, we compute a minimum-weight c-component A'-B'-separator for G^* with $\lambda(A') = s$ using Lemma 1. If among these separators there is an A'-B'-separator S' with $|\lambda(A') - \lambda(B')| \le k - \lambda(S') + 1$, then we compute $S := \phi^{-1}(S'), A := \phi^{-1}(A')$, and $B := \phi^{-1}(B')$. Note that, by trimmer property (i), S is a c-component A-B-separator for G. Moreover, since ϕ is a total mapping, $||A|-|B|| \le k-|S|+1$. We move $k-|S|$ vertices from A or B to S such that S is a c-component balanced separator for G and we output S. Note that, unless $|V(G)|$ is bounded by a function of k and the problem is trivial, moving the vertices to S without changing the number of components is always possible, because not every vertex of a connected component can separate it into multiple ones and there is always a component of size at least two. If no suitable separator is found, we output that there is no c-component balanced separator of size at most k for G.

Let S be a c-component balanced separator of size k for G and pick vertices $v_1, \dots, v_c$, one from each connected component of $G - S$. Let us observe that the above algorithm finds a c-component balanced separator of size at most k. Note that S is a v_i-v_j-separator for each $1 \leq i < j \leq c$. Hence, S contains inclusion-wise minimal v_i-v_j-separators $S_{i,j}$ of size at most k. Let $\hat{S} = \bigcup_{1 \leq i < j \leq c} S_{i,j}$, call a connected component in $G - \hat{S}$ *odd* if it does not contain any v_i, and let $\tilde{S}$ be the union of $\hat{S}$ and all odd components. Note that odd components are contained in S. Hence, $\tilde{S}$ is a c-component $\tilde{A}$-$\tilde{B}$-separator for G with $||\tilde{A}| - |\tilde{B}|| \leq k - |\tilde{S}| + 1$ and $|V(G)|/2 - 1 - k \leq |\tilde{A}| \leq |V(G)|/2 + k$. By trimmer property (ii) we have that $\phi(\hat{S}) = \hat{S}$ is contained in G^*. Thus, by trimmer property (i), ϕ implies a one-to-one mapping of connected components C in $G - \hat{S}$ and their counterparts $\phi(C)$ in $G^* - \hat{S}$. In particular, there is such a mapping for all odd connected components. Thus, $\phi(\tilde{S})$ is a c-component $\phi(\tilde{A})$-$\phi(\tilde{B})$-separator for G^* and we have $\lambda(\phi(\tilde{S})) = |\tilde{S}|, \lambda(\phi(\tilde{A})) = |\tilde{A}|$, and $\lambda(\phi(\tilde{S})) = |\tilde{S}|$. Hence, an A'-B'-separator S' for G^* with $\lambda(S') \leq \lambda(\phi(\tilde{S}))$ and $\lambda(A') = \lambda(\phi(\tilde{A}))$ is enumerated by the algorithm of Lemma 1. Applying the size bounds of $\tilde{A}, \tilde{B}, \tilde{S}$ we have $||\lambda(A')| - |\lambda(B')|| \leq k - |\lambda(S')| + 1$ and $|V(G)|/2 - 1 - k \leq |\lambda(A')| \leq |V(G)|/2 + k$. Thus, the algorithm described above finds a c-component balanced separator of size at most k for G. The proof of the running time bound is deferred to a full version of the paper. □

3 Incompressibility

Problem kernelization is a powerful preprocessing tool in attacking NP-hard problems [6, 21]. A *reduction to a problem kernel* is an algorithm that, given an instance I with parameter p of a parameterized problem, in time polynomial in $(|I| + p)$ outputs an instance I' of the same problem and a parameter p' such that

i) I is a yes-instance if and only if I' is a yes-instance,
ii) $|I'| + p' \leq f(p)$, where f is a function only depending on p.

The function f is called the *size* of the problem kernel. It is desirable to find problem kernels of size polynomial in the parameter p.

In this section, we show that Bisection has no polynomial-size kernel with respect to any parameter that is polynomial in the input size and does not increase when taking disjoint unions of graphs. Our result excludes polynomial-size problem kernels for the parameters treewidth, cut size of the bisection (the "standard parameter"), cliquewidth, and more generally pathwidth (see [5, Theorem 44]).

Theorem 3. *Unless coNP ⊆ NP/poly,* Bisection *does not have polynomial-size kernels with respect to any parameter that is polynomial in the input size and that does not increase when taking disjoint unions of graphs.*

For Theorem 3, we first show that a version of Bisection with integer edge weights does not have a polynomial-size kernel, and then show how to remove the weights. To prove that Edge-Weighted Bisection does not have a polynomial-size kernel, it is sufficient to show a cross composition (cf. Bodlaender et al. [7]) from the NP-hard [19] Maximum Cut problem to Edge-Weighted Bisection.

Maximum Cut
Input: A graph $G = (V, E)$ and an integer k.
Question: Is there a partition of V into sets A and B such that at least k edges have one endpoint in A and one in B?

Lemma 2. *There is a cross composition of* Maximum Cut *to* Edge-Weighted Bisection *with respect to any parameter that is polynomial in the input size and does not increase when taking the disjoint union of graphs.*

Showing the cross composition amounts to the following. We give a polynomial-time algorithm that transforms input instances $(G_1, k_1), \ldots, (G_t, k_t)$ of Maximum Cut into one instance (G^*, k^*) of Edge-Weighted Bisection such that (G^*, k^*) is a yes-instance if and only if one of the Maximum Cut instances is and such that k^* is polynomial in the size of the largest input instance.

Construction 1. The construction resembles the reduction given for the NP-hardness of Bisection by Garey et al. [20]. To easier present the construction, without loss of generality we assume that

i) each of the G_i, $1 \leq i \leq t$, has exactly n vertices and $k_1 = \cdots = k_t =: k$ (cf. Bodlaender et al. [7]),
ii) $1 \leq k \leq n^2$: if $k = 0$, all instances are yes-instances, and if $k > n^2$, all instances are no-instances. Hence, if not $1 \leq k \leq n^2$, we can return a trivial yes-instance or no-instance of Edge-Weighted Bisection,
iii) t is odd: otherwise, we can add a no-instance to the list of input instances that consists of the edgeless graph on n vertices.

Since the output graph G^* will consist of connected components, each having at most $2n$ vertices, and since our parameter is polynomial in the input size and does not increase when taking the disjoint union of graphs, we trivially obtain that the output parameter is polynomial in n. We create G^* as follows: for each input graph $G_i = (V_i, E_i)$, $1 \leq i \leq t$, add to G^* the vertices in V_i and a clique V_i' with $|V_i|$ vertices and edges of weight $W := n^2$ each. All vertices in V_i' are adjacent to all vertices in V_i in G^* via an edge of weight W. Now, for each pair $v, w \in V_i$, add an edge $\{v, w\}$ to G^* with weight W if $\{v, w\} \notin E_i$ and with weight $W - 1$ if $\{v, w\} \in E_i$. We set $k^* := Wn^2 - k$.

We use Construction 1 to show Lemma 2 and subsequently Theorem 3.

4 FPT for the Cliquewidth-q Vertex Deletion Number

This section shows Bisection to be fixed-parameter tractable with respect to the number of vertices that have to be removed from a graph to reduce its cliquewidth to a constant q. Thus, we generalize many well-studied graph parameters like vertex cover ($q = 1$) [10], cluster vertex deletion number and cograph vertex deletion number ($q = 2$) [11], or feedback vertex set ($q = 3$) [24] and treewidth-t vertex deletion set [18]. Our definition of cliquewidth is inspired by Hliněný et al. [22].

Let q be a positive integer. We call (G, λ) a *q-labeled graph* if G is a graph and $\lambda : V(G) \to \{1, 2, \ldots, q\}$ is a mapping. The number $\lambda(v)$ is called *label* of a vertex v. We introduce the following operations on labeled graphs:

(1) For every i in $\{1,\dots,q\}$, we let $\bullet_i$ denote the graph with only one vertex that is labeled by i (a constant operation).
(2) For every pair of distinct $i,j \in \{1,2,\dots,q\}$, we define a unary operator $\eta_{i,j}$ such that $\eta_{i,j}(G,\lambda) = (G',\lambda)$, where $V(G') = V(G)$, and $E(G') = E(G) \cup \{(v,w) \mid v,w \in V, \lambda(v) = i, \lambda(w) = j\}$. In other words, the operator adds all edges between label-i vertices and label-j vertices.
(3) For every pair of distinct $i,j \in \{1,2,\dots,q\}$, we let $\rho_{i\to j}$ be the unary operator such that $\rho_{i\to j}(G,\lambda) = (G,\lambda')$, where $\lambda'(v) = j$ if $\lambda(v) = i$, and $\lambda'(v) = \lambda(v)$ otherwise. The operator only changes the labels of vertices labeled i to j.
(4) Finally, $\oplus$ is a binary operation that makes the disjoint union, while keeping the labels of the vertices unchanged. Note explicitly that the union is disjoint in the sense that $(G,\lambda) \oplus (G,\lambda)$ has twice the number of vertices of G.

A *q-expression* is a well-formed expression φ written with these symbols. The q-labeled graph produced by performing these operations therefore has a vertex for each occurrence of the constant symbol in φ; and this q-labeled graph (and any q-labeled graph isomorphic to it) is called the *value* $val(\varphi)$ of φ. If a q-expression φ has value (G,λ), we say that φ is a *q-expression of G*. The *clique-width* of a graph G, denoted by $cwd(G)$, is the minimum q such that there is a q-expression of G. We say that a join $\eta_{i,j}$ is *full* if there is no edge between vertices of label i and j in the labeled graph on which the join is applied.

Proposition 2. *For any q-expression for an n-vertex graph there is an equivalent one which is at most as long as φ, contains $O(q^2 \cdot n)$ symbols, and for which every join is full.*

In the following, we show how to compute an optimal bisection using the q-expression of a given graph G. This will naturally also solve the decision problem Bisection. Let $D \subseteq V(G)$ and φ be a q-expression for $G \setminus D$, i.e. $val(\varphi) = (G \setminus D, \lambda)$. Let A_0, B_0 be a partition of D. For now, we assume that there are no edges between A_0 and B_0. Let $n_i(\varphi)$ for $i \in \{1,\dots,q\}$ be the number of vertices of $G \setminus D$ with label i. For every pair of vectors $\boldsymbol{a} = (a_1,\dots,a_q), \boldsymbol{b} = (b_1,\dots,b_q) \in \mathbb{N}^q$ with $a_i + b_i = n_i(\varphi)$, let us denote by $Cut_{A_0,B_0}(\varphi,\boldsymbol{a},\boldsymbol{b})$ the minimum number of edges between different parts of a partition (A,B) of $V(G)$ which satisfies the following conditions: (i) $A_0 \subseteq A$, $B_0 \subseteq B$, (ii) the number of vertices in $A \setminus D$ and $B \setminus D$ of label i are a_i and b_i, respectively. In the following we use x_i to denote the i'th entry in a vector $\boldsymbol{x}$.

Lemma 3. *There is an algorithm that for given G, A_0, B_0 and φ in time $O(n^{2q} \cdot q \cdot |\varphi|)$ computes all the numbers $Cut_{A_0,B_0}(\varphi,\boldsymbol{a},\boldsymbol{b})$.*

Proof. We prove the lemma by induction on the length of the q-expression. By Proposition 2 we can assume that every join in φ is full. If $\varphi = \bullet_i$, then we have $n_i(\varphi) = 1$ and $n_j(\varphi) = 0$ for every $j \neq i$. Hence, in each pair of q-dimensional vectors $\boldsymbol{a},\boldsymbol{b}$ there is either $a_i = 1$ or $b_i = 1$ and the other numbers are zero. In this case, there is exactly one partition fulfilling the conditions (i) and (ii), namely the one which puts the only vertex of $G \setminus D$ to set A or B as required. It is easy to compute the number of edges between the parts in this partition.

Now, suppose $\varphi = \eta_{i,j}(\varphi')$. Since φ' is shorter than φ, by the induction hypothesis we can use an algorithm for φ' to store all the results in a table $Cut_{A_0,B_0}(\varphi', \boldsymbol{a}, \boldsymbol{b})$. Note that $val(\varphi')$ differs from $G \setminus D$ only in that $G \setminus D$ has an edge between every vertex of label i and every vertex of label j, while $val(\varphi')$ has no such edges (as the join is full). Therefore, every partition (A, B) of $G \setminus D$ fulfilling the conditions (i) and (ii), is also a partition for $val(\varphi')$ fulfilling these conditions, but in $G \setminus D$ there are exactly $a_i \cdot b_j + a_j \cdot b_i$ more edges between the parts. Hence, we can output $Cut_{A_0,B_0}(\varphi, \boldsymbol{a}, \boldsymbol{b}) = Cut_{A_0,B_0}(\varphi', \boldsymbol{a}, \boldsymbol{b}) + a_i \cdot b_j + a_j \cdot b_i$.

Next, let us assume that $\varphi = \rho_{i \to j}(\varphi')$, and the table containing the values of $Cut_{A_0,B_0}(\varphi', \boldsymbol{a}', \boldsymbol{b}')$ is already computed. Note that in $G \setminus D$ there are no vertices of label i, so we have $0 = n_i(\varphi) = a_i = b_i$. On the other hand, some of the vertices which have label j in $G \setminus D$ had label i in $val(\varphi')$. A minimal partition for $G \setminus D$, $\boldsymbol{a}$, and $\boldsymbol{b}$ which satisfies the conditions (i) and (ii) is also a partition for $val(\varphi')$ which satisfies the conditions (i) and (ii) for some $\boldsymbol{a}', \boldsymbol{b}'$, but we don't know the distributions of a_j to a'_j and a'_i and of b_j to b'_j and b'_i. Therefore $Cut_{A_0,B_0}(\varphi, \boldsymbol{a}, \boldsymbol{b})$ can be computed as $\min\{Cut_{A_0,B_0}(\varphi', \boldsymbol{a}', \boldsymbol{b}')\}$ where the minimum is taken over all pairs $\boldsymbol{a}', \boldsymbol{b}'$ where $a'_t = a_t$ and $b'_t = b_t$ for every $t \in \{1, \ldots, q\} \setminus \{i, j\}$; $a_j = a'_j + a'_i$; $b_j = b'_j + b'_i$ and $a'_t + b'_t = n_t(\varphi')$ for $t \in \{i, j\}$. As every pair $\boldsymbol{a}', \boldsymbol{b}'$ gives rise to exactly one $\boldsymbol{a}, \boldsymbol{b}$, all the minima can be computed in one pass over all $\boldsymbol{a}', \boldsymbol{b}'$.

Finally, let $\varphi = \varphi^1 \oplus \varphi^2$ and let the values of $Cut_{A_0,B_0}(\varphi^1, \boldsymbol{a}^1, \boldsymbol{b}^1)$ and $Cut_{A_0,B_0}(\varphi^2, \boldsymbol{a}^2, \boldsymbol{b}^2)$ be already computed and stored in a table. A minimal partition for $G \setminus D$ and $\boldsymbol{a}, \boldsymbol{b}$ satisfying the conditions (i) and (ii) also induces partitions for $val(\varphi^1)$ and $val(\varphi^2)$, which satisfy the conditions (i) and (ii) for some $\boldsymbol{a}^1, \boldsymbol{b}^1$ and $\boldsymbol{a}^2, \boldsymbol{b}^2$, but we don't know the distributions of a_i to a^1_i and a^2_i and of b_i to b^1_i and b^2_i. Moreover, there are no edges between $val(\varphi^1)$ and $val(\varphi^2)$. Thus $\min\{Cut_{A_0,B_0}(\varphi^1, \boldsymbol{a}^1, \boldsymbol{b}^1) + Cut_{A_0,B_0}(\varphi^2, \boldsymbol{a}^2, \boldsymbol{b}^2)\}$ gives $Cut_{A_0,B_0}(\varphi, \boldsymbol{a}, \boldsymbol{b})$, where the minimum is taken over all $\boldsymbol{a}^1, \boldsymbol{b}^1$ and $\boldsymbol{a}^2, \boldsymbol{b}^2$ where for every $i \in \{1, \ldots, q\}$, $a_i = a^1_i + a^2_i$, $b_i = b^1_i + b^2_i$, and $a^l_i + b^l_i = n_i(\varphi^l)$ for $l \in \{1, 2\}$. As every pair of pairs $\boldsymbol{a}^1, \boldsymbol{b}^1$ and $\boldsymbol{a}^2, \boldsymbol{b}^2$ gives rise to exactly one pair $\boldsymbol{a}, \boldsymbol{b}$, all the minima can be computed in one pass over all combinations of $\boldsymbol{a}^1, \boldsymbol{b}^1$ and $\boldsymbol{a}^2, \boldsymbol{b}^2$.

Concerning the running time, we again argue by induction to show that the overall time is $O(n^{2q} \cdot q \cdot |\varphi|)$. If $\varphi = \bullet_i$, then $|\varphi| = 1$ and the computation of Cut_{A_0,B_0} for the only possible pair of q-dimensional vectors takes $O(m + n) \subseteq O(n^{2q} \cdot q)$ time. This constitutes the induction basis. Otherwise, for any sub-expression φ' of a given expression φ, the computation of the table for φ' takes $O(n^{2q} \cdot q \cdot |\varphi'|)$ time by the induction hypothesis. Observe that there are $O(n^q)$ different pairs of q-dimensional vectors $\boldsymbol{a}, \boldsymbol{b}$ with $a_i + b_i = n_i(\varphi)$. If $\varphi = \eta_{i,j}(\varphi')$, then the computation for each pair of vectors takes $O(q)$ time. For $\varphi = \rho_{i \to j}(\varphi')$, one pass through the table of φ' is obviously accomplished in $O(n^q)$ time, spending $O(1)$ time per entry. Since in both cases $|\varphi| = |\varphi'| + 1$, this proves the time bound for φ for these expressions. Finally, if $\varphi = \varphi^1 \oplus \varphi^2$ then the tables for φ^1 and φ^2 can be computed in $O(n^{2q} \cdot q \cdot (|\varphi^1| + |\varphi^2|))$ time. Then we cycle over the entries of both tables and for each combination we spend $O(q)$ time, so this can be accomplished in $O(n^{2q} \cdot q)$ time. Since $|\varphi| = |\varphi^1| + |\varphi^2| + 1$, also in this case the algorithm runs in $O(n^{2q} \cdot q \cdot |\varphi|)$ time. □

Theorem 4. *For G a graph, $D \subseteq V(G)$, and φ a q-expression for $G \setminus D$ there is an $O(2^{|D|} \cdot n^{2q+1}q^3)$ time algorithm which computes the optimal bisection of G.*

Proof. It is enough to find the minimum of $Cut_{A_0,B_0}(\varphi, \boldsymbol{a}, \boldsymbol{b})$ over all partitions A_0, B_0 and pairs of q-dimensional vectors $\boldsymbol{a}, \boldsymbol{b}$ with $|A_0| + \sum_{i=1}^{q} a_i$ equal to $|B_0| + \sum_{i=1}^{q} b_i$. Since Lemma 3 only applies when there are no edges between A_0 and B_0, we delete them and add the number of them to the sum. As the size of φ is $O(q^2 \cdot n)$ by Proposition 2, the running time follows from Lemma 3. □

Given D, an $f(q)$-expression for $G \setminus D$ can be computed in polynomial time using a cliquewidth approximation [31]. Thus, Bisection is FPT with respect to the size of any constant-cliquewidth vertex-deletion set that is obtainable in FPT time.

Corollary 1. Bisection *is fixed-parameter tractable with respect to the size of a feedback vertex set, the size of a cluster vertex deletion set, and the size of a treewidth-t vertex deletion set.*

It is easy to generalize Theorem 4 to Balanced d-Partitioning, where one searches for a partition into some constant $d > 2$ many equal-sized parts. The running time achieved is $O(d^{|D|+1} \cdot n^{2(d-1)q+1}q^3)$. We note that such a running time bound is tight in the sense that there is no algorithm with running time $f(d, |D|)n^{O(1)}$ for constant q unless FPT = W[1]: since the deletion of a feedback vertex set leaves a forest, the resulting graph has clique-width at most 3 [11]. Thus, if there was an algorithm with the above running time, then Balanced Partitioning would be fixed-parameter tractable with respect to the combined parameter size of a minimum feedback vertex set and number of parts in the partition. However, we can show that this parameter combination yields a W[1]-hard problem.

5 Conclusion

A natural generalization of the Bisection problem is to partition the graph into d equally-sized sets, for some arbitrary d instead of only two. This problem is called Balanced Partitioning and is considerably harder than Bisection. For instance Balanced Partitioning is hard to approximate even on trees [15]. Nonetheless it is of great importance in applications such as parallel computing [3] and VLSI circuit design [4]. Due to the hardness results [15] it was asked whether the problem is FPT for parameters resulting in algorithms useful in practice. Many of the known results already rule out FPT algorithms for some parameters such as treewidth or cluster vertex deletion number (Balanced Partitioning is NP-hard for trees [16] and graphs formed by a disjoint union of cliques [1]). We addressed this question and were able to show that the problem is W[1]-hard for the *combined* parameter cut size, feedback vertex set, treewidth, and number d of partitions.[5] We can, however, show that Balanced Partitioning is FPT with respect to the vertex cover number.[2]

[2] These results are deferred to a full version.

The main open problem remaining from this paper is the status of the parameterized complexity of Bisection with respect to the parameter cut size alone. But also, for the Balanced Partitioning problem, the question posed by Feldmann [15] of whether practical algorithms beyond the standard deterministic worst-case scenario exist, remains unanswered.

Acknowledgments. We thank Bart M. P. Jansen, Stefan Kratsch, Rolf Niedermeier and the anonymous referees for helpful suggestions. We are furthermore grateful to Rolf Niedermeier for many interesting discussions.

References

[1] Andreev, K., Räcke, H.: Balanced graph partitioning. Theory of Computing Systems 39(6), 929–939 (2006)

[2] Arbenz, P.: Personal communication, ETH Zürich (2013)

[3] P. Arbenz, G. van Lenthe, U. Mennel, R. Müller, and M. Sala. Multi-level μ-finite element analysis for human bone structures. In *Proc. 8th PARA*, volume 4699 of *LNCS*, pages 240–250. Springer, 2007.

[4] Bhatt, S.N., Leighton, F.T.: A framework for solving VLSI graph layout problems. J. Comput. Syst. Sci. 28(2), 300–343 (1984)

[5] Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theor. Comput. Science 209(1-2), 1–45 (1998)

[6] Bodlaender, H.L.: Kernelization: New upper and lower bound techniques. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 17–37. Springer, Heidelberg (2009)

[7] Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Cross-composition: A new technique for kernelization lower bounds. In: Proc. 28th STACS. LIPIcs, vol. 9, pp. 165–176. Dagstuhl (2011)

[8] Bui, T.N., Peck, A.: Partitioning planar graphs. SIAM J. Comput. 21(2), 203–215 (1992)

[9] Bui, T.N., Chaudhuri, S., Leighton, F.T., Sipser, M.: Graph bisection algorithms with good average case behavior. Combinatorica 7(2), 171–191 (1987)

[10] Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theor. Comput. Sci. 411(40-42), 3736–3756 (2010)

[11] Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. Discrete Appl. Math. 101(1-3), 77–114 (2000)

[12] Delling, D., Goldberg, A.V., Pajor, T., Werneck, R.F.F.: Customizable route planning. In: Pardalos, P.M., Rebennack, S. (eds.) SEA 2011. LNCS, vol. 6630, pp. 376–387. Springer, Heidelberg (2011)

[13] Diestel, R.: Graph Theory, 4th edn. Graduate Texts in Mathematics, vol. 173. Springer (2010)

[14] Espelage, W., Gurski, F., Wanke, E.: How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In: Brandstädt, A., Le, V.B. (eds.) WG 2001. LNCS, vol. 2204, pp. 117–128. Springer, Heidelberg (2001)

[15] Feldmann, A.E.: Fast balanced partitioning is hard, even on grids and trees. Theor. Comput. Sci. 485, 61–68 (2013)

[16] Feldmann, A.E., Foschini, L.: Balanced partitions of trees and applications. In: Proc. 29th STACS. LIPIcs, vol. 14, pp. 100–111. Dagstuhl (2012)

[17] Feldmann, A.E., Widmayer, P.: An $\mathcal{O}(n^4)$ time algorithm to compute the bisection width of solid grid graphs. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 143–154. Springer, Heidelberg (2011)
[18] Fomin, F.V., Lokshtanov, D., Misra, N., Saurabh, S.: Planar $\mathcal{F}$-deletion: Approximation, kernelization and optimal fpt algorithms. In: Proc. 53rd FOCS, pp. 470–479. IEEE Computer Society (2012)
[19] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Co. (1979)
[20] Garey, M.R., Johnson, D.S., Stockmeyer, L.J.: Some simplified NP-complete graph problems. Theor. Comput. Science 1(3), 237–267 (1976)
[21] Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. SIGACT News 38(1), 31–45 (2007)
[22] Hliněný, P., Oum, S., Seese, D., Gottlob, G.: Width parameters beyond tree-width and their applications. Comput. J. 51(3), 326–362 (2008)
[23] Khot, S.A., Vishnoi, N.K.: The Unique Games Conjecture, integrality gap for cut problems and embeddability of negative type metrics into ℓ_1. In: Proc. 46th FOCS, pp. 53–62. IEEE Computer Society (2005)
[24] Kloks, T., Lee, C.M., Liu, J.: New algorithms for k-face cover, k-feedback vertex set, and k-disjoint cycles on plane and planar graphs. In: Kučera, L. (ed.) WG 2002. LNCS, vol. 2573, pp. 282–295. Springer, Heidelberg (2002)
[25] Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: Image and video synthesis using graph cuts. ACM T. Graphic. 22(3), 277–286 (2003)
[26] Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. SIAM J. Comput. 9, 615–627 (1980)
[27] MacGregor, R.M.: On Partitioning a Graph: a Theoretical and Empirical Study. PhD thesis, University of California, Berkeley (1978)
[28] Marx, D.: Parameterized graph separation problems. Theor. Comput. Sci. 351(3), 394–406 (2006)
[29] Marx, D., O'Sullivan, B., Razgon, I.: Treewidth reduction for constrained separation and bipartization problems. In: Proc. 27th STACS. LIPIcs, vol. 5, pp. 561–572. Dagstuhl (2010)
[30] Marx, D., O'Sullivan, B., Razgon, I.: Finding small separators in linear time via treewidth reduction. CoRR, abs/1110.4765 (2011)
[31] Oum, S.: Approximating rank-width and clique-width quickly. ACM T. Algorithms 5 (1) (2008)
[32] Räcke, H.: Optimal hierarchical decompositions for congestion minimization in networks. In: Proc. 40th STOC, pp. 255–264. ACM (2008)
[33] Soumyanath, K., Deogun, J.S.: On the bisection width of partial k-trees. In: Proc. 20th Southeastern Conference on Combinatorics, Graph Theory, and Computing. Congressus Numerantium, vol. 74, pp. 25–37 (1990)
[34] Wiegers, M.: The k-section of treewidth restricted graphs. In: Rovan, B. (ed.) MFCS 1990. LNCS, vol. 452, pp. 530–537. Springer, Heidelberg (1990)

Fixed-Parameter Tractability and Characterizations of Small Special Treewidth[⋆]

Hans L. Bodlaender[1], Stefan Kratsch[2], and Vincent J.C. Kreuzen[3]

[1] Utrecht University, The Netherlands
h.l.bodlaender@uu.nl
[2] Technical University Berlin, Germany
stefan.kratsch@tu-berlin.de
[3] Maastricht University, The Netherlands
v.kreuzen@maastrichtuniversity.nl

Abstract. We investigate fixed-parameter aspects of the notion of *special treewidth*, which was recently introduced by Courcelle [8,9]. In a special tree decomposition, for each vertex v, the bags containing v form a rooted path in decomposition tree. We resolve an open problem by Courcelle, and show that an algorithm by Bodlaender and Kloks [7] can be modified to obtain for each fixed k, a linear time algorithm that decides if the special treewidth of a given graph is at most k, and if so, finds a corresponding special tree decomposition. This establishes that special treewidth is fixed-parameter tractable.

We obtain characterizations for the class of graphs of special treewidth at most two. The first characterization consists of certain linear structures (termed *mambas*, or equivalently, biconnected partial two-paths) arranged in a specific tree-like fashion, building upon characterizations of biconnected graphs of treewidth two or of pathwidth two. We show that the class of graphs of special treewidth at most two is closed under taking of minors, and give explicitly the obstruction set for this class. For $k \geq 3$, the class of graphs of special treewidth at most k is not closed under taking minors.

1 Introduction

In his recent work on model checking for properties in Monadic Second Order Logic with edge set quantifications, Courcelle [8,9] introduced a variant of the notion of treewidth, called *special treewidth*. A special tree decomposition is one where for each vertex, the bags containing this vertex form a rooted path in the tree. The special treewidth of a graph then is the minimum width over all special tree decompositions of the graph. As path decompositions are a trivial example of special tree decompositions, the notion of special treewidth can be seen as an intermediate form between pathwidth and treewidth.

It is well known that the pathwidth of a graph G is always one smaller than the minimum over all *interval graphs* H that contain G as a subgraph of the

[⋆] This work was partially supported by the Netherlands Organization for Scientific Research (N.W.O.), project "KERNELS: Combinatorial Analysis of Data Reduction".

maximum clique size of H. Similarly, the treewidth of G is the minimum over all *chordal* graphs H that contain G as subgraph of the maximum clique size of H. A similar characterization for special treewidth also exists, now using *rooted path graphs* [9], i.e., a graph has special treewidth at most k, if and only if it is a subgraph of a rooted path graph with maximum clique size at most $k+1$. A graph $G=(V,E)$ is a rooted (unrooted) path graph, if there is a rooted tree T, such that we can associate to each $v \in V$ a path P_v between a node x_v in T and an ancestor of x_v (another node in T), such that two vertices $v, w \in V$, $v \neq w$, are adjacent in G if and only if P_v and P_w have at least one vertex in common.

Courcelle posed as an open problem [9] whether deciding if a given graph has special treewidth at most k is fixed-parameter tractable. In Section 3, we resolve this problem: a modification of an algorithm by Bodlaender and Kloks [7] can be used to obtain a linear time algorithm, but with a constant factor that is exponential in $O(k^3)$.

We then take a closer look at the graphs with special treewidth at most two. For these, we have two main results. First, in Section 4, we give a structural characterization of the graphs with special treewidth two. We show that each biconnected component of a graph of special treewidth two has pathwidth two and then build upon a characterization of biconnected graphs of pathwidth two by de Fluiter and Bodlaender [10,4]. Additional conditions define how the biconnected components can form connected components.

When taking minors, the special treewidth of a graph may increase; Courcelle [9] showed that the classes of graphs with special treewidth k are not closed under taking minors for all $k \geq 5$; we improve upon the construction and show that this holds for all $k \geq 3$. Interestingly, the class of graphs with special treewidth at most two is closed under taking of minors, as is not hard to observe from our structural characterization. (The class of graphs with special treewidth at most one is the class of the forests [8,9], and thus is also closed under taking minors.) By the graph minor theorem of Robertson and Seymour [15] (see for an introduction [11, Chapter 12]), each minor closed class of graphs has a finite characterization in terms of the minor minimal elements of its complement, called the *obstruction set* or *Kuratowski set.* For several minor closed graph classes, the obstruction set is known, e.g., planar graphs ($\{K_3, K_{5,5}\}$ [16]), graphs embeddable in the projective plane [1], graphs of treewidth at most two ($\{K_4\}$, see [11, Proposition 12.4.2]), graphs of treewidth at most three (a set of four graphs [2]), graphs of pathwidth at most two (a set of 110 graphs [12]), and outerplanar graphs ($\{K_4, K_{2,3}\}$). We add to these results the obstruction set for the graphs of special treewidth at most two; in Section 5, we prove for an explicitly given set of six graphs (shown in Figure 2) that it is the obstruction set of graphs of special treewidth at most two. The set contains three graphs that are not biconnected and three biconnected graphs, including K_4.

2 Preliminaries

The notions of pathwidth and treewidth were first introduced by Robertson and Seymour [13,14]. Courcelle [8,9] introduced the notion of special treewidth.

Definition 1. *A* tree decomposition *of a graph $G = (V, E)$ is a pair $(\{X_i | i \in I\}, T = (I, F))$ with $\{X_i | i \in I\}$ a family of subsets (*bags*) of V, and T a rooted tree such that:*

1. $\bigcup_{i \in I} X_i = V$
2. *For all $\{v, w\} \in E$, there is an $i \in I$ with $v, w \in X_i$.*
3. *For each $v \in V$, the set $I_v = \{i \in I | v \in X_i\}$ induces a subtree of T.*

A tree decomposition is a path decomposition *if T is a path. A tree decomposition is a* special tree decomposition *if for each vertex $v \in V$, the set $I_v = \{i \in I | v \in X_i\}$ induces a rooted path in T.*

The width *of a tree decomposition $(\{X_i | i \in I\}, T = (I, F))$ is defined as $\max_{i \in I} |X_i| - 1$. The treewidth (pathwidth; special treewidth) of a graph G is the minimum width of a tree decomposition (path decomposition; special tree decomposition) of G.*

We also denote a path decomposition by the series of its successive bags along the path, i.e., as $(X_1, \ldots, X_r)$.

Courcelle [9] showed that the special treewidth of a graph does not increase when we contract a vertex of degree two with a neighbor. Another useful insight from his work is that for each graph G, if we construct a graph G' by adding a universal vertex to G, then the special treewidth of G' equals the pathwidth of G', which is exactly one larger than the pathwidth of G. From this, it directly follows that deciding if the special treewidth of a graph is at most a given integer k is NP-complete. In [9], we also find that the special treewidth of a graph equals the special treewidth of its connected components, and is either equal or one larger than the special treewidth of its biconnected components, where both cases are possible. We will also use the fact that the special treewidth of a graph is never smaller than its treewidth and never larger than its pathwidth; this follows directly from the definitions.

We say that a graph $H = (W, F)$ is a *minor* of a graph $G = (V, E)$, if a graph isomorphic to H can be obtained from G by a series of the following operations: deletion of a vertex, deletion of an edge, and contraction of an edge. We extend the notion to pairs of a graph and a vertex. For pairs (G, v), (H, w), with $G = (V, E)$, $H = (W, F)$, $v \in V$, $w \in W$, we say that (H, w) is a minor of (G, v), if we can obtain a pair (K, v), $K = (W', F')$, such that there is an isomorphism f from K to H with $f(v) = w$, by a series of the following operations: deletion of a vertex other than v, deletion of an edge, and contraction of an edge, such that whenever we contract an edge with v as endpoint, the resulting vertex is named v. Intuitively, (H, w) is a minor of (G, v), if we can obtain H as minor from G such that v is contracted to w.

3 Fixed-Parameter Tractability of Special Treewidth

Courcelle [9] posed as an open problem whether deciding if a given graph has special treewidth at most k is fixed-parameter tractable. In this section, we show that this is indeed the case. An FPT algorithm for special treewidth directly implies an FPT algorithm for pathwidth: to obtain the latter, run the former on the graph obtained by adding a universal vertex. Thus, an algorithm for special treewidth cannot be expected to be simpler than an algorithm for pathwidth. Indeed, our FPT algorithm for special treewidth is mostly a variant on existing FPT algorithms for treewidth and pathwidth from Bodlaender and Kloks [3,7]. The discussion in this section is not self-contained; at several points, we refer for details to the paper by Bodlaender and Kloks [7]. A self-contained algorithm description and proof would require the tedious and very lengthy repetition of many technicalities.

The algorithm consists of two main steps. Let k be some given parameter, and let $G = (V, E)$ be the input graph.

1. Test if the treewidth of G is at most k, using the algorithm of Bodlaender [3]. If the treewidth of G is larger than k, then the special treewidth of G is also larger than k, and we can return NO. Otherwise, the algorithm of [3] also provides us with a tree decomposition of G of width at most k.
2. Now, use this tree decomposition to execute a dynamic programming algorithm to test if the special treewidth of G is at most k. We use Lemma 1 with $\ell = k$.

Lemma 1. *For each k, ℓ, with $\ell \leq k$, there is an algorithm that given a graph $G = (V, E)$ and a tree decomposition of G of width at most ℓ uses time $O(2^{O(k^3)} \cdot n)$ and correctly decides if the special treewidth of G is at most k, and, if so, outputs a special tree decomposition of G of width at most k.*

Proof. The algorithm is a variant upon a similar algorithm for treewidth of Bodlaender and Kloks [7, Sections 5 and 6]. The algorithm contains many technical details, most identical to the details of the algorithm in [7, Sections 5 and 6]. Instead of repeating these here, we describe only the differences with the algorithm in [7, Sections 5 and 6]. At the cost of not being self-contained, this avoids the need to repeat a lengthy series of technicalities.

As in [7, Sections 5 and 6], first the tree decomposition of G is transformed to a nice tree decomposition of width at most ℓ. To each bag i, we associate the graph $G_i = G[V_i]$, with V_i the union of all bags X_j with $j = i$ or j a descendant of i. A *partial special tree decomposition* at i is a special tree decomposition for G_i. Now, we define the *restriction* of a partial special tree decomposition in the same way as in [7, Definition 5.5]. The definition of a *trunk* and *filled trunk* has a subtle but important difference: while the algorithm of Bodlaender and Kloks sees the trees as unrooted trees, we now work with rooted trees. When making the trunk, we carry out the same procedure as described in [7, Definition 5.6], but never remove the root of the tree.

As a consequence, we have that the trunk of a partial special tree decomposition has at most $2k + 1$ vertices. (Compare [7, Lemma 5.3]: the trunk now has at most $k + 2$ vertices with one neighbor in T.)

The *tree model*, *trunk representation*, *typical list*, *characteristic*, and *full set of characteristics* are defined identically as in [7], except that we now assume that the partial tree decompositions are partial special tree decompositions.

The computation of full sets is almost identical to the computation of full sets for the treewidth algorithm in [7, Sections 5.2, 5.3, 5.4 and 5.5], with one difference. After computing the full set as described for **join** and **introduce** nodes i with corresponding bag X_i, we check for each characteristic in the so-far computed full set, whether for each vertex $v \in X_i$ the bags in the trunk that contain v form a rooted path. If this is not the case, we delete this characteristic of the full set.

Intuitively, we just run the algorithm for treewidth in [7], except that we always include the root in the trunk, and at runtime, delete all characteristics that violate the property for special tree decompositions for one of the vertices of the bag that we currently look at. In this way, the running time is of the same order as the running time of the algorithm in [7].

The proof that this is correct follows the same lines as the correctness proof in [7]. For the decision variant, the last step just checks if the full set for the root bag of the nice tree decomposition is nonempty. As each characteristic in a full set always corresponds to a special tree decomposition, one can use the algorithm from [7, Section 6] without changes to turn the decision algorithm into an algorithm that also constructs corresponding special tree decompositions of width at most k. $\square$

4 Characterizing Special Treewidth Two via Mambas

In this section, we give a structural characterization of graphs with special treewidth two. A central role in this characterization is played by biconnected graphs of pathwidth at most two, which we will call *mambas*. We will show that biconnected graphs of special treewidth two have pathwidth two, and thus have a linear structure. We then introduce the concept of *mamba trees*. Intuitively, a mamba tree is built as a composition of separate mambas; a single mamba is a mamba tree, and further mambas can be attached at “head vertices” (to be defined later). In the following section, we continue the investigation of the characterization using ‘paths of cycles’.

Lemma 2. *Let G be a biconnected graph. G has special treewidth at most two, if and only if G has pathwidth at most two.*

Proof. If the pathwidth of G is at most two, then trivially, the special treewidth of G is at most two, as each path decomposition is a special tree decomposition.

For the other direction, assume that the special treewidth of G is at most two. Take a special tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of G of width two that is minimal in the following sense: there is no special tree decomposition of G of width two with fewer bags, and whenever we remove a vertex from a bag from the special tree decomposition, we no longer have a special tree decomposition of G.

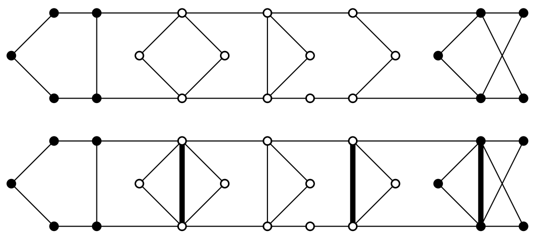


Fig. 1. A mamba G and the cell completion $\bar{G}$ of G. Black vertices are *head vertices*; white vertices cannot be a head vertex. See the discussion later in this section.

Suppose now that $(\{X_i | i \in I\}, T = (I, F))$ is not a path decomposition. Then there is a bag X_j such that X_j has two child-bags, say X_a and X_b.

We consider a number of cases. As we reach a contradiction in each of the cases, the result follows.

Case 1: $X_j \cap X_a = \emptyset$. G is not connected, contradiction.

Case 2: $|X_j \cap X_a| = 1$. Let $v \in X_j \cap X_a$ and let $W = \bigcup\{X_k | k = a$ or X_k is a descendant of $X_a\}$. If $|W| = 1$, then $W = \{v\}$, and then the special tree decomposition is not minimal, as we can delete a and all descendant bags. So, assume $|W| > 1$. Take a vertex $w \in W$ with $w \neq v$. If $|X_b| = 1$, then we obtain a contradiction with minimality or with connectivity of G. So let $x \in X_b$ with $x \neq v$. All paths from x to w must use v. Thus v is a cut vertex and G is not biconnected, contradiction.

Case 3: $|X_j \cap X_a| \geq 2$. If $|X_j \cap X_b| < 2$, we reach a contradiction in the same way as in Case 1 or Case 2. So, we may assume that $|X_j \cap X_b| \geq 2$. Since T is a special tree decomposition, we know that $X_a \cap X_b = \emptyset$ and thus $|X_j| \geq 4$. Thus, the width of $(\{X_i | i \in I\}, T = (I, F))$ is at least three, contradiction. Hence $(\{X_i | i \in I\}, T = (I, F))$ is a path decomposition of G of width at most two. □

In the remainder of the paper, we use the term *mamba* for a biconnected component of a graph of special treewidth at most two, i.e., a biconnected component of a graph is a *mamba*, if and only if it has special treewidth at most two, if and only if it has pathwidth at most two.

There are two different cases for mambas: the trivial case of biconnected components that consist of a single edge, and the case of biconnected components that have at least three vertices. In the latter case, the component contains a cycle, and hence its treewidth, special treewidth, and pathwidth all equal two.

Connected graphs of special treewidth two are formed by attaching mambas to each other in a specific way. We first need the notion of *head vertices*.

Definition 2. *A* head vertex *of a mamba $G = (V, E)$ is a vertex $v \in V$ such that there is a path decomposition $(X_1, \ldots, X_m)$ of G of width at most two with $v \in X_1$.*

First, we give a recursive definition of a *mamba tree*; we will then (Theorem 1) show that these precisely characterize the connected graphs of special treewidth two, and then (Theorem 3) characterize which vertices in a mamba are head vertices. Mamba trees characterize connected graphs of special treewidth two (cf. Theorem 1; its proof is given in the full paper); a characterization in terms of an obstruction set is in the next section.

Definition 3. *The class of* mamba trees *is the class of graphs recursively defined as follows.*

- *Each mamba is a mamba tree.*
- *For each mamba tree G and each mamba M, the graph obtained by identifying a vertex in G with a head vertex in M, is a mamba tree.*

Theorem 1. *A graph $G = (V, E)$ has special treewidth at most two, if and only if G is a disjoint union of mamba trees.*

The structure of biconnected graphs of pathwidth two was studied by de Fluiter and Bodlaender [10,4]. We give this structure below; after that, the structural characterization is finished by characterizing which vertices can be head vertex (cf. Theorem 3).

Definition 4 ([10,4]). *The* cell completion $\bar{G}$ *of a biconnected graph $G = (V, E)$ is the graph, obtained from G by adding an edge $\{v, w\}$ for all pairs of nonadjacent vertices v, $w \in V$ such that $G[V - \{v, w\}]$ has at least three connected components.*

Definition 5 ([10,4,6]). *The class of* trees of cycles *is the class of graphs recursively defined as follows.*

- *Each cycle is a tree of cycles.*
- *For each tree of cycles G and each cycle C, the graph obtained from G and C by taking the disjoint union and then identifying an edge and its end vertices in G with an edge and its end vertices in C, is a tree of cycles.*

Note that two different chordless cycles in a tree of cycles have at most one edge in common.

Definition 6 ([10,4]). *A* path of cycles *is a tree of cycles G such that each chordless cycle of G has at most two edges which are contained in other chordless cycles of G, and if an edge $e \in E(G)$ is contained in $m \geq 3$ chordless cycles of G, then at least $m - 2$ of these cycles have no other edges in common with other chordless cycles, and consist of three vertices each.*

Theorem 2 ([10,4]). *Let G be a biconnected graph with at least three vertices. G has pathwidth two if and only if $\bar{G}$ is a path of cycles.*

Thus, we have that each biconnected component of a graph of pathwidth two is either a single edge or has a cell completion that is a path of cycles. A path of cycles can be represented by a *cycle path*, which is defined next.

Definition 7 ([10,4]). *Let G be a path of cycles, let $C = (C_1, \ldots, C_p)$ be a sequence of chordless cycles in G, such that each chordless cycle in G appears exactly once in the sequence, and for $1 \leq i \leq p-1$, C_i shares exactly one edge with C_{i+1}. Let $E = (e_1, \ldots, e_{p-1})$ be the corresponding set of common edges. The pair (C, E) is called a* cycle path *for G.*

For easier discussion, we distinguish two different types of cycles on a cycle path (C, E), $C = (C_1, \ldots, C_p)$ and $E = (e_1, \ldots, e_{p-1})$. A cycle C_i with three vertices that has an edge that is contained in at least three cycles of the cycle path and $i \notin \{1, p\}$ is called a *scale cycle*; all other cycles are called *body cycles.* Note that we always take C_1 and C_p as body cycles. The body cycles form a path: each body cycle except C_1 and C_p is incident to exactly one body cycle with a smaller index and exactly one body cycle with a larger index. Each of the scale cycles shares an edge (in the cell completion) with two successive body cycles (and possibly other scale cycles). Observe that when an edge e belongs to the cell completion $\bar{G}$ but not to G, then there must be two body cycles and a scale cycle that use e. The following theorem characterizes the head vertices for mambas, that are not just a single edge; for the latter, both endpoints of this edge are head vertices. The proof is given in the full paper.

Theorem 3. *Let $G = (V, E)$ be a mamba, and let $v \in V$. Suppose that G has at least three vertices. The following statements are equivalent.*

1. *v is a head vertex.*
2. *There exists a cycle path (C, E), with $C = (C_1, \ldots, C_p)$ for G, such that v is a vertex on C_1.*
3. *For each cycle path (C, E) with $C = (C_1, \ldots, C_p)$ and $E = (e_1, \ldots, e_{p-1})$, at least one of the following cases holds:*
   (a) *v is a vertex on C_1.*
   (b) *v is a vertex on C_p.*
   (c) *There is an i with v on a cycle C_i; C_1 and C_i are cycles of length three, and $e_1 = e_2 = \cdots = e_{i-1}$, i.e., $C_2, \ldots, C_i$ are scale cycles.*
   (d) *There is an i with v on a cycle C_i; C_p and C_i are cycles of length three, and $e_i = e_{i+1} = \cdots = e_{p-1}$, i.e., $C_i, \ldots, C_{p-1}$ are scale cycles.*

Thus, a graph has special treewidth at most two, if and only if for each connected component, we can arrange the biconnected components of it in a tree like fashion, for each biconnected component its cell completion is a path of cycles, and a biconnected component (not corresponding to the root of the tree) has a head vertex (as characterized above) as cut vertex through which it is connected to the parent biconnected component.

5 The Obstruction Set of Special Treewidth Two

The main result of this section is the following.

Theorem 4. *The class of graphs with special treewidth at most two is closed under taking of minors, and its obstruction set is $\{K_4, S_3, D_3, G_1, G_2, G_3\}$.*

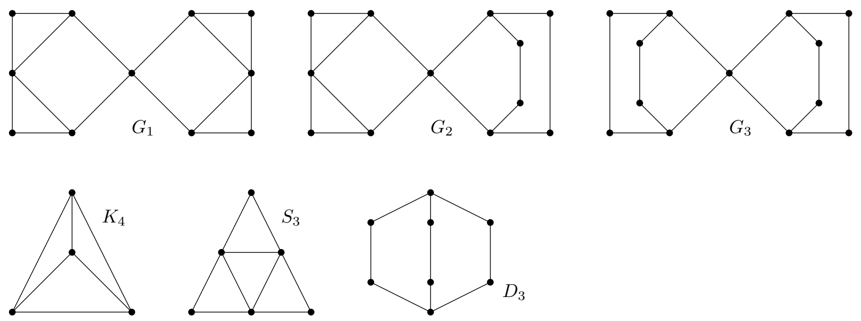


Fig. 2. The six graphs of the obstruction set for graphs of special treewidth two: G_1, G_2, G_3, K_4, S_3 and D_3

In Figure 2, the graphs in the obstruction set are displayed. The result is complemented by the following proposition, which improves upon a result by Courcelle [9], who showed it for $k \geq 5$. The proof is given in the full paper.

Proposition 1. *Let $\mathcal{G}_k$ be the graphs of special treewidth at most k. If $k \geq 3$, then $\mathcal{G}_k$ is not closed under taking minors.*

The remainder of the section is devoted to the proof of Theorem 4; some case analysis is omitted from this extended abstract and given in the full paper. From our structural characterization of graphs of special treewidth at most two of the previous sections, an easy case analysis shows that this characterization is minor closed. From the characterization, it follows immediately that each of the graphs in the set $\{K_4, S_3, D_3, G_1, G_2, G_3\}$ has special treewidth at least three; a tedious case analysis shows that each proper minor of the graphs has special treewidth at most two. So $\{K_4, S_3, D_3, G_1, G_2, G_3\}$ is a subset of the obstruction set of the graphs of special treewidth at most two. Thus, Theorem 4 follows from the next lemma.

Lemma 3. *Let G be a graph that does not contain K_4, S_3, D_3, G_1, G_2 or G_3 as a minor. Then the special treewidth of G is at most two.*

Proof. Suppose that the lemma does not hold. Let G be a minimal counterexample; i.e., no minor of G is a counterexample. Clearly, no minor of G contains one of the six graphs as a minor, so minimality is equivalent here to assuming that all minors of G have special treewidth two.

As the special treewidth of a graph is the maximum of its connected components, we clearly have that G is connected. As G does not contain K_4 as a minor, the treewidth of G is at most two.

Case 1: G is not biconnected We use the well known fact that the biconnected components of a graph form a tree. Consider a leaf of the tree of biconnected components, i.e., a biconnected component with vertex set W_1 that contains exactly one cut vertex, say v.

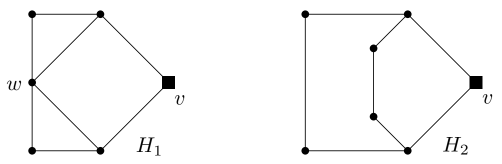


Fig. 3. Biconnected components of obstructions that are not biconnected

Claim. There is no special tree decomposition of $G[W_1]$ of width at most two that contains v in its root bag.

Proof. Suppose there is such a special tree decomposition. $G[V \backslash W_1 \cup \{v\}]$ is a minor of G so has special treewidth at most two. Take such a special tree decomposition. Let j_v be the bag with maximum depth that contains v. Take a special tree decomposition of $G[W_1]$ of width at most two that contains v in its root bag i_r. Join these two special tree decompositions, making i_r a child of j_v. This gives a special tree decomposition of G of width at most two. □

Recall our extension of the notion of minor to pairs of a graph and a vertex.

Claim. $(G[W_1], v)$ has (H_1, v) or (H_2, v) as a minor, with H_1 and H_2 as in Figure 3, with v the marked vertex.

The proof of this claim is given in the full paper. It is based on a case analysis, using the characterization of head vertices of the previous section.

It follows that G contains G_1, G_2 or G_3 as a minor: take two biconnected components of G that are a leaf of the tree of biconnected components, say with vertex sets Z_1 and Z_2, and with cut vertices v_1 and v_2. Do contractions and deletions to $(G[Z_1], v_1)$ and obtain (H_1, v_1) or (H_2, v_1): by the claim above, at least one of these is possible. Similarly, obtain (H_1, v_2) or (H_2, v_2) as minor of $(G[Z_2], v_2)$. Now contract all other vertices and v_1 and v_2 to one vertex. If we obtained H_1 for both connected components, we have G_1 as minor. If one of the components gave H_1 and the other gave H_2 (in either order), then we have G_2 as minor. If both gave H_2, then we obtained G_3 as minor. In each case, we reached a contradiction.

Case 2: G is biconnected As G has treewidth two and is biconnected, its cell completion is a tree of cycles (see Bodlaender and Kloks [6] and the discussion in the previous section). However, as we assumed it does not have special treewidth two, the cell completion of G is not a path of cycles. Thus, the cell completion of G, $\bar{G}$, must have one of the following two properties.

– There is a chordless cycle that has at least three edges which are contained in other chordless cycles of the graph.
– There is an edge e which is contained in m chordless cycles, $m \geq 3$, such that at most $m-3$ of these cycles have no other edges in common with other chordless cycles and consist of three vertices.

First, assume that the cell completion of G contains a chordless cycle C which has three or more edges contained in other chordless cycles. Let e_1, e_2, and e_3 be these edges. Consider e_1. Either $e_1 \in E$, or e_1 is an edge that is added by the cell completion of G. In the former case, there is a path in G between the endpoints of e_1 that is disjoint from C. In the latter case, there are two such paths. We now can contract one of these paths to the edge e_1; in both cases, we contract the remaining path to two edges that form a triangle with e_1. Note that these paths are disjoint for e_1, e_2 and e_3, as the cell completion of G is a tree of cycles. If we apply the same steps to e_2 and e_3, and contract the other edges on C, and remove the remainder of the graph, we obtain S_3 as a minor.

Second, assume we have an edge $e = \{v, w\}$ that is contained in $m \geq 3$ chordless cycles, with at most $m - 3$ of these cycles having no other edges in common with other chordless cycles and consisting of three vertices. I.e., there are at least three cycles C_1, C_2, C_3 that share e, that either have at least four vertices, or have an edge in common with another chordless cycle.

Note that C_1, C_2 and C_3 overlap only at e, and the trees of cycles attached to (and including) C_1, C_2, and C_3 are disjoint from each other.

Consider C_1. We can find a path P_1 from v to w with at least three edges as follows. If C_1 contains at least four vertices, we use the path obtained by removing e from C_1. Otherwise, C_1 has an edge, say $e_1 = \{x, y\}$ with another chordless cycle, say C_4. Then, take the path from v to w by removing e from C_1, and replacing e_1 by the path from x to y that is formed by $C_4 - e_1$.

In the same way, we can find paths P_2, P_3 from v to w with at least three edges for C_2 and C_3 in the cell completion of G. These paths are disjoint, which, again, follows from the fact that the cell completion of G is a tree of cycles.

Now, for each edge on these paths that is not an edge in G, i.e., was added by making the cell completion, notice that there is a path between its endpoints that is disjoint from P_1, P_2 and P_3; replace the edge by this path. By repeating this step, we obtain three disjoint paths between v and w, each with at least three edges. Thus G contains D_3 as a minor. Contradiction. $\square$

6 Conclusions

We end this paper with some final remarks and open problems. The fixed-parameter algorithm is exponential in k^3; pathwidth has an FPT algorithm that is exponential in k^2 (e.g., first run the algorithm from [5] to find a tree decomposition of width $O(k)$, and then the algorithm for pathwidth from [7]), and thus we ask whether an algorithm for special treewidth with running time $O^*(2^{O(k^2)})$ exists.

A variant of special treewidth, that was also suggested by Arie Koster, is a notion we term *spaghetti treewidth*: a tree decomposition is a spaghetti tree decomposition if for each vertex v, the bags that contain v form a path in the tree. I.e., the difference with special treewidth is that we no longer demand that the path induced by the bags is rooted. One can alternatively define the spaghetti treewidth of a graph G as the minimum over all undirected path graphs H that contain G as a subgraph of the maximum clique size of H.

With arguments, similar to those in Section 3, one can argue that spaghetti treewidth is fixed parameter tractable, with an algorithm that runs in $O(2^{O(k^3)})$ time. Again, adding a universal vertex to a graph gives a graph whose spaghetti treewidth equals its pathwidth, proving that spaghetti treewidth is NP-complete.

Similar to Proposition 1, we have that for each $k \geq 3$, the class of graphs of spaghetti treewidth at most k is not closed under taking minors. Recently, O-Joung Kwon and Seongmin Ok showed that the graphs of spaghetti treewidth two are closed under minor taking, and that the obstruction set for the class of graphs of spaghetti treewidth two equals $\{K_4, D_3\}$.[1]

References

1. Archdeacon, D.: A Kuratowski theorem for the projective plane. Journal of Graph Theory 7, 325–334 (1983)
2. Arnborg, S., Proskurowski, A., Corneil, D.G.: Forbidden minors characterization of partial 3-trees. Discrete Mathematics 80, 1–19 (1990)
3. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing 25, 1305–1317 (1996)
4. Bodlaender, H.L., de Fluiter, B.: On intervalizing k-colored graphs for DNA physical mapping. Discrete Applied Mathematics 71, 55–77 (1996)
5. Bodlaender, H.L., Drange, P.G., Dregi, M.S., Fomin, F.V., Lokshtanov, D., Pilipczuk, M.: A $O(c^k n)$ 5-approximation algorithm for treewidth. CoRR, abs/1304.6321 (2013), Extended abstract to appear in FOCS 2013
6. Bodlaender, H.L., Kloks, T.: A simple linear time algorithm for triangulating three-colored graphs. Journal of Algorithms 15, 160–172 (1993)
7. Bodlaender, H.L., Kloks, T.: Efficient and constructive algorithms for the pathwidth and treewidth of graphs. Journal of Algorithms 21, 358–402 (1996)
8. Courcelle, B.: Special tree-width and the verification of monadic second-order graph properties. In: FSTTCS. LIPIcs, vol. 8, pp. 13–29. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2010)
9. Courcelle, B.: On the model-checking of monadic second-order formulas with edge set quantifications. Discrete Applied Mathematics 160, 866–887 (2012)
10. de Fluiter, B.: Algorithms for Graphs of Small Treewidth. PhD thesis, Utrecht University (1997)
11. Diestel, R.: Graph theory, 4th edn. Springer (2010)
12. Kinnersley, N.G., Langston, M.A.: Obstruction set isolation for the gate matrix layout problem. Discrete Applied Mathematics 54, 169–213 (1994)
13. Robertson, N., Seymour, P.D.: Graph minors. I. Excluding a forest. Journal of Combinatorial Theory, Series B 35, 39–61 (1983)
14. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. Journal of Algorithms 7, 309–322 (1986)
15. Robertson, N., Seymour, P.D.: Graph minors. XX. Wagner's conjecture. Journal of Combinatorial Theory, Series B 92, 325–357 (2004)
16. Wagner, K.: Über eine Eigenshaft der ebenen Complexe. Math. Ann. 14, 570–590 (1937)

[1] Personal communication.

The θ_5-Graph is a Spanner

Prosenjit Bose, Pat Morin, André van Renssen, and Sander Verdonschot[⋆]

School of Computer Science, Carleton University
{jit,morin}@scs.carleton.ca, {andre,sander}@cg.scs.carleton.ca

Abstract. Given a set of points in the plane, we show that the θ-graph with 5 cones is a geometric spanner with spanning ratio at most $\sqrt{50+22\sqrt{5}} \approx 9.960$. This is the first constant upper bound on the spanning ratio of this graph. The upper bound uses a constructive argument, giving a, possibly self-intersecting, path between any two vertices, whose length is at most $\sqrt{50+22\sqrt{5}}$ times the Euclidean distance between the vertices. We also give a lower bound on the spanning ratio of $\frac{1}{2}(11\sqrt{5}-17) \approx 3.798$.

1 Introduction

A t-spanner ($t \geq 1$) of a weighted graph G is a spanning subgraph H with the property that for all pairs of vertices, the weight of the shortest path between the vertices in H is at most t times the weight of the shortest path in G. The *spanning ratio* of H is the smallest t for which it is a t-spanner. The graph G is referred to as the *underlying graph*. In this paper, the underlying graph is the complete graph on a finite set of n points in the plane and the weight of an edge is the Euclidean distance between its endpoints. A spanner of such a graph is called a *geometric spanner*. We focus on a specific class of geometric spanners, called θ-graphs. For a more comprehensive overview of geometric spanners, we refer the reader to the book by Narasimhan and Smid [1].

Introduced independently by Clarkson [2] and Keil [3], θ-graphs form an important class of geometric spanners. Given a set P of points in the plane, we consider each point $u \in P$ and partition the plane into m cones (regions in the plane between two rays originating from the same point) with apex u, each defined by two rays at consecutive multiples of $\theta = 2\pi/m$ radians from the negative y-axis. We label the cones C_0 through C_{m-1},

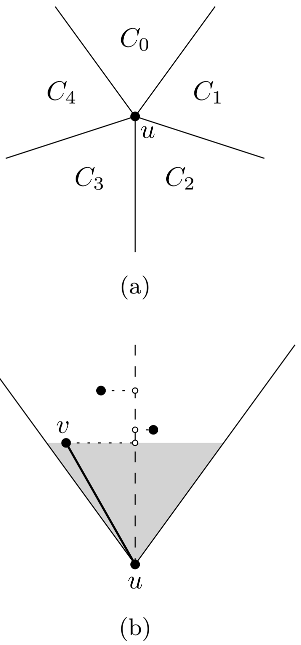


Fig. 1. (a) The cones around a vertex u. (b) The construction of the θ_5-graph.

[⋆] Research supported in part by NSERC.

in clockwise order around u, starting from the top (see Figure 1a). If the apex is not clear from the context, we use C_i^u to denote cone C_i with apex u. We refer to the θ-graph with m cones as the θ_m-graph.

To build the θ-graph, we consider each vertex u and add an edge to the 'closest' vertex in each of its cones. However, instead of using the Euclidean distance, we measure distance by projecting each vertex onto the bisector of that cone (see Figure 1b). We use this definition of *closest* in the remainder of the paper. For simplicity, we assume that no two points lie on a line parallel or perpendicular to a cone boundary, guaranteeing that each vertex connects to at most one vertex in each cone. Thus, the graph has at most $m \cdot n$ edges.

Ruppert and Seidel [4] showed that for $m \geq 7$, the spanning ratio of these graphs is at most $1/(1 - 2\sin(\theta/2))$, but until recently little was known about θ-graphs with fewer cones. The only results so far are a matching upper and lower bound of 2 on the spanning ratio of the θ_6-graph by Bonichon *et al.* [5], and negative results showing that there is no constant t for which the θ_2- and θ_3-graphs are t-spanners (shown by El Molla [6] for Yao-graphs, but the proof translates to θ-graphs). Very recently, the θ_4-graph was shown to be a spanner as well [7], leaving the θ_5-graph as the only θ-graph for which it is not known whether the graph is a spanner or not. We answer this question affirmatively.

Choosing a θ_m-graph with smallest possible value of m is important for many practical applications where the cost of a network is mostly determined by the number of edges. One such example is point-to-point wireless networks. These networks use narrow directional wireless transceivers that can transmit over long distances (up to 50km [8,9]). The cost of an edge in such a network is therefore equal to the cost of the two transceivers that are used at each endpoint of that edge. If the transceivers are distributed uniformly at random, the cost of building a θ_6-graph is approximately 29% higher than the cost of building a θ_5-graph [10].

We present the first constant upper bound on the spanning ratio of the θ_5-graph, proving that it is a geometric spanner. Since the proof is constructive, it gives us a path between any two vertices, u and w, with length at most $\sqrt{50 + 22\sqrt{5}} \approx 9.960$ times $|uw|$. Surprisingly, this path can cross itself, a property we observed for the shortest path as well. We also prove a lower bound on the spanning ratio of $\frac{1}{2}(11\sqrt{5} - 17) \approx 3.798$.

2 Connectivity

To introduce the structure of the spanning proof, we first show that the θ_5-graph is connected.

Given two vertices u and v, we define their *canonical triangle* T_{uv} to be the triangle bounded by the cone of u that contains v and the line through v perpendicular to the bisector of that cone. For example, the shaded region in Figure 1b is the canonical triangle T_{uv}. Note that for any pair of vertices u and v, there are two canonical triangles: T_{uv} and T_{vu}. We equate the size $|T_{uv}|$ of a canonical triangle to the length of one of the sides incident to the apex u. This gives us the useful property that any line between u and a point inside the triangle has length at most $|T_{uv}|$.

Theorem 1. *The θ_5-graph is connected.*

Proof. We prove that there is a path between any (ordered) pair of vertices in the θ_5-graph, using induction on the size of their canonical triangle. Formally, given two vertices u and w, we perform induction on the rank of T_{uw} among the canonical triangles of all pairs of vertices, when ordered by size. For ease of description, we assume that w lies in the right half of C_0^u. The other cases are analogous.

If T_{uw} has rank 1, it is the smallest canonical triangle. Therefore there can be no point closer to u in C_0^u, so the θ_5-graph must contain the edge (u, w). This proves the base case.

If T_{uw} has a larger rank, our inductive hypothesis is that there exists a path between any pair of vertices with a smaller canonical triangle. Let a and b be the left and right corners of T_{uw}. Let m be the midpoint of ab and let x be the intersection of ab and the bisector of $\angle mub$ (see Figure 2a).

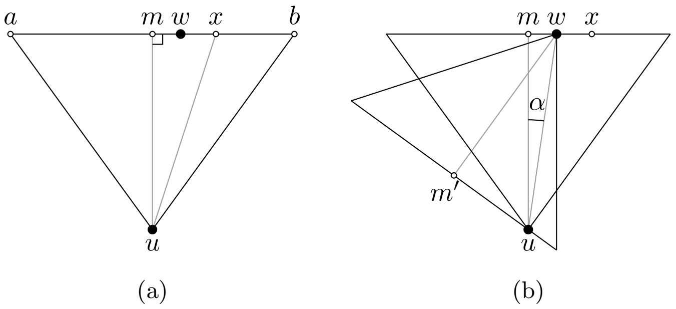


Fig. 2. (a) The canonical triangle T_{uw}. (b) If w lies between m and x, T_{wu} is smaller than T_{uw}.

If w lies to the left of x, consider the canonical triangle T_{wu}. Let m' be the midpoint of the side of T_{wu} opposite w and let $\alpha = \angle muw$ (see Figure 2b). We can express the size of T_{wu} as follows.

$$|T_{wu}| = \frac{|wm'|}{\cos\frac{\pi}{5}} = \frac{\cos\angle uwm' \cdot |uw|}{\cos\frac{\pi}{5}} = \frac{\cos\left(\frac{\pi}{5} - \alpha\right) \cdot \frac{|um|}{\cos\alpha}}{\cos\frac{\pi}{5}} = \frac{\cos\left(\frac{\pi}{5} - \alpha\right)}{\cos\alpha} \cdot |T_{uw}|$$

Since w lies to the left of x, the angle α is less than $\pi/10$, which means that $\cos(\frac{\pi}{5} - \alpha)/\cos\alpha$ is less than 1. Hence T_{wu} is smaller than T_{uw} and by induction, there is a path between w and u. Since the θ_5-graph is undirected, we are done in this case. The rest of the proof deals with the case where w lies on or to the right of x.

If T_{wu} is empty, there is an edge between u and w and we are done, so assume that this is not the case. Then there is a vertex v_w that is closest to w in C_3^w (the cone of w that contains u). This gives rise to four cases, depending on the location of v_w (see Figure 3a). In each case, we will show that T_{uv_w} is smaller than T_{uw} and hence we can apply induction to obtain a path between u and v_w. Since v_w is the closest vertex to w in C_3, there is an edge between v_w and w, completing the path between u and w.

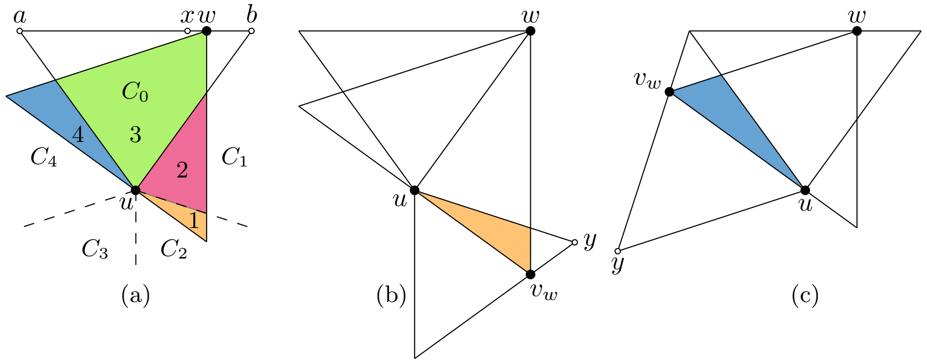


Fig. 3. (a) The four cases for v_w. (b) Case 1: The situation that maximizes $|T_{uv_w}|$ when v_w lies in C_2^u. (c) Case 4: The situation that maximizes $|T_{uv_w}|$ when v_w lies in C_4^u.

Case 1. v_w lies in C_2^u. In this case, the size of T_{uv_w} is maximized when v_w lies in the bottom right corner of T_{wu} and w lies on b. Let y be the rightmost corner of T_{uv_w} (see Figure 3b). Using the law of sines, we can express the size of T_{uv_w} as follows.

$$|T_{uv_w}| = |uy| = \frac{\sin \angle uv_wy}{\sin \angle uyv_w} \cdot |uv_w| = \frac{\sin \frac{3\pi}{5}}{\sin \frac{3\pi}{10}} \cdot \tan \frac{\pi}{5} \cdot |T_{uw}| < |T_{uw}|$$

Case 2. v_w lies in C_1^u. In this case, the size of T_{uv_w} is maximized when w lies on b and v_w lies almost on w. By symmetry, this gives $|T_{uv_w}| = |T_{uw}|$. However, v_w cannot lie precisely on w and must therefore lie a little closer to u, giving us that $|T_{uv_w}| < |T_{uw}|$.

Case 3. v_w lies in C_0^u. As in the previous case, the size of T_{uv_w} is maximized when v_w lies almost on w, but since v_w must lie closer to u, we have that $|T_{uv_w}| < |T_{uw}|$.

Case 4. v_w lies in C_4^u. In this case, the size of T_{uv_w} is maximized when v_w lies in the left corner of T_{wu} and w lies on x. Let y be the bottom corner of T_{uv_w} (see Figure 3c). Since x is the point where $|T_{uw}| = |T_{wu}|$, and v_wyuw forms a parallelogram, $|T_{uv_w}| = |T_{uw}|$. However, by general position, v_w cannot lie on the boundary of T_{wu}, so it must lie a little closer to u, giving us that $|T_{uv_w}| < |T_{uw}|$. $\square$

3 Spanning Ratio

In this section, we prove an upper bound on the spanning ratio of the θ_5-graph.

Lemma 1. *Between any pair of vertices u and w of a θ_5-graph, there is a path of length at most $c \cdot |T_{uw}|$, where $c = 2\left(2+\sqrt{5}\right) \approx 8.472$.*

Proof. We begin in a way similar to the proof of Theorem 1. Given an ordered pair of vertices u and w, we perform induction on the size of their canonical triangle. If $|T_{uw}|$ is minimal, there must be a direct edge between them. Since $c > 1$ and any edge inside T_{uw} with endpoint u has length at most $|T_{uw}|$, this proves the base case. The rest of the proof deals with the inductive step, where we assume that there exists a path with length at most $c \cdot |T|$ between every pair of vertices whose canonical triangle T is smaller than T_{uw}. As in the proof of Theorem 1, we assume that w lies in the right half of C_0^u. If w lies to the left of x, we have seen that T_{wu} is smaller than T_{uw}. Therefore we can apply induction to obtain a path of length at most $c \cdot |T_{wu}| < c \cdot |T_{uw}|$ between u and w. Hence we need to concern ourselves only with the case where w lies on or to the right of x.

If u is the vertex closest to w in C_3^w or w is the closest vertex to u in C_0^u, there is a direct edge between them and we are done by the same reasoning as in the base case. Therefore assume that this is not the case and let v_w be the vertex closest to w in C_3^w. We distinguish the same four cases for the location of v_w (see Figure 3a). We already showed that we can apply induction on T_{uv_w} in each case. This is a crucial part of the proof for the first three cases.

Most of the cases come down to finding a path between u and w of length at most $(g + h \cdot c) \cdot |T_{uw}|$, for constants g and h with $h < 1$. The smallest value of c for which this is bounded by $c \cdot |T_{uw}|$ is $g/(1-h)$. If this is at most $2\left(2+\sqrt{5}\right) \approx 8.472$, we are done.

Case 1. v_w lies in C_2^u. By induction, there exists a path between u and v_w of length at most $c \cdot |T_{uv_w}|$. Since v_w is the closest vertex to w in C_3^w, there is a direct edge between them, giving a path between u and w of length at most $|wv_w| + c \cdot |T_{uv_w}|$.

Given any initial position of v_w in C_2^u, we can increase $|wv_w|$ by moving w to the right. Since this does not change $|T_{uv_w}|$, the worst case occurs when w lies on b. Then we can increase both $|wv_w|$ and $|T_{uv_w}|$ by moving v_w into the bottom corner of T_{wu}. This gives rise to the same worst-case configuration as in the proof of Theorem 1, depicted in Figure 3b. Building on the analysis there, we can bound the worst-case length of the path as follows.

$$|wv_w| + c \cdot |T_{uv_w}| \quad = \quad \frac{|T_{uw}|}{\cos\frac{\pi}{5}} + c \cdot \frac{\sin\frac{3\pi}{5}}{\sin\frac{3\pi}{10}} \cdot \tan\frac{\pi}{5} \cdot |T_{uw}|$$

This is at most $c \cdot |T_{uw}|$ for $c \geq 2\left(2+\sqrt{5}\right)$. Since we picked $c = 2\left(2+\sqrt{5}\right)$, the theorem holds in this case. Note that this is one of the cases that determines the value of c.

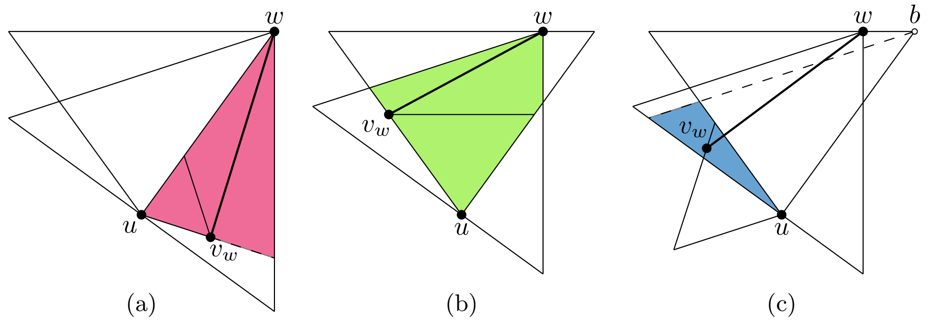


Fig. 4. (a) Case 2: Vertex v_w lies on the boundary of C_1^u after moving it down along the side of T_{uv_w}. (b) Case 3: Vertex v_w lies on the boundary of C_0^u after moving it left along the side of T_{uv_w}. (c) Case 4: Vertex v_w lies in $C_4^u \cap C_3^b$.

Case 2. v_w lies in C_1^u. By the same reasoning as in the previous case, we have a path of length at most $|wv_w| + c \cdot |T_{uv_w}|$ between u and w and we need to bound this length by $c \cdot |T_{uw}|$.

Given any initial position of v_w in C_1^u, we can increase $|wv_w|$ by moving w to the right. Since this does not change $|T_{uv_w}|$, the worst case occurs when w lies on b. We can further increase $|wv_w|$ by moving v_w down along the side of T_{uv_w} opposite u until it hits the boundary of C_1^u or C_3^w, whichever comes first (see Figure 4a).

Now consider what happens when we move v_w along these boundaries. If v_w lies on the boundary of C_1^u and we move it away from u by Δ, $|T_{uv_w}|$ increases by Δ. At the same time, $|wv_w|$ might decrease, but not by more than Δ. Since $c > 1$, the total path length is maximized by moving v_w as far from u as possible, until it hits the boundary of C_3^w. Once v_w lies on the boundary of C_3^w, we have that $|T_{uv_w}| = |T_{uw}| - |wv_w| \cdot \left(3 - \sqrt{5}\right)/2$. Since $c > 2/\left(3 - \sqrt{5}\right) \approx 2.618$, this gives $|wv_w| + c \cdot |T_{uv_w}| = c \cdot |T_{uw}| - \left(c \cdot \left(3 - \sqrt{5}\right)/2 - 1\right) \cdot |wv_w| < c \cdot |T_{uw}|$.

Case 3. v_w lies in C_0^u. Again, we have a path of length at most $|wv_w| + c \cdot |T_{uv_w}|$ between u and w and we need to bound this length by $c \cdot |T_{uw}|$.

Given any initial position of v_w in C_0^u, moving v_w to the left increases $|wv_w|$ while leaving $|T_{uv_w}|$ unchanged. Therefore the path length is maximized when v_w lies on the boundary of either C_0^u or C_3^w, whichever it hits first (see Figure 4b).

Again, consider what happens when we move v_w along these boundaries. Similar to the previous case, if v_w lies on the boundary of C_0^u and we move it away from u by Δ, $|T_{uv_w}|$ increases by Δ, while $|wv_w|$ might decrease by at most Δ. Since $c > 1$, the total path length is maximized by moving v_w as far from u as possible, until it hits the boundary of C_3^w. Once there, the situation is symmetric to the previous case, with $|T_{uv_w}| = |T_{uw}| - |wv_w| \cdot \left(3 - \sqrt{5}\right)/2$. Therefore the theorem holds in this case as well.

Case 4. v_w lies in C_4^u. This is the hardest case. Similar to the previous two cases, the size of T_{uv_w} can be arbitrarily close to that of T_{uw}, but in this case $|wv_w|$

does not approach 0. This means that simply invoking the inductive hypothesis on T_{uv_w} does not work, so another strategy is required. We first look at a subcase where we *can* apply induction directly, before considering four subcases for the position of v_u, the closest vertex to u in C_0.

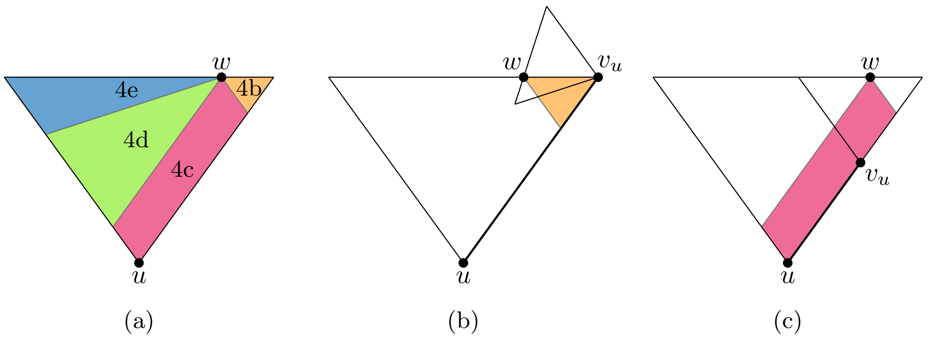


Fig. 5. (a) Four different cases for the position of v_u. (b) The worst-case configuration with w in $C_4^{v_u}$. (c) A configuration with w in $C_0^{v_u}$, after moving v_u onto the right side of C_0^u.

Case 4a. v_w lies in $C_4^u \cap C_3^b$. This situation is illustrated in Figure 4c. Given any initial position of v_w, moving w to the right onto b increases the total path length by increasing $|wv_w|$ while not affecting $|T_{uv_w}|$. Here we use the fact that v_w already lies in C_3^b, otherwise we would not be able to move w onto b while keeping v_w in C_3^w. Now the total path length is maximized by placing v_w on the left corner of T_{wu}. Since this situation is symmetrical to the worst-case situation in Case 1, the theorem holds by the same analysis.

Next, we distinguish four cases for the position of v_u (the closest vertex to u in C_0), illustrated in Figure 5a. We can solve the first two by applying our inductive hypothesis to $T_{v_u w}$.

Case 4b. w lies in $C_4^{v_u}$. To apply our inductive hypothesis, we need to show that $|T_{v_u w}| < |T_{uw}|$. If that is the case, we obtain a path between v_u and w of length at most $c \cdot |T_{v_u w}|$. Since v_u is the closest vertex to u, there is a direct edge from u to v_u, resulting in a path between u and w of length at most $|uv_u| + c \cdot |T_{v_u w}|$.

Given any intial positions for v_u and w, moving w to the left increases $|T_{v_u w}|$ while leaving $|uv_u|$ unchanged. Moving v_u closer to b increases both. Therefore the path length is maximal when w lies on x and v_u lies on b (see Figure 5b). We can express $|T_{v_u w}|$ as follows.

$$|T_{v_u w}| = \frac{\sin \frac{3\pi}{5}}{\sin \frac{3\pi}{10}} \cdot |wv_u| = \frac{\sin \frac{3\pi}{5}}{\sin \frac{3\pi}{10}} \cdot \frac{\sin \frac{\pi}{10}}{\sin \frac{3\pi}{5}} \cdot |T_{uw}| = \frac{1}{2}\left(3 - \sqrt{5}\right) \cdot |T_{uw}|$$

Since $|uv_u| = |T_{uw}|$, the complete path has length at most $c \cdot |T_{uw}|$ for

$$c \geq \frac{1}{1 - \frac{1}{2}\left(3 - \sqrt{5}\right)} = \frac{1}{2}\left(1 + \sqrt{5}\right) \approx 1.618.$$

Case 4c. w lies in $C_0^{v_u}$. Since v_u lies in C_0^u, it is clear that $|T_{v_u w}| < |T_{uw}|$, which allows us to apply our inductive hypothesis. This gives us a path between u and w of length at most $|uv_u| + c \cdot |T_{v_u w}|$. For any initial location of v_u, we can increase the total path length by moving v_u to the right until it hits the side of C_0^u (see Figure 5c), since $|T_{v_u w}|$ stays the same and $|uv_u|$ increases. Once there, we have that $|uv_u| + |T_{v_u w}| = |T_{uw}|$. Since $c > 1$, this immediately implies that $|uv_u| + c \cdot |T_{v_u w}| \leq c \cdot |T_{uw}|$, proving the theorem for this case.

To solve the last two cases, we need to consider the positions of both v_u and v_w.

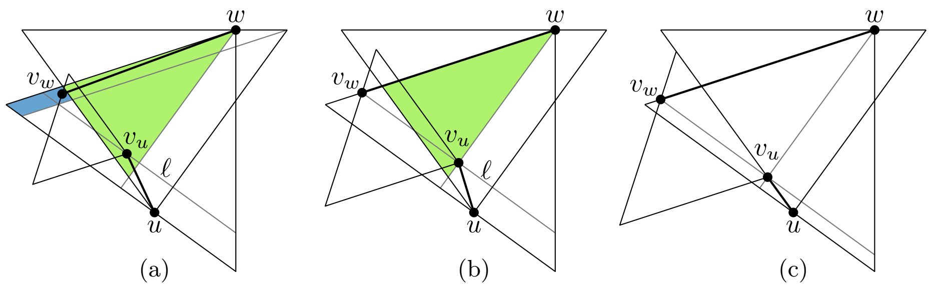


Fig. 6. (a) The regions where v_u (light) and v_w (dark) can lie. (b) The worst case when v_u lies on a given line ℓ. (c) The worst case for a fixed position of w.

Case 4d. w lies in $C_1^{v_u}$ and v_u lies in C_3^w. We would like to apply our inductive hypothesis to $T_{v_u v_w}$, resulting in a path between v_u and v_w of length at most $c \cdot |T_{v_u v_w}|$. The edges (w, v_w) and (u, v_u) complete this to a path between u and w, giving a total length of at most $|uv_u| + c \cdot |T_{v_u v_w}| + |v_w w|$.

First, note that v_u cannot lie in T_{wv_w}, as this region is empty by definition. This means that v_w must lie in $C_4^{v_u}$. We first show that $T_{v_u v_w}$ is always smaller than T_{uw}, which means that we are allowed to use induction. Given any initial position for v_u, consider the line ℓ through v_u, perpendicular to the bisector of C_3 (see Figure 6a). Since v_w cannot be further from w than v_u, the size of $T_{v_u v_w}$ is maximized when v_w lies on the intersection of ℓ and the top boundary of T_{wu}. We can increase $|T_{v_u v_w}|$ further by moving v_u along ℓ until it reaches the bisector of C_3^w (see Figure 6b). Since the top boundary of T_{wu} and the bisector of C_3^w approach each other as they get closer to w, the size of $T_{v_u v_w}$ is maximized when v_u lies on the bottom boundary of T_{wu} (ignoring for now that this would move v_u out of T_{uw}). Now it is clear that $|T_{v_u v_w}| < |T_{uv_w}|$. Since we already established that T_{uv_w} is smaller than T_{uw} in the proof of Theorem 1, this holds for $T_{v_u v_w}$ as well and we can use induction.

All that is left is to bound the total length of the path. Given any initial position of v_u, the path length is maximized when we place v_w at the intersection of ℓ and the top boundary of T_{wu}, as this maximizes both $|T_{v_u v_w}|$ and $|wv_w|$. When we move v_u away from v_w along ℓ by Δ, $|uv_u|$ decreases by at most Δ, while $|T_{v_u v_w}|$ increases by $\sin \frac{3\pi}{5} / \sin \frac{3\pi}{10} \cdot \Delta > \Delta$. Since $c > 1$, this increases the total path length. Therefore the worst case again occurs when v_u lies on the bisector of C_3^w, as depicted in Figure 6b. Moving v_u down along the bisector of T_{wu} by Δ decreases $|uv_u|$ by at most Δ, while increasing $|wv_w|$ by $1/\sin \frac{3\pi}{10} \cdot \Delta > \Delta$ and increasing $|T_{v_u v_w}|$. Therefore this increases the total path length and the worst case occurs when v_u lies on the left boundary of T_{uw} (see Figure 6c).

Finally, consider what happens when we move v_u Δ towards u, while moving w and v_w such that the construction stays intact. This causes w to move to the right. Since v_u, w and the left corner of T_{uw} form an isosceles triangle with apex v_u, this also moves v_u Δ further from w. We saw before that moving v_u away from w increases the size of $T_{v_u v_w}$. Finally, it also increases $|wv_w|$ by $1/\sin \frac{3\pi}{10} \cdot \Delta > \Delta$. Thus, the increase in $|wv_w|$ cancels the decrease in $|uv_u|$ and the total path length increases. Therefore the worst case occurs when v_u lies on u and v_w lies in the corner of T_{wu}, which is symmetric to the worst case of Case 1. Thus the theorem holds by the same analysis.

Case 4e. v_u lies in C_4^w. We split this case into three final subcases, based on the position of v_u. These cases are illustrated in Figure 7a.

Case 4e-1. $|T_{wv_u}| \leq \frac{c-1}{c} \cdot |T_{uw}|$. If T_{wv_u} is small enough, we can apply our inductive hypothesis to obtain a path between v_u and w of length at most $c \cdot |T_{wv_u}|$. Since there is a direct edge between u and v_u, we obtain a path between u and w with length at most $|uv_u| + c \cdot |T_{wv_u}|$. Any edge from u to a point inside T_{uw} has length at most $|T_{uw}|$, so we can bound the length of the path as follows.

$$|uv_u| + c \cdot |T_{wv_u}| \;\leq\; |T_{uw}| + c \cdot \frac{c-1}{c} \cdot |T_{uw}| \;=\; |T_{uw}| + (c-1) \cdot |T_{uw}| \;=\; c \cdot |T_{uw}|$$

In the other two cases, we use induction on $T_{v_w v_u}$ to obtain a path between v_w and v_u of length at most $c \cdot |T_{v_w v_u}|$. The edges (u, v_u) and (w, v_w) complete this to a (self-intersecting) path between u and w. We can bound the length of these edges by the size of the canonical triangle that contains them, as follows.

$$|uv_u| + |wv_w| \;\leq\; |T_{uw}| + |T_{wu}| \;\leq\; |T_{uw}| + \frac{1}{\cos \frac{\pi}{5}} \cdot |T_{uw}| \;=\; \sqrt{5} \cdot |T_{uw}|$$

All that is left now is to bound the size of $T_{v_w v_u}$ and express it in terms of T_{uw}.

Case 4e-2. v_u lies in $C_0^{v_w}$. In this case, the size of $T_{v_w v_u}$ is maximal when v_u lies on the top boundary of T_{uw} and v_w lies at the lowest point in its possible region: the left corner of T_{bu} (see Figure 7b). Now we can express $|T_{v_w v_u}|$ as follows.

$$|T_{v_w v_u}| \;=\; \frac{\sin \frac{\pi}{10}}{\sin \frac{7\pi}{10}} \cdot |bv_w| \;=\; \frac{\sin \frac{\pi}{10}}{\sin \frac{7\pi}{10}} \cdot \frac{1}{\cos \frac{\pi}{5}} \cdot |T_{uw}| \;=\; 2\left(\sqrt{5} - 2\right) \cdot |T_{uw}|$$

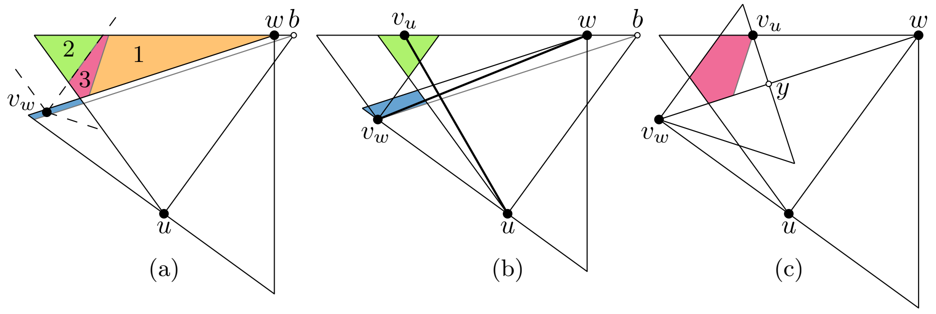


Fig. 7. (a) The three subcases for the position of v_u. (b) The situation that maximizes $T_{v_w v_u}$ when v_u lies in $C_0^{v_w}$. (c) The worst case when v_u lies in $C_1^{v_w}$.

Since $2\left(\sqrt{5}-2\right) < 1$, we can use induction. The total path length is bounded by $c \cdot |T_{uw}|$ for

$$c \;\geq\; \frac{\sqrt{5}}{1-2\left(\sqrt{5}-2\right)} \;=\; 2+\sqrt{5} \;\approx\; 4.236.$$

Case 4e-3. v_u lies in $C_1^{v_w}$. Since $|T_{wv_u}| > \frac{c-1}{c} \cdot |T_{uw}|$, $T_{v_w v_u}$ is maximal when v_w lies on the left corner of T_{wu} and v_u lies on the top boundary of T_{uw}, such that $|T_{wv_u}| = \frac{c-1}{c} \cdot |T_{uw}|$ (see Figure 7c). Let y be the intersection of $T_{v_w v_u}$ and T_{wu}. Note that since v_w lies on the corner of T_{wu}, y is also the midpoint of the side of $T_{v_w v_u}$ opposite v_w. We can express the size of $T_{v_w v_u}$ as follows.

$$|T_{v_w v_u}| \;=\; \frac{|v_w y|}{\cos\frac{\pi}{5}} \;=\; \frac{|wv_w| - |wy|}{\cos\frac{\pi}{5}} \;=\; \frac{\dfrac{|T_{uw}|}{\cos\frac{\pi}{5}} - \cos\frac{\pi}{10} \cdot |wv_u|}{\cos\frac{\pi}{5}}$$

$$= \frac{\dfrac{|T_{uw}|}{\cos\frac{\pi}{5}} - \cos\frac{\pi}{10} \cdot \dfrac{\sin\frac{3\pi}{10}}{\sin\frac{3\pi}{5}} \cdot |T_{wv_u}|}{\cos\frac{\pi}{5}} \;=\; \frac{\dfrac{|T_{uw}|}{\cos\frac{\pi}{5}} - \cos\frac{\pi}{10} \cdot \dfrac{\sin\frac{3\pi}{10}}{\sin\frac{3\pi}{5}} \cdot \dfrac{c-1}{c} \cdot |T_{uw}|}{\cos\frac{\pi}{5}}$$

$$= \left(\frac{1}{c} + 5 - 2\sqrt{5}\right) \cdot |T_{uw}|$$

Thus we can use induction for $c > 1/\left(2\sqrt{5}-4\right) \approx 2.118$ and the total path length can be bounded by $c \cdot |T_{uw}|$ for

$$c \;\geq\; \frac{\sqrt{5}+1}{2\sqrt{5}-4} \;=\; \frac{1}{2}\left(7+3\sqrt{5}\right) \;\approx\; 6.854.$$

□

Using this result, we can compute the exact spanning ratio.

Theorem 2. *The θ_5-graph is a spanner with spanning ratio at most*

$$\sqrt{50+22\sqrt{5}} \approx 9.960.$$

Proof. Given two vertices u and w, we know from Lemma 1 that there is a path between them with length at most $c \cdot \min(|T_{uw}|, |T_{wu}|)$, where $c = 2\left(2+\sqrt{5}\right) \approx 8.472$. This gives an upper bound on the spanning ratio of $c \cdot \min(|T_{uw}|, |T_{wu}|)/|uw|$. We assume without loss of generality that w lies in the right half of C_0^u. Let α be the angle between the bisector of C_0^u and the line uw (see Figure 2b). Using some expressions derived in the proof of Theorem 1, we can express the spanning ratio in terms of α.

$$\frac{c \cdot \min\left(|T_{uw}|, \frac{\cos\left(\frac{\pi}{5}-\alpha\right)}{\cos\alpha} \cdot |T_{uw}|\right)}{\frac{\cos\frac{\pi}{5}}{\cos\alpha} \cdot |T_{uw}|} \quad = \quad \frac{c}{\cos\frac{\pi}{5}} \cdot \min\left(\cos\alpha, \cos\left(\frac{\pi}{5}-\alpha\right)\right)$$

To get an upper bound on the spanning ratio, we need to maximize the minimum of $\cos\alpha$ and $\cos\left(\frac{\pi}{5}-\alpha\right)$. Since for $\alpha \in [0, \pi/5]$, one is increasing and the other is decreasing, this maximum occurs at $\alpha = \pi/10$, where they are equal. Thus, our upper bound becomes

$$\frac{c}{\cos\frac{\pi}{5}} \cdot \cos\frac{\pi}{10} \quad = \quad \sqrt{50+22\sqrt{5}} \quad \approx \quad 9.960.$$

□

4 Lower Bound

In this section, we derive a lower bound on the spanning ratio of the θ_5-graph.

Theorem 3. *The θ_5-graph has spanning ratio at least $\frac{1}{2}(11\sqrt{5}-17) \approx 3.798$.*

Proof. For the lower bound, we present and analyze a path between two vertices that has a large spanning ratio. The path has the following structure (illustrated in Figure 8).

The path can be thought of as being directed from w to u. First, we place w in the right corner of T_{uw}. Then we add a vertex v_1 in the bottom corner of T_{wu}. We repeat this two more times, each time adding a new vertex in the corner of $T_{v_i u}$ furthest

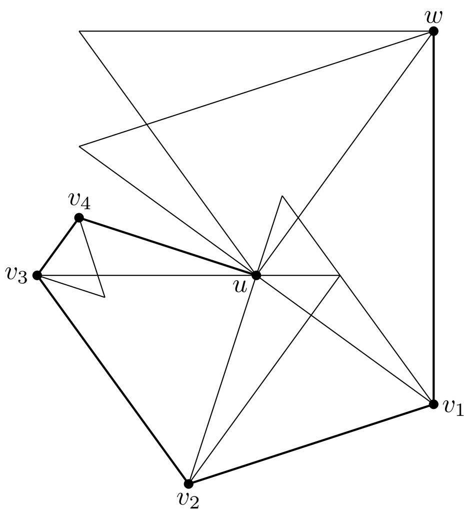


Fig. 8. A path with a large spanning ratio

away from u. The final vertex v_4 is placed on the top boundary of $C_1^{v_3}$, such that u lies in $C_1^{v_4}$. Since we know all the angles involved, we can compute the length of each edge, taking $|uw| = 1$ as baseline.

$$|wv_1| = \frac{1}{\cos\frac{\pi}{5}} \qquad |v_1v_2| = |v_2v_3| = 2\sin\tfrac{\pi}{5}\tan\tfrac{\pi}{5}$$

$$|v_3v_4| = \frac{\sin\frac{\pi}{10}}{\sin\frac{3\pi}{5}}\tan\tfrac{\pi}{5} \qquad |v_4u| = \frac{\sin\frac{3\pi}{10}}{\sin\frac{3\pi}{5}}\tan\tfrac{\pi}{5}$$

Since we set $|uw| = 1$, the spanning ratio is simply $|wv_1| + |v_1v_2| + |v_2v_3| + |v_3v_4| + |v_4u| = \frac{1}{2}(11\sqrt{5} - 17) \approx 3.798$. Note that the θ_5-graph with just these 5 vertices would have a far smaller spanning ratio, as there would be a lot of shortcut edges. However, a graph where this path is the shortest path between two vertices can be found in Appendix A. □

5 Conclusions

We showed that there is a path between every pair of vertices in the θ_5-graph, with length at most $\sqrt{50 + 22\sqrt{5}} \approx 9.960$ times the straight-line distance between them. This is the first constant upper bound on the spanning ratio of the θ_5-graph, proving that it is a geometric spanner. We also presented a θ_5-graph with spanning ratio arbitrarily close to $\frac{1}{2}(11\sqrt{5} - 17) \approx 3.798$, thereby giving a lower bound on the spanning ratio. There is still a significant gap between these bounds, which is caused by the upper bound proof mostly ignoring the main obstacle to improving the lower bound: that every edge requires at least one of its canonical triangles to be empty. Hence we believe that the true spanning ratio is closer to the lower bound.

While our proof for the upper bound on the spanning ratio returns a spanning path between the two vertices, it requires knowledge of the neighbours of both the current vertex and the destination vertex. This means that the proof does not lead to a local routing strategy that can be applied in, say, a wireless setting. This raises the question whether it is possible to route *competitively* on this graph, i.e. to discover a spanning path from one vertex to another by using only information local to the current vertex at each step.

References

1. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press (2007)
2. Clarkson, K.: Approximation algorithms for shortest path motion planning. In: Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC 1987), pp. 56–65 (1987)
3. Keil, J.: Approximating the complete Euclidean graph. In: Karlsson, R., Lingas, A. (eds.) SWAT 1988. LNCS, vol. 318, pp. 208–213. Springer, Heidelberg (1988)

4. Ruppert, J., Seidel, R.: Approximating the d-dimensional complete Euclidean graph. In: Proceedings of the 3rd Canadian Conference on Computational Geometry (CCCG 1991), pp. 207–210 (1991)
5. Bonichon, N., Gavoille, C., Hanusse, N., Ilcinkas, D.: Connections between theta-graphs, Delaunay triangulations, and orthogonal surfaces. In: Thilikos, D.M. (ed.) WG 2010. LNCS, vol. 6410, pp. 266–278. Springer, Heidelberg (2010)
6. El Molla, N.M.: Yao spanners for wireless ad hoc networks. Master's thesis, Villanova University (2009)
7. Barba, L., Bose, P., De Carufel, J.-L., van Renssen, A., Verdonschot, S.: On the stretch factor of the theta-4 graph. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 109–120. Springer, Heidelberg (2013)
8. Rysavy, P.: Wireless broadband and other fixed-wireless systems, `http://www.networkcomputing.com/netdesign/bb1.html` (accessed December 3, 2012)
9. SATEL Oy. What is a radio modem?, `http://www.satel.com/products/what-is-a-radio-modem` (accessed December 3, 2012)
10. Morin, P., Verdonschot, S.: On the average number of edges in theta graphs. CoRR, abs/1304.3402 (2013)

A Lower Bound Construction

#	Action	Shortest path
1	Start with a vertex v_1.	-
2	Add v_2 in C_0^u, such that v_2 is arbitrarily close to the top right corner of $T_{v_1 v_2}$.	$v_1 v_2$
3	Remove edge (v_1, v_2) by adding two vertices, v_3 and v_4, arbitrarily close to the counter-clockwise corners of $T_{v_1 v_2}$ and $T_{v_2 v_1}$.	$v_1 v_4 v_2$
4	Remove edge (v_1, v_4) by adding two vertices, v_5 and v_6, arbitrarily close to the clockwise corner of $T_{v_1 v_4}$ and the counter-clockwise corner of $T_{v_4 v_1}$.	$v_1 v_3 v_2$
5	Remove edge (v_2, v_3) by adding two vertices, v_7 and v_8, arbitrarily close to the clockwise corner of $T_{v_2 v_3}$ and the counter-clockwise corner of $T_{v_3 v_2}$.	$v_1 v_6 v_4 v_2$
6	Remove edge (v_1, v_6) by adding two vertices, v_9 and v_{10}, arbitrarily close to the clockwise corner of $T_{v_1 v_6}$ and the counter-clockwise corner of $T_{v_6 v_1}$.	$v_1 v_5 v_4 v_2$
7	Remove edge (v_4, v_5) by adding two vertices, v_{11} and v_{12}, arbitrarily close to the counter-clockwise corner of $T_{v_4 v_5}$ and the clockwise corner of $T_{v_5 v_4}$.	$v_1 v_5 v_6 v_4 v_2$
8	Remove edge (v_5, v_6) by adding two vertices, v_{13} and v_{14}, arbitrarily close to the counter-clockwise corner of $T_{v_5 v_6}$ and the clockwise corner of $T_{v_6 v_5}$.	$v_1 v_5 v_{14} v_6 v_4 v_2$
9	Remove edge (v_5, v_{14}) by adding two vertices, v_{15} and v_{16}, arbitrarily close to the counter-clockwise corner of $T_{v_5 v_{14}}$ and the clockwise corner of $T_{v_{14} v_5}$.	$v_1 v_5 v_{13} v_6 v_4 v_2$
10	Remove edge (v_6, v_{13}) by adding two vertices, v_{17} and v_{18}, arbitrarily close to the clockwise corner of $T_{v_6 v_{13}}$ and the counter-clockwise corner of $T_{v_{13} v_6}$.	$v_1 v_3 v_8 v_2$
11	Remove edge (v_2, v_8) by adding a vertex v_{19} in the union of, and arbitrarily close to the intersection point of $T_{v_2 v_8}$ and $T_{v_8 v_2}$.	$v_1 v_3 v_7 v_2$
12	Remove edge (v_3, v_7) by adding two vertices, v_{20} and v_{21}, arbitrarily close to the counter-clockwise corner of $T_{v_3 v_7}$ and the clockwise corner of $T_{v_7 v_3}$.	$v_1 v_5 v_{12} v_2$
13	Remove edge (v_2, v_{12}) by adding a vertex v_{22} arbitrarily close to the counter-clockwise corner of $T_{v_2 v_{12}}$.	$v_1 v_{10} v_6 v_4 v_2$
14	Remove edge (v_1, v_{10}) by adding a vertex v_{23} in the union of $T_{v_1 v_{10}}$ and $T_{v_{10} v_1}$, arbitrarily close to the top boundary of $C_1^{v_{10}}$, and such that v_1 lies in $C_1^{v_{23}}$, arbitrarily close to the bottom boundary.	$v_1 v_5 v_{12} v_4 v_2$

(Continued on the next page.)

#	Action	Shortest path
15	Remove edge (v_4, v_{12}) by adding two vertices, v_{24} and v_{25}, arbitrarily close to the counter-clockwise corner of $T_{v_4 v_{12}}$ and the clockwise corner of $T_{v_{12} v_4}$.	$v_1 v_5 v_{13} v_{14} v_6 v_4 v_2$
16	Remove edge (v_{13}, v_{14}) by adding two vertices, v_{26} and v_{27}, arbitrarily close to the clockwise corner of $T_{v_{13} v_{14}}$ and the counter-clockwise corner of $T_{v_{14} v_{13}}$.	$v_1 v_9 v_{18} v_6 v_4 v_2$
17	Remove edge (v_9, v_{18}) by adding two vertices, v_{28} and v_{29}, arbitrarily close to the clockwise corner of $T_{v_9 v_{18}}$ and the counter-clockwise corner of $T_{v_{18} v_9}$.	$v_1 v_5 v_{16} v_{11} v_4 v_2$
18	Remove edge (v_{11}, v_{16}) by adding two vertices, v_{30} and v_{31}, arbitrarily close to the counter-clockwise corner of $T_{v_{11} v_{16}}$ and the clockwise corner of $T_{v_{16} v_{11}}$.	$v_1 v_{23} v_{10} v_6 v_4 v_2$

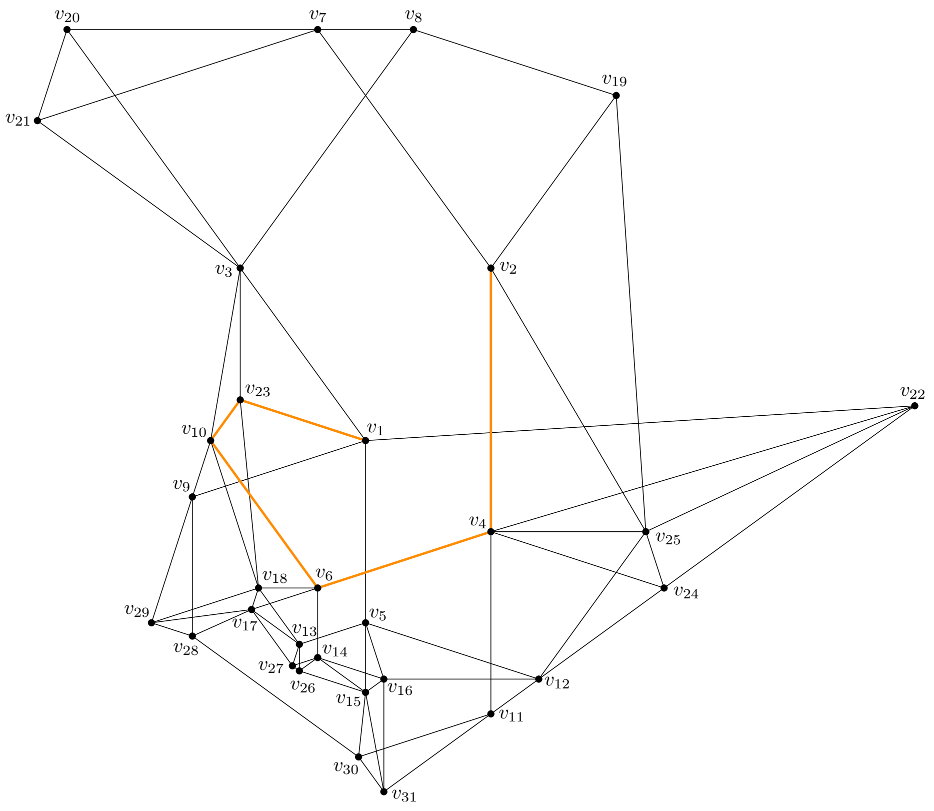


Fig. 9. A θ_5-graph with a spanning ratio that matches the lower bound. The shortest path between v_1 and v_2 is indicated in orange.

Graphs of Edge-Intersecting Non-splitting Paths in a Tree: Towards Hole Representations*

(Extended Abstract)

Arman Boyacı[1], Tınaz Ekim[1], Mordechai Shalom[2], and Shmuel Zaks[3,4,5]

[1] Department of Industrial Engineering, Bogazici University, Istanbul, Turkey
{arman.boyaci,tinaz.ekim}@boun.edu.tr
[2] TelHai Academic College, Upper Galilee, 12210, Israel
cmshalom@telhai.ac.il
[3] Department of Computer Science, Technion, Haifa, Israel
zaks@cs.technion.ac.il
[4] Visiting Professor, Departamento de Ingenieria Telematica, Universidad Carlos III de Madrid, Spain
[5] Visiting Researcher, Institute IMDEA Networks, Madrid, Spain

Abstract. Given a tree and a set $\mathcal{P}$ of non-trivial simple paths on it, $\text{VPT}(\mathcal{P})$ is the VPT graph (i.e. the vertex intersection graph) of $\mathcal{P}$, and $\text{EPT}(\mathcal{P})$ is the EPT graph (i.e. the edge intersection graph) of the paths $\mathcal{P}$ of the tree T. These graphs have been extensively studied in the literature. Given two (edge) intersecting paths in a graph, their *split vertices* is the set of vertices having degree at least 3 in their union. A pair of (edge) intersecting paths is termed *non-splitting* if they do not have split vertices (namely if their union is a path). In this work, we define the graph $\text{ENPT}(\mathcal{P})$ of edge intersecting non-splitting paths of a tree, termed the ENPT graph, as the (edge) graph having a vertex for each path in $\mathcal{P}$, and an edge between every pair of paths that are both edge-intersecting and non-splitting. A graph G is an ENPT graph if there is a tree T and a set of paths $\mathcal{P}$ of T such that $G = \text{ENPT}(\mathcal{P})$, and we say that $\langle T, \mathcal{P} \rangle$ is a *representation* of G. We show that trees, cycles and complete graphs are ENPT graphs. We characterize the representations of chordless ENPT cycles that satisfy a certain assumption. Unlike chordless EPT cycles which have a unique representation, these representations turn out to be multiple and have a more complex structure. Therefore, in order to give this characterization, we assume the EPT graph induced by the vertices of a chordless ENPT cycle is given, and we provide an algorithm that returns the unique representation of this EPT, ENPT pair of graphs. These representations turn out to have a more complex structure than chordless EPT cycles.

* This work was supported in part by the Israel Science Foundation grant No. 1249/08, by TUBITAK PIA BOSPHORUS grant No. 111M303, by BAP grant No. 6461 and by Catedra de Excelencia of Universidad Carlos III de Madrid.

1 Introduction

1.1 Background

Given a tree T and a set $\mathcal{P}$ of non-trivial simple paths in T, the Vertex Intersection Graph of Paths in a Tree (VPT) and the Edge Intersection Graph of Paths in a Tree (EPT) of $\mathcal{P}$ are denoted by $\textsc{Vpt}(\mathcal{P})$ and $\textsc{Ept}(\mathcal{P})$, respectively. Both graphs have $\mathcal{P}$ as vertex set. $\textsc{Vpt}(\mathcal{P})$ (resp. $\textsc{Ept}(\mathcal{P})$) contains an edge between two vertices if the corresponding two paths intersect in at least one vertex (resp. edge). A graph G is VPT (resp. EPT) if there exist a tree T and a set $\mathcal{P}$ of non-trivial simple paths in T such that G is isomorphic to $\textsc{Vpt}(\mathcal{P})$ (resp. $\textsc{Ept}(\mathcal{P})$). In this case we say that $\langle T, \mathcal{P} \rangle$ is a VPT (resp. EPT) representation of G.

In this work we focus on edge intersections of paths, therefore whenever we are concerned with intersection of paths we omit the word "edge" and simply write that two paths *intersect*. We define a new class of graphs, called the graphs of edge intersecting and non-splitting paths of a tree (ENPT). Given a representation $\langle T, \mathcal{P} \rangle$ as described above, the related ENPT graph, denoted by $\textsc{Enpt}(\mathcal{P})$, has a vertex v for each path P_v of $\mathcal{P}$ and two vertices u, v of $\textsc{Enpt}(\mathcal{P})$ are adjacent if the paths P_u and P_v intersect and do not split (that is, their union is a path). A graph G is an ENPT graph if there is a tree T and a set of paths $\mathcal{P}$ of T such that G is isomorphic to $\textsc{Enpt}(\mathcal{P})$. We study the properties of this class ENPT.

We note that when T is a path $\textsc{Ept}(\mathcal{P}) = \textsc{Enpt}(\mathcal{P})$ is an Interval Graph. Therefore the class ENPT includes all Interval Graphs. Our aim is to study the structure of ENPT in order to classify them in the hierarchy of edge intersection graphs of paths in a tree.

EPT and VPT graphs have applications in communication networks. Assume that we model a communication network as a tree T and the message routes to be delivered in this communication network as paths on T. Two paths conflict if they both require to use the same link (vertex). This conflict model is equivalent to an EPT (a VPT) graph. Suppose we try to find a schedule for the messages such that no two messages sharing a link (vertex) are scheduled in the same time interval. Then a vertex coloring of the EPT (VPT) graph corresponds to a feasible schedule on this network. The motivation for the split condition for the paths can be summarized as follows: In optical networks, a router is an equipment responsible to route a message to a direction determined by its wavelength. So that two messages, corresponding to two splitting paths, can be correctly routed to different directions, they should be assigned two different wavelengths (see [REF] for more information).

EPT and VPT graphs have been extensively studied in the literature. Although VPT graphs can be characterized by a fixed number of forbidden subgraphs [13], it is shown that EPT graph recognition is NP-complete [11]. Edge intersection and vertex intersection give rise to identical graph classes in the case of paths in a line and in the case of subtrees of a tree. However, VPT graphs and EPT graphs are incomparable in general; neither VPT nor EPT contains the other. Main optimization and decision problems such as recognition [5], the maximum clique [6], the minimum vertex coloring [9] and the maximum stable set problems [14] are polynomial-time solvable in VPT whereas recognition and minimum vertex coloring problems remain NP-complete in EPT graphs [10]. In contrast, one can solve in polynomial time the maximum clique [11] and the maximum stable set [15] problems in EPT graphs.

After these works on EPT and VPT graphs in the early 80's, this topic did not get focus until very recently. Current research on intersection graphs is concentrated on the comparison of various intersection graphs of paths in a tree and their relation to chordal and weakly chordal graphs [7,12]. Also, some tolerance model is studied via k-edge intersection graphs where two vertices are adjacent if their corresponding paths intersect on at least k edges [8]. Besides, several recent papers consider the edge intersection graphs of paths on a grid (e.g [1]).

1.2 Our Contribution

In this work we define the new family of ENPT graphs, and investigate its basic properties. We first study possible ENPT representations of some basic structures such as trees, cliques and holes. It should be noted that cliques play a crucial role in showing the NP-hardness of EPT graph recognition [10]. On the other hand, some forbidden subgraphs are determined in [11] using the fact that chordless cycles have a unique EPT representation, called a pie. It turns out that in ENPT graphs, representations of chordless cycles have a much more complicated structure, yielding several possible representations. In fact, given a chordless cycle C, several ENPT representations $\langle T, \mathcal{P} \rangle$ such that $\textsc{Enpt}(\mathcal{P})$ is isomorphic to C but $\textsc{Ept}(\mathcal{P})$ are non-isomorphic to each other are possible (see Figure 2).

Consider the pair (G, C) where G is a graph and C is a Hamiltonian cycle of G. We restrict our attention to the determination of a representation $\langle T, \mathcal{P} \rangle$ such that $\textsc{Ept}(\mathcal{P}) = G$ and $\textsc{Enpt}(\mathcal{P}) = C$. In this case we will say that $\langle T, \mathcal{P} \rangle$ is a representation of (G, C).

In Section 2 we give definitions, preliminaries and we provide ENPT representations of basic graphs such as trees, cliques and cycles. We also characterize all the ENPT representations of cliques. In Section 3 we obtain basic results regarding ENPT graphs, their relationship with EPT graphs and their representation. We then define the contraction operation, which is basically replacing two paths with their union provided that this union is a path. In Section 4 we introduce three assumptions and we characterize the representations of ENPT holes, i.e. representations $\langle T, \mathcal{P} \rangle$ for pairs (G, C), where C is a Hamiltonian cycle of G, such that $\textsc{Ept}(\mathcal{P}) = G$ and $\textsc{Enpt}(\mathcal{P}) = C$, satisfying these assumptions. In Section 5 we relax two out of these three assumptions, and extend the result of Section 4. Most of the proofs are either sketched or omitted in this Extended Abstract. The complete proofs appear in [2], except for Section 5 whose details can be found in [3].

2 Preliminaries and Basic Results

In this section we give definitions, present known related results, and develop basic results. The section is organized as follows: Section 2.1 is devoted to basic definitions, in Section 2.2 we present known results on EPT graphs and in Section 2.3 we present some basic ENPT graphs.

2.1 Definitions

General Notation: Given a graph G and a vertex v of G, we denote by $d_G(v)$ the degree of v in G. A vertex is called a *leaf* (resp. *intermediate vertex*, *junction*) if $d_G(v) = 1$ (resp. $= 2$, ≥ 3). Whenever there is no ambiguity we omit the subscript G and write $d(v)$. Given a graph G, $\bar{V} \subseteq V(G)$ and $\bar{E} \subseteq E(G)$ we denote by $G[\bar{V}]$ and $G[\bar{E}]$ the subgraphs of G induced by $\bar{V}$ and by $\bar{E}$, respectively. The *union* of two graphs G, G' is the graph $G \cup G' \overset{def}{=} (V(G) \cup V(G'), E(G) \cup E(G'))$. The *join* $G + G'$ of two disjoint graphs G, G' is the graph $G \cup G'$ together with all the edges joining $V(G)$ and $V(G')$, i.e. $G + G' \overset{def}{=} (V(G) \cup V(G'), E(G) \cup E(G') \cup (V(G) \times V(G')))$. Given a (simple) graph G and $e \in E(G)$, we denote by $G_{/e}$ the (simple) graph obtained by contracting the edge $e = \{p, q\}$ of G, i.e. by coinciding the two endpoints of e to a single vertex $p.q$ and removing self loops and parallel edges. For any two vertices u, v of a tree T we denote by $p_T(u, v)$ the unique path between u and v in T. We denote the set of all positive integers at most k as $[k]$.

Intersections and union of paths: Given two paths P, P' in a graph, $P \parallel P'$ means that P and P' are non-intersecting, i.e. *edge-disjoint*. The *split vertices* of P and P' is the set of junctions in their union $P \cup P'$ and is denoted by *split*(P, P'). Whenever P and P' intersect and *split*$(P, P') = \emptyset$ we say that P and P' are *non-splitting* and denote this by $P \sim P'$. In this case $P \cup P'$ is a path or a cycle. When P and P' intersect and *split*$(P, P') \neq \emptyset$ we say that they are *splitting* and denote this by $P \nsim P'$. Clearly, for any two paths P and P' exactly one of the following holds: $P \parallel P'$, $P \sim P'$, $P \nsim P'$. When the graph G is a tree, the union $P \cup P'$ of two intersecting paths P, P' of G is a tree with at most two junctions, i.e. $|split(P, P')| \leq 2$ and $P \cup P'$ is a path whenever $P \sim P'$.

The VPT, EPT and ENPT graphs: Let $\mathcal{P}$ be a set of paths in a tree T. The graphs $\textsc{Vpt}(\mathcal{P}), \textsc{Ept}(\mathcal{P})$ and $\textsc{Enpt}(\mathcal{P})$ are graphs such that $V(\textsc{Enpt}(\mathcal{P})) = V(\textsc{Ept}(\mathcal{P})) = V(\textsc{Vpt}(\mathcal{P})) = \{p | P_p \in \mathcal{P})\}$. Given two distinct paths $P_p, P_q \in \mathcal{P}$, $\{p, q\}$ is an edge of $\textsc{Enpt}(\mathcal{P})$ if $P_p \sim P_q$, and $\{p, q\}$ is an edge of $\textsc{Ept}(\mathcal{P})$ (resp. $\textsc{Vpt}(\mathcal{P})$) if P_p and P_q have a common edge (resp. vertex) in T. It follows that:

Remark 1. $E(\textsc{Enpt}(\mathcal{P})) \subseteq E(\textsc{Ept}(\mathcal{P})) \subseteq E(\textsc{Vpt}(\mathcal{P}))$.

Two graphs G and G' such that $V(G) = V(G')$ and $E(G') \subseteq E(G)$ are termed a *pair (of graphs)* denoted as (G, G'). If $\textsc{Ept}(\mathcal{P}) = G$ (resp. $\textsc{Enpt}(\mathcal{P}) = G$) we say that $\langle T, \mathcal{P} \rangle$ is an EPT (resp. ENPT) representation for G. If $\textsc{Ept}(\mathcal{P}) = G$ and $\textsc{Enpt}(\mathcal{P}) = G'$ we say that $\langle T, \mathcal{P} \rangle$ is a representation for the pair (G, G'). Given a pair (G, G') the sub-pair induced by $\bar{V} \subseteq V(G)$ is the pair $(G[\bar{V}], G'[\bar{V}])$. Clearly, any representation of a pair induces representations for its induced sub-pairs.

Throughout this work, in all figures, the edges of the tree T of a representation $\langle T, \mathcal{P} \rangle$ are drawn as solid edges whereas the paths on the tree are shown by dashed edges. Similarly, edges of $\textsc{Enpt}(\mathcal{P})$ are drawn with solid or blue lines whereas edges in $E(\textsc{Ept}(\mathcal{P})) \setminus E(\textsc{Enpt}(\mathcal{P}))$ are dashed or red. We sometimes refer to them as blue and red edges, respectively. For an edge $e = \{p, q\}$ we use *split*(e) as a shorthand for *split*(P_p, P_q). We note that e is a red edge if and only if *split*$(e) \neq \emptyset$.

Cycles, Chords, Holes, Outerplanar graphs, Trees: Given a graph G and a cycle C of it, a *chord* of C in G is an edge of $E(G) \setminus E(C)$ connecting two vertices of $V(C)$. The *length* of a chord connecting the vertices i,j is the length of a shortest path between i and j on C. C is a *hole* (chordless cycle) of G if G does not contain any chord of C. This is equivalent to saying that the subgraph $G[V(C)]$ of G induced by the vertices of C is a cycle. For this reason a chordless cycle is also called an *induced* cycle.

An *outerplanar* graph is a planar graph that can be embedded in the plane such that all its vertices are on the unbounded face of the embedding. An outerplanar graph is Hamiltonian if and only if it is biconnected; in this case the unbounded face forms the unique Hamiltonian cycle. The *weak dual* graph of a planar graph G is the graph obtained from its dual graph by removing the vertex corresponding to the unbounded face of G. The weak dual graph of an outerplanar graph is a forest, and in particular the weak dual graph of a Hamiltonian outerplanar graph is a tree [4].

2.2 Preliminaries on EPT Graphs

We now present definitions and results from [11]. A *pie* of a representation $\langle T, \mathcal{P} \rangle$ of an EPT graph is an induced star $K_{1,k}$ of T with k leaves $v_0, v_1, \ldots, v_{k-1} \in V(T)$, and k paths $P_0, P_1, \ldots P_{k-1} \in \mathcal{P}$, such that for every $0 \leq i \leq k-1$ both v_i and $v_{(i+1) \textbf{ mod } k}$ are vertices of P_i. We term the central vertex of the star as the *center* of the pie. It is easy to see that the EPT graph of a pie with k leaves is the hole C_k on k vertices. Moreover, this is the only possible EPT representation of C_k when $k \geq 4$:

Theorem 1. *[11] If an* EPT *graph contains a hole with $k \geq 4$ vertices, then every representation of it contains a pie with k paths.*

Let $\mathcal{P}_e \overset{def}{=} \{p \in \mathcal{P} | \; e \in p\}$ be the set of paths in $\mathcal{P}$ containing the edge e. A star $K_{1,3}$ is termed a *claw*. For a claw K of a tree T, $\mathcal{P}[K] \overset{def}{=} \{p \in \mathcal{P} | \; p \text{ uses two edges of } K\}$. It is easy to see that both $\textsc{Ept}(\mathcal{P}_e)$ and $\textsc{Ept}(\mathcal{P}[K])$ are cliques. These cliques are termed *edge clique* and *claw clique*, respectively. Moreover, these are the only possible representations of cliques.

Theorem 2. *[11] Any maximal clique of an* EPT *graph with representation $\langle T, \mathcal{P} \rangle$ corresponds to a subcollection $\mathcal{P}_e$ of paths for some edge e of T, or to a subcollection $\mathcal{P}[K]$ of paths for some claw K of T.*

2.3 Some ENPT Graphs

In this section we show that trees, cycles and cliques are ENPT graphs, and give a complete characterization of the ENPT representations of cliques:

Lemma 1. *Every clique K of* ENPT($\mathcal{P}$) *corresponds to an edge clique of* EPT($\mathcal{P}$)*, such that the union of the paths representing K is a path.*

A direct consequence of Lemma 1 is that the maximum clique problem in ENPT graphs can be solved in polynomial time. As there are at most $O(V(T)^3)$ maximal cliques in G, a maximum clique can be found using a clique enumeration algorithm, e.g. [16].

Lemma 2. *Every tree is an* ENPT *graph.*

Let T be a tree with k leaves and $\pi = (\pi_0, \ldots, \pi_{k-1})$ a cyclic permutation of the leaves. The *tour* (T, π) is the following set of $2k$ paths: (T, π) contains k *long* paths, each of which connecting two consecutive leaves $\pi_i, \pi_{i+1 \textbf{ mod } k}$. (T, π) contains k *short* paths, each of which connecting a leaf π_i and its unique neighbor in T (see Figure 1-c). Note that ENPT((T, π)) is a cycle.

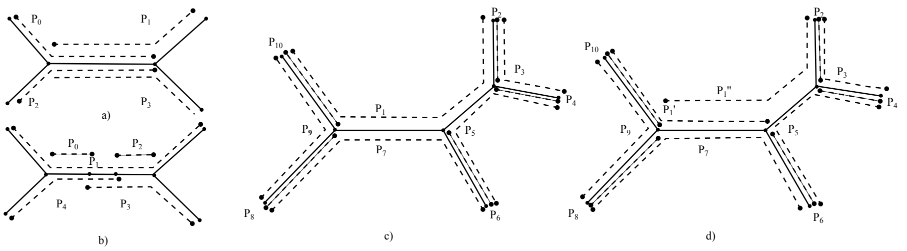


Fig. 1. a) A minimal representation of C_4 b) A minimal representation of C_5 c) A tour representation of the even hole C_{10}, d) A representation of the odd hole C_{11}

A *planar embedding* of a tour is a planar embedding of the underlying tree such that any two paths of the tour do not cross each other. A tour is *planar* if there exists a planar embedding of it. The tour in Figure 1-c is a planar embedding of a tour. Note that a tour (T, π) is planar if and only if π corresponds to the order in which the leaves are encountered by some DFS traversal of T.

Lemma 3. *Every cycle C_k is an* ENPT *graph.*

Proof. $C_3 = K_3$ is an ENPT graph by Lemma 1. As for C_4 and C_5, possible ENPT representations are shown in Figure 1-(a,b), respectively. Any even hole C_{2k}, $(k \geq 3)$ is an ENPT graph. Indeed, for any tree T with k leaves, and a cyclic permutation π of its leaves, the tour (T, π) constitutes an ENPT representation of C_{2k}. Any odd hole C_{2k+1}, $(k \geq 3)$ is an ENPT graph. Let T be a tree with k leaves. Split any long path of some tour (T, π) into two intersecting sub-paths such that no chord is created (if necessary subdivide an edge of the tree into two edges) (see Figure 1-d). The set of $2k + 1$ paths obtained in this way constitutes an ENPT representation for C_{2k+1}. □

3 Representations of EPT, ENPT Pairs: Basic Properties

In this section we develop the basic tools towards our goal of characterizing representations of EPT, ENPT pairs. We define an equivalence relation and a partial order on representations. In this work, we focus on finding representations that are minimal with respect to this partial order. We define the contraction operation on pairs, and the union operation on representations. The contraction operation is a restricted variant of graph contraction operation that operates on both graphs of a pair. The union operation is the operation of replacing two paths by their union whenever possible.

Equivalent and Minimal Representations: We say that the representations $\langle T_1, \mathcal{P}_1 \rangle$ and $\langle T_2, \mathcal{P}_2 \rangle$ are *equivalent*, and denote it by $\langle T_1, \mathcal{P}_1 \rangle \approxeq \langle T_2, \mathcal{P}_2 \rangle$, if their corresponding EPT and ENPT graphs are isomorphic under the same isomorphism (in other words, if they constitute representations of the same pair of graphs (G, G')).

We write $\langle T_2, \mathcal{P}_2 \rangle \lesssim \langle T_2, \mathcal{P}_2 \rangle$ if $\langle T_2, \mathcal{P}_2 \rangle$ can be obtained from $\langle T_1, \mathcal{P}_1 \rangle$ by successive application of (one of) the following *minifying* operations: a) Contraction of an edge e of T_1 (and of all the paths in $\mathcal{P}_1$ using e), b) Removal of an initial edge (*tail*) of a path in $\mathcal{P}_1$. $\langle T, \mathcal{P} \rangle$ is a *minimal* representation if it is minimal in the partial order $\lesssim$ restricted to the representations representing the same pair. Throughout the work we aim at characterizing minimal representations.

EPT Holes: The ENPT graph of a pie is an independent set. Therefore

Remark 2. A hole of size at least 4 of an EPT graph does not contain blue (i.e. ENPT) edges.

Combining with Theorem 1, we obtain the following characterization of pairs (C_k, G'):

- $k > 3$. In this case C_k is represented by a pie. Therefore G' is an independent set. In other words, C_k consists of red edges. We term such a hole a red hole.
- $k = 3$ and C_k consists of red edges (G' is an independent set). We term such a hole a red triangle.
- $k = 3$ and C_k contains exactly one ENPT (blue) edge ($G' = P_1 \cup P_2$). We term such a hole a *BRR* triangle, and its representation is an edge clique.
- $k = 3$ and C_k contains two ENPT (blue) edges ($G' = P_3$). We term such a hole a *BBR* triangle, and its representation is an edge clique.
- $k = 3$ and C_k consists of blue edges ($G' = C_3$). We term such a hole a blue triangle.

EPT Contraction: Let $\langle T, \mathcal{P} \rangle$ be a representation and $P_p, P_q \in \mathcal{P}$ such that $P_p \sim P_q$. We denote by $\langle T, \mathcal{P} \rangle_{/P_p, P_q}$ the representation that is obtained from $\langle T, \mathcal{P} \rangle$ by replacing the two paths P_p, P_q by the path $P_p \cup P_q$, i.e. $\langle T, \mathcal{P} \rangle_{/P_p, P_q} \overset{def}{=} \langle T, \mathcal{P} \setminus \{P_p, P_q\} \cup \{P_p \cup P_q\} \rangle$. We term this operation a *union*, and note the following important property of split vertices with respect to the union operation, and the following Lemma that it implies.

Remark 3. For every $P_p, P_q, P_r \in \mathcal{P}$ such that $P_p \sim P_q$, $\mathit{split}(P_p \cup P_q, P_r) = \mathit{split}(P_p, P_r) \cup \mathit{split}(P_q, P_r)$.

Lemma 4. *Let $\langle T, \mathcal{P} \rangle$ be a representation for the pair (G, G'), and let $e = \{p, q\} \in E(G')$. Then $G_{/e}$ is an* EPT *graph. Moreover $G_{/e} = \textsc{Ept}(\langle T, \mathcal{P} \rangle_{/P_p, P_q})$.*

We now extend the definition of the contraction operation to pairs. Based on Observation 3, the contraction of an ENPT edge does not preserve ENPT edges. More concretely, let P_p,P_q and $P_{q'}$ such that $P_p \sim P_q$, $P_p \sim P_{q'}$ and $P_q \nsim P_{q'}$. Then ${G'}_{/p,q}$ is not necessarily isomorphic to $\textsc{Enpt}(\langle T, \mathcal{P} \rangle_{/P_p, P_q})$ as $\{q', p.q\} \notin E(\textsc{Enpt}(\langle T, \mathcal{P} \rangle_{/P_p, P_q}))$. Let (G, G') be a pair and $e \in E(G')$. If for every edge $e' \in E(G')$ incident to e, the edge $e'' = e \triangle e'$ (forming a triangle together with e and e') is not an edge of G then $(G, G')_{/e} \overset{def}{=} (G_{/e}, {G'}_{/e})$, otherwise $(G, G')_{/e}$ is undefined. Whenever $(G, G')_{/e}$ is defined we say that (G, G') is *contractible* on e, or that e is *contractible.* A pair (G, G') is *contractible* if it contains at least one contractible edge. Clearly, (G, G') is non-contractible if and only if every edge of G' is contained in at least one BBR triangle.

4 Representation of ENPT Holes

The ENPT representations of C_3 is characterized by Lemma 1. Therefore we assume $n > 3$, which implies that (G, C_n) does not contain blue triangles. Moreover, in this section we confine ourselves to pairs (G, C_n) and representations $\langle T, \mathcal{P} \rangle$ satisfying the following three assumptions:

$(P1)$: (G, C_n) is not contractible.
$(P2)$: (G, C_n) is (K_4, P_4)-free, i.e., it does not contain an induced sub-pair isomorphic to a (K_4, P_4).
$(P3)$: Every red triangle of (G, C_n) is a claw clique, i.e. corresponds to a pie of $\langle T, \mathcal{P} \rangle$.

Assumptions $(P1), (P2)$ are relaxed in Section 5. Note that $(P1)$ and $(P2)$ are assumptions about the pair (G, C) and $(P3)$ is an assumption about the representation $\langle T, \mathcal{P} \rangle$. We say that $(P3)$ holds for a pair (G, C) if it has a representation $\langle T, \mathcal{P} \rangle$ satisfying $(P3)$.

W.l.o.g. let $V(G) = V(C_n) = \{0, 1, \ldots, n-1\}$ where the numbering of the vertices follows their order in C. Arithmetic operations on vertex numbers are modulo n. The corresponding set of paths is $\mathcal{P} = \{P_0, \ldots, P_{n-1}\}$.

C_4 is exceptional because all its representations satisfy assumptions $(P1-3)$, but some of our results fail to hold. The following Lemma 5 characterizes the representations of (G, C_4).

Lemma 5. *(i) All the representations of (G, C_4) satisfy assumptions $(P1-3)$, (ii) G is one of the two graphs in Figure 2, and (iii) each one of these two graphs has a unique minimal representation (also depicted in Figure 2).*

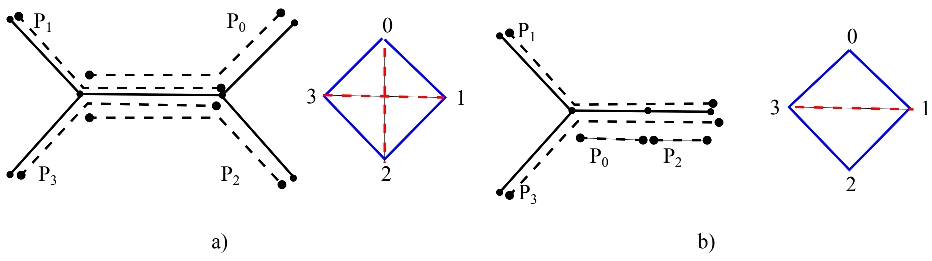


Fig. 2. Two possible ENPT representations of C_4 corresponding to different (G, C_4) pairs

Weak Dual Trees: We extend the definition of the weak dual tree of Hamiltonian outerplanar graphs to any Hamiltonian graph as follows. Given a pair (G, C) where C is a Hamiltonian cycle of G, a weak dual tree of (G, C) is the weak dual tree $\mathcal{W}(G, C)$ of an arbitrary Hamiltonian maximal outerplanar subgraph $\mathcal{O}(G, C)$ of G. $\mathcal{O}(G, C)$ can be built by starting from C and adding to it arbitrarily chosen chords from G as long as such chords exists and the resulting graph is planar.

Vertices of $\mathcal{W}(G, C)$ correspond to faces of $\mathcal{O}(G, C)$, and the faces of $\mathcal{O}(G, C)$ correspond to holes of G. The degree of a vertex of $\mathcal{W}(G, C)$ is the number of red edges in the corresponding face of $\mathcal{O}(G, C)$. To emphasize the difference, for an outerplanar graph G we will refer to *the* weak dual tree of G, whereas for a (not necessarily outerplanar) graph G we will refer to *a* weak dual tree of G.

Edges of $\mathcal{W}(G, C)$ correspond to red edges of $\mathcal{O}(G, C)$. The degree of a vertex of $\mathcal{W}(G, C)$ is the number of red edges in the corresponding face of $\mathcal{O}(G, C)$. Therefore leaves (resp. intermediate vertices, junctions) of $\mathcal{W}(G, C)$ correspond to BBR triangles (resp. BRR triangles, red holes) of (G, C). $|V(G)| = |V(C)| = |E(C)| = 2\ell + i$ where ℓ is the number of leaves of $\mathcal{W}(G, C)$ and i is the number of its intermediate vertices.

Lemma 6. *Let $n > 4$ and (G, C_n) be a pair satisfying $(P1 - 3)$. Then every edge of C_n is in exactly one BBR triangle.*

Lemma 7. *Let (G, C) be a pair satisfying $(P2), (P3)$ and let $\mathcal{W}(G, C)$ be a weak dual tree of (G, C). (i) There is a bijection between the contractible edges of (G, C) and the intermediate vertices of $\mathcal{W}(G, C)$. (ii) The tree obtained from $\mathcal{W}(G, C)$ by smoothing out the intermediate vertex corresponding to a contractible edge e is a weak dual tree of $(G, C)_{/e}$.*

We note that Lemma 6 does not hold for $n = 4$. However the following corollary of lemmata 6 and 7 holds for every n.

Corollary 1. *If (G, C) is a pair satisfying $(P1 - 3)$ with C isomorphic to C_n, then: (i) $\mathcal{W}(G, C)$ does not have intermediate vertices, (ii) n is even and $\mathcal{W}(G, C)$ has $n/2$ leaves. (iii) $\mathcal{W}(G, C)$ is a path if and only if $n = 4$.*

The Minimal Representation: Algorithm 1 gets a pair (G, C) satisfying assumptions $(P1), (P2)$ where C is a (Hamiltonian) cycle of G, and returns a planar tour that is the unique minimal representation of (G, C) satisfying $(P3)$.

The algorithm finds a planar tour of a weak dual tree $\mathcal{W}(G,C)$, and verifies that the solution found is valid before returning it, otherwise it returns that no solution exists. The representation $\langle \bar{T}, \bar{\mathcal{P}} \rangle$ calculated by the algorithm is a planar tour, that clearly satisfies $(P3)$. If (G,C) has no representation satisfying $(P3)$, then the algorithm detects this at line 10 and returns correctly that there is no solution. Therefore, we assume that (G,C) has at least one representation satisfying $(P3)$. The correctness is implied by the following lemma.

Algorithm 1. BUILDPLANARTOUR(G,C)

```
Require: $|V(G)| \geq 5$, $(G,C)$ satisfies $(P1),(P2)$
 1: $\bar{T} \leftarrow \mathcal{W}(G,C)$.                ▷ Corresponding to $\mathcal{O}(G,C)$
    Build the planar tour:
 2: Let $\{v_0, v_1, \ldots, v_{k-1}\}$ be the leaves of $\bar{T}$ ordered by the DFS traversal of $\bar{T}$
 3: corresponding to the planar embedding suggested by $\mathcal{O}(G,C)$.
 4: Let $L_i = p_{\bar{T}}(v_i, v_{i+1 \textbf{ mod } k})$
 5: Let $S_i$ be the path of length 1 starting at $v_i$.
 6: $\bar{\mathcal{P}}_L \leftarrow \{L_i | \ 0 \leq i \leq n-1\}$, $\bar{\mathcal{P}}_S \leftarrow \{S_i | \ 0 \leq i \leq n-1\}$.
 7: Let $\bar{P}_i = \begin{cases} L_{i/2} & \text{if } i \text{ is even} \\ S_{\lfloor i/2 \rfloor} & \text{otherwise} \end{cases}$
 8: $\bar{\mathcal{P}} \leftarrow \left\{\bar{P}_i | \ 0 \leq i \leq 2n-1\right\}$                ▷ $= \bar{\mathcal{P}}_L \cup \bar{\mathcal{P}}_S$
 9:
10: if EPT$(\bar{\mathcal{P}}) = G$ then return $\langle \bar{T}, \bar{\mathcal{P}} \rangle$
11: else return "NO SOLUTION"
12: end if
```

Lemma 8. *Let (G,C) be a pair satisfying $(P1-3)$, $\langle T, \mathcal{P} \rangle$ a representation of (G,C) satisfying $(P3)$ and $\langle \bar{T}, \bar{\mathcal{P}} \rangle$ the representation returned by the algorithm. Then $\langle \bar{T}, \bar{\mathcal{P}} \rangle \cong \langle T, \mathcal{P} \rangle$ and $\langle \bar{T}, \bar{\mathcal{P}} \rangle \lesssim \langle T, \mathcal{P} \rangle$.*

Sketch of proof: For a representation $\langle T, \mathcal{P} \rangle$ of (G,C) that satisfies $(P3)$ we define a mapping $f : V(\bar{T}) \mapsto V(T)$ that maps junctions to junctions. The basic property of this mapping is that for a given vertex u of $\mathcal{W}(G,C)$, and every vertex i on the corresponding face of $\mathcal{O}(G,C)$, the vertex $f(u)$ is on the path P_i.

A junction u of $\bar{T}(= \mathcal{W}(G,C))$ corresponds to a face of $\mathcal{O}(G,C)$ which in turn corresponds to a hole of G corresponding to a pie of $\langle T, \mathcal{P} \rangle$. $f(u)$ is the center of this pie. A leaf v of $\bar{T}$ is adjacent to a junction u. v corresponds to a BBR triangle $\{i-1, i, i+1\}$ of $\mathcal{O}(G,C)$. Then $\{i-1, i+1\}$ is a red edge of G belonging to the face in $\mathcal{O}(G,C)$ corresponding to a pie centered at $f(u)$. Therefore P_{i-1} and P_{i+1} are two consecutive paths of this pie, i.e. $f(u) \in$ *split*(P_{i-1}, P_{i+1}) and the paths P_{i-1}, P_{i+1} intersect on some path P of T starting at $f(u)$. The path P_i satisfies $P_i \sim P_{i-1}$ and $P_i \sim P_{i+1}$, therefore it intersects the path P. $f(v)$ is the most distant vertex from $f(u)$ on this intersection.

We prove that f preserves the topology of the tree. We then define a set of paths $\mathcal{P}^*$ of the minimum subtree T^* of T containing all the vertices $\left\{f(u) | u \in \bar{T}\right\}$ such that $\langle T^*, \mathcal{P}^* \rangle$ is equivalent to $\langle \bar{T}, \bar{\mathcal{P}} \rangle$ and $\langle \bar{T}, \bar{\mathcal{P}} \rangle \lesssim \langle T^*, \mathcal{P}^* \rangle \lesssim \langle T, \mathcal{P} \rangle$. □

We are now ready to prove our main result

Theorem 3. *If $n > 4$ the following statements are equivalent:*
(i) (G, C_n) satisfies assumptions $(P1-3)$.
(ii) (G, C_n) has a unique minimal representation satisfying $(P3)$ which is a planar tour of a weak dual tree of G.
(iii) G is Hamiltonian outerplanar and every face adjacent to the unbounded face F is a triangle having two edges in common with F, (i.e. a BBR triangle).

Proof. (i) ⇒ (ii) Implied by Lemma 8.
(ii) ⇒ (iii) Consider a planar tour representation $\langle T, \mathcal{P}\rangle$. We show that $\mathrm{EPT}(\mathcal{P})$ is a Hamiltonian outerplanar graph. As $\mathcal{P}$ is a tour, $\mathrm{ENPT}(\mathcal{P})$ is a ring, therefore $\mathrm{EPT}(\mathcal{P})$ is Hamiltonian. It is not hard to prove that no chords of this cycle are crossing.

(iii) ⇒ (i) Assume that G is outerplanar with faces adjacent to the unbounded face being BBR triangles. Consequently G is K_4-free, thus satisfies $(P2)$. Moreover every edge of C is in (exactly) one BBR triangle, therefore $(P1)$ holds. The planar tour of the weak dual tree of G is a representation of (G, C). This representation satisfies $(P3)$ because every edge clique of size 3 contains one short path whose incident edges are blue. □

5 Extensions

The details of the results presented in this section are given in [3]. When we relax assumption $(P1)$ then the unique minimal representation can be obtained by slightly modifying the planar tour as follows. Let us call *breaking apart* the inverse of a sequence of union operations that create one path. A *broken tour* is a representation obtained by breaking apart long paths of a tour.

Theorem 4. *[3] Let (G, C) be a pair satisfying $(P2), (P3)$. The unique minimal representation $\langle T', \mathcal{P}'\rangle$ of (G, C) satisfying $(P3)$ is a broken planar tour. Moreover $\langle T', \mathcal{P}'\rangle$ can be calculated in polynomial-time.*

We further relax assumption $(P2)$ and we replace all sub-pairs (K_4, P_4) by BBR triangles. The unique minimal representation of the modified pair is a broken planar tour by Theorem 4. We replace the short paths (corresponding to the inserted BBR triangles) by two paths in an appropriate way. We call such a representation a *broken planar tour with cherries.*

Theorem 5. *[3] The minimum representation $\langle T, \mathcal{P}\rangle$ satisfying $(P3)$ of a pair (G, C) is an broken planar tour with cherries. Moreover $\langle T', \mathcal{P}'\rangle$ can be calculated in polynomial-time.*

Generalization of the results to representations that do not satisfy assumption $(P3)$ is work in progress. Note that if we allow red edge cliques in the representation $\langle T, \mathcal{P}\rangle$ then $\langle T, \mathcal{P}\rangle$ is not necessarily a planar tour, as any tour is a representation of a hole.

Another direction of research would be to investigate the relation of the class of ENPT graphs with other graph classes, in particular with EPT. ENPT$\setminus$EPT $\neq \emptyset$ because the wheel graph $W_{5,1} = C_5 + K_1$ is in ENPT $\setminus$ EPT. In [11] graphs in VPT$\cap$EPT are characterized. The characterization of the graphs in EPT$\cap$ENPT is an interesting research topic. Lastly, decision/optimization problems restricted to ENPT graphs, such as minimum vertex coloring, maximum stable set, and hardness of recognition of ENPT graphs seem to be major problems to investigate.

References

1. Biedl, T.C., Stern, M.: On edge-intersection graphs of k-bend paths in grids. Discrete Mathematics & Theoretical Computer Science 12(1), 1–12 (2010)
2. Boyacı, A., Ekim, T., Shalom, M., Zaks, S.: Graphs of Edge-Intersecting Non-Splitting Paths in a Tree: Towards Hole Representations-Part I. arXiv:1309.2898 (2013)
3. Boyacı, A., Ekim, T., Shalom, M., Zaks, S.: Graphs of Edge-Intersecting Non-Splitting Paths in a Tree: Towards Hole Representations-Part II. arXiv:1309.6471 (2013)
4. Chartrand, G., Harary, F.: Planar permutation graphs. Annales de l'institut Henri Poincaré (B) Probabilités et Statistiques 3(4), 433–438 (1967)
5. Gavril, F.: A recognition algorithm for the intersection graphs of paths in trees. Discrete Mathematics 23(3), 211–227 (1978)
6. Gavril, F.: Maximum weight independent sets and cliques in intersection graphs of filaments. Information Processing Letters 73(5-6), 181–188 (2000)
7. Golumbic, M.C., Lipshteyn, M., Stern, M.: Equivalences and the complete hierarchy of intersection graphs of paths in a tree. Discrete Appl. Math. 156, 3203–3215 (2008)
8. Golumbic, M.C., Lipshteyn, M., Stern, M.: The k-edge intersection graphs of paths in a tree. Discrete Appl. Math. 156, 451–461 (2008)
9. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Annals of Discrete Mathematics, vol. 57. North-Holland Publishing Co., Amsterdam (2004)
10. Golumbic, M.C., Jamison, R.E.: Edge and vertex intersection of paths in a tree. Discrete Mathematics 55(2), 151–159 (1985)
11. Golumbic, M.C., Jamison, R.E.: The edge intersection graphs of paths in a tree. Journal of Combinatorial Theory, Series B 38(1), 8–22 (1985)
12. Golumbic, M.C., Lipshteyn, M., Stern, M.: Representing edge intersection graphs of paths on degree 4 trees. Discrete Mathematics 308(8), 1381–1387 (2008)
13. Lévêque, B., Maffray, F., Preissmann, M.: Characterizing path graphs by forbidden induced subgraphs. J. Graph Theory 62, 369–384 (2009)
14. Spinrad, J., Sritharan, R.: Algorithms for weakly triangulated graphs. Discrete Applied Mathematics 59(2), 181–191 (1995)
15. Tarjan, R.E.: Decomposition by clique separators. Discrete Mathematics 55(2), 221–232 (1985)
16. Tsukiyama, S., Ide, M., Ariyoshi, H., Shirakawa, I.: A New Algorithm for Generating All the Maximal Independent Sets. SIAM Journal on Computing 6(3), 505–517 (1977)

Linear-Time Algorithms for Scattering Number and Hamilton-Connectivity of Interval Graphs[⋆]

Hajo Broersma[1], Jiří Fiala[2,⋆⋆], Petr A. Golovach[3], Tomáš Kaiser[4,⋆⋆⋆], Daniël Paulusma[5,†], and Andrzej Proskurowski[6]

[1] Faculty of EEMCS, University of Twente, Enschede
h.j.broersma@utwente.nl
[2] Department of Applied Mathematics, Charles University, Prague
fiala@kam.mff.cuni.cz
[3] Institute of Computer Science, University of Bergen
petr.golovach@ii.uib.no
[4] Department of Mathematics, University of West Bohemia, Plzeň
kaisert@kma.zcu.cz
[5] School of Engineering and Computing Sciences, Durham University
daniel.paulusma@durham.ac.uk
[6] Department of Computer Science, University of Oregon, Eugene
andrzej@cs.uoregon.edu

Abstract. We show that for all $k \leq -1$ an interval graph is $-(k+1)$-Hamilton-connected if and only if its scattering number is at most k. We also give an $O(n+m)$ time algorithm for computing the scattering number of an interval graph with n vertices and m edges, which improves the $O(n^3)$ time bound of Kratsch, Kloks and Müller. As a consequence of our two results the maximum k for which an interval graph is k-Hamilton-connected can be computed in $O(n+m)$ time.

1 Introduction

The Hamilton Cycle problem is that of testing whether a given graph has a Hamilton cycle, i.e., a cycle passing through all the vertices. This problem is one of the most notorious NP-complete problems within Theoretical Computer Science and remains NP-complete on many graph classes. In contrast, for interval graphs, Keil [19] showed in 1985 that Hamilton Cycle can be solved in $O(n+m)$ time, thereby strengthening an earlier result of Bertossi [4] for proper interval graphs. Bertossi and Bonucelli [5] proved that Hamilton Cycle is NP-complete for undirected path graphs, double interval graphs and rectangle graphs, all three of which are classes of intersection graphs that contain the class of interval graphs. We examine whether the linear-time result of Keil [19] can be

[⋆] Paper supported by Royal Society Joint Project Grant JP090172.
[⋆⋆] Author supported by the GraDR-EuroGIGA project GIG/11/E023 and by the project Kontakt LH12095.
[⋆⋆⋆] Author supported by project P202/11/0196 of the Czech Science Foundation.
[†] Author supported by EPSRC (EP/G043434/1).

strengthened on interval graphs to hold for other connectivity properties, which are NP-complete to verify in general.

1.1 Terminology

We only consider undirected finite graphs with no self-loops and no multiple edges. Throughout the paper we let n and m denote the number of vertices and edges, respectively, of the input graph.

Let $G = (V, E)$ be a graph. If G has a *Hamilton cycle*, i.e., a cycle containing all the vertices of G, then G is *hamiltonian.* Recall that the corresponding NP-complete decision problem is called Hamilton Cycle. If G contains a *Hamilton path*, i.e., a path containing all the vertices of G, then G is *traceable.* In this case, the corresponding decision problem is called the Hamilton Path problem, which is also well known to be NP-complete (cf. [15]). The problems 1-Hamilton Path and 2-Hamilton Path are those of testing whether a given graph has a Hamilton path that starts in some given vertex u or that is between two given vertices u and v, respectively. Both problems are NP-complete by a straightforward reduction from Hamilton Path. The Longest Path problem is to compute the maximum length of a path in a given graph. This problem is NP-hard by a reduction from Hamilton Path as well.

Let $G = (V, E)$ be a graph. If for each two distinct vertices $s, t \in V$ there exists a Hamilton path with end-vertices s and t, then G is *Hamilton-connected.* If $G - S$ is Hamilton-connected for every set $S \subset V$ with $|S| \leq k$ for some integer $k \geq 0$, then G is *k-Hamilton-connected.* Note that a graph is Hamilton-connected if and only if it is 0-Hamilton-connected. The Hamilton Connectivity problem is that of computing the maximum value of k for which a given graph is k-Hamilton-connected. Dean [12] showed that already deciding whether $k = 0$ is NP-complete. Kužel, Ryjáček and Vrána [21] proved this for $k = 1$. A straightforward generalization of the latter result yields the same for any integer $k \geq 1$. As an aside, the Hamilton Connectivity problem has recently been studied by Kužel, Ryjáček and Vrána [21], who showed that NP-completeness of the case $k = 1$ for line graphs would disprove the conjecture of Thomassen that every 4-connected line graph is hamiltonian, unless $\mathsf{P} = \mathsf{NP}$.

A *path cover* of a graph G is a set of mutually vertex-disjoint paths $P_1, \ldots, P_k$ with $V(P_1) \cup \cdots \cup V(P_k) = V(G)$. The size of a smallest path cover is denoted by $\pi(G)$. The Path Cover problem is to compute this number, whereas the 1-Path Cover problem is to compute the size of a smallest path cover that contains a path in which some given vertex u is an end-vertex. Because a Hamilton path of a graph is a path cover of size 1, Path Cover and 1-Path Cover are NP-hard via a reduction from Hamilton Path and 1-Hamilton Path, respectively.

We denote the number of connected components of a graph $G = (V, E)$ by $c(G)$. A subset $S \subset V$ is a *vertex cut* of G if $c(G - S) \geq 2$, and G is called *k-connected* if the size of a smallest vertex cut of G is at least k. We say that G is *t-tough* if $|S| \geq t \cdot c(G - S)$ for every vertex cut S of G. The *toughness* $\tau(G)$ of a graph $G = (V, E)$ was defined by Chvátal [10] as $\tau(G) = \min\{\frac{|S|}{c(G-S)} \ : \ S \subset$

V and $c(G-S) \geq 2\}$, where we set $\tau(G) = \infty$ if G is a complete graph. Note that $\tau(G) \geq 1$ if G is hamiltonian.

The *scattering number* of a graph $G = (V, E)$ was defined by Jung [18] as $\mathrm{sc}(G) = \max\{c(G-S) - |S| \ : \ S \subset V \text{ and } c(G-S) \geq 2\}$, where we set $\mathrm{sc}(G) = -\infty$ if G is a complete graph. We call a set S on which $\mathrm{sc}(G)$ is attained a *scattering set.* Note that $\mathrm{sc}(G) \leq 0$ if G is hamiltonian. Shih, Chern and Hsu [25] show that $\mathrm{sc}(G) \leq \pi(G)$ for all graphs G. Hence, $\mathrm{sc}(G) \leq 1$ if G is traceable. The Scattering Number problem is to compute $\mathrm{sc}(G)$ for a graph G.

A set of p internally vertex-disjoint paths $P_1, \ldots, P_p$, all of which have the same end-vertices u and v of a graph G, is called a *stave* or *p-stave* of G, which is *spanning* if $V(P_1) \cup \cdots \cup V(P_p) = V(G)$. Given an integer $p \geq 1$ and two vertices u and v of a general input graph G, deciding whether there exists a spanning p-stave between u and v is clearly an NP-complete problem: for $p = 1$ the problem is equivalent to 2-Hamilton Path; for $p = 2$ the problem is equivalent to the NP-complete problem of deciding whether a graph is hamiltonian; for $p \geq 3$, the NP-completeness follows easily by induction and by considering the graph obtained after adding one vertex adjacent to u and v. We call a spanning stave between two vertices u and v of a graph *optimal* if it is a p-stave and there does not exist a spanning $(p+1)$-stave between u and v.

A graph G is an *interval graph* if it is the intersection graph of a set of closed intervals on the real line, i.e., the vertices of G correspond to the intervals and two vertices are adjacent in G if and only if their intervals have at least one point in common. An interval graph is *proper* if it has a closed interval representation in which no interval is properly contained in some other interval.

1.2 Known Results

We first discuss the results on testing hamiltonicity properties for proper interval graphs. Besides giving a linear-time algorithm for solving Hamilton Cycle on proper interval graphs, Bertossi [4] also showed that a proper interval graph is traceable if and only if it is connected. His work was extended by Chen, Chang and Chang [9] who showed that a proper interval graph is hamiltonian if and only if it is 2-connected, and that a proper interval graph is Hamilton-connected if and only if it is 3-connected. In addition, Chen and Chang [8] showed that a proper interval graph has scattering number at most $2-k$ if and only if it is k-connected.

Below we survey the results on testing hamiltonicity properties for interval graphs that appeared after Keil [19] solved the Hamilton Cycle problem on interval graphs.

Testing for Hamilton cycles and Hamilton paths. The $O(n+m)$ time algorithm of Keil [19] makes use of an interval representation. One can find such a representation by executing the $O(n+m)$ time interval recognition algorithm of Booth and Lueker [6]. If an interval representation is already given, Manacher, Mankus and Smith [24] showed that Hamilton Cycle and Hamilton Path can be solved in $O(n \log n)$ time. In the same paper, they ask whether the time bound for these two problems can be improved to $O(n)$ time if a so-called sorted interval representation is given. Chang, Peng and Liaw [7] answered this question in the affirmative. They showed that this even holds for Path Cover.

When no Hamilton path exists. In this case, Longest Path and Path Cover are natural problems to consider. Ioannidou, Mertzios and Nikolopoulos [17] gave an $O(n^4)$ algorithm for solving Longest Path on interval graphs. Arikati and Pandu Rangan [1] and also Damaschke [11] showed that Path Cover can be solved in $O(n+m)$ time on interval graphs. Damaschke [11] posed the complexity status of 1-Hamilton Path and 2-Hamilton Path on interval graphs as open questions. The latter question is still open, but Asdre and Nikolopoulos [3] answered the former question by presenting an $O(n^3)$ time algorithm that solves 1-Path Cover, and hence 1-Hamilton Path. Li and Wu [22] announced an $O(n+m)$ time algorithm for 1-Path Cover on interval graphs. Deogun, Kratsch and Steiner [13] show that for all $k \geq 1$ any cocomparability graph, and hence also any interval graph, has a path cover of size at most k if and only if its scattering number is at most k. [1] They also prove that a cocomparability graph G is hamiltonian if and only if $\text{sc}(G) \leq 0$. Recall that the latter condition is equivalent to $\tau(G) \geq 1$. Hung and Chang [16] gave an $O(n+m)$ time algorithm that finds a scattering set of an interval graph G with $\text{sc}(G) \geq 0$.

1.3 Our Results

When a Hamilton path does exist. In this case, Hamilton Connectivity is a natural problem to consider. However, the results of Deogun, Kratsch and Steiner [13] suggest that trying to characterize k-Hamilton-connectivity in terms of the scattering number of an interval graph may be more appropriate than doing this in terms of its toughness. We confirm this by showing that for all $k \geq 0$ an interval graph is k-Hamilton-connected if and only if its scattering number is at most $-(k+1)$. Together with the results of Deogun, Kratsch and Steiner [13] this leads to the following theorem.

Theorem 1. *Let G be an interval graph. Then* $\text{sc}(G) \leq k$ *if and only if*

(i) G has a path cover of size at most k when $k \geq 1$
(ii) G has a Hamilton cycle when $k = 0$
(iii) G is $-(k+1)$-Hamilton-connected when $k \leq -1$.

Moreover, we give an $O(n+m)$ time algorithm for solving Scattering Number that also produces a scattering set. This improves the $O(n^3)$ time bound of a previous algorithm due to Kratsch, Kloks and Müller [20]. Combining this result with Theorem 1 yields that Hamilton Connectivity can be solved in $O(n+m)$ time on interval graphs. For proper interval graphs we combine Theorem 1 with the result of Chen and Chang [8] to state that for all $k \geq 0$, a proper interval graph is k-Hamilton-connected if and only if it is $(k+3)$-connected.

[1] This has also been shown by Lehel in an unpublished manuscript [23].

Damaschke's algorithm [11] for solving PATH COVER on interval graphs, which is based on the approach of Keil [19], actually solves the following problem in $O(n+m)$ time: given an interval graph G and an integer p, does G have a spanning p-stave between the vertex u_1 corresponding to the leftmost interval of an interval model of G and the vertex u_n corresponding to the rightmost one? We extend Damaschke's algorithm in Section 2 to an $O(n+m)$ time algorithm that takes as input only an interval graph G and finds an optimal stave of G between u_1 and u_n, unless it detects that it is not hamiltonian. Hence, $\mathrm{sc}(G) \geq 1$ as shown by Deogun, Kratsch and Steiner [13]. Therefore, the $O(n+m)$ time algorithm by Hung and Chang [16] for computing a scattering set may be applied. If there is an optimal stave between u_1 and u_n, we show how this enables us to compute a scattering set of G in $O(n+m)$ time. We then conclude that G contains a spanning p-stave between u_1 and u_n if and only if $\mathrm{sc}(G) \leq 2-p$.

In Section 3 we prove Theorem 1 (iii), i.e., the case when $k \leq -1$. In particular, for proving the subcase $k=-1$, we show that an interval graph G is Hamilton-connected if it contains a spanning 3-stave between the vertex corresponding to the leftmost interval of an interval model of G and the vertex corresponding to the rightmost one.

2 Spanning Staves and the Scattering Number

In order to present our algorithm we start by giving the necessary terminology and notations.

A set $D \subseteq V$ *dominates* a graph $G=(V,E)$ if each vertex of G belongs to D or has a neighbor in D. We will usually denote a path in a graph by its sequence of distinct vertices such that consecutive vertices are adjacent. If $P = u_1 \ldots u_n$ is a path, then we denote its *reverse* by $P^{-1} = u_n \ldots u_1$. We may concatenate two paths P and P' whenever they are vertex-disjoint except for the last vertex of P coinciding with the first vertex of P'. The resulting path is then denoted by $P \circ P'$.

A *clique path* of an interval graph G with vertices $u_1, \ldots, u_n$ is a sequence $C_1, \ldots,$ C_s of all maximal cliques of G, such that each edge of G is present in some clique C_i and each vertex of G appears in consecutive cliques only. This yields a specific interval model for G that we will use throughout the remainder of this paper: a vertex u_i of G is represented by the interval $I_{u_i} = [\ell_i, r_i]$, where $\ell_i = \min\{j : u_i \in C_j\}$ and $r_i = \max\{j : u_i \in C_j\}$, which are referred to as the *start point* and the *end point* of u_i, respectively. By definition, C_1 and C_s are maximal cliques. Hence both C_1 and C_s contain at least one vertex that does not occur in any other clique. We assume that u_1 is such a vertex in C_1 and that u_n is such a vertex in C_s. Note that $I_{u_1} = [1,1]$ and $I_{u_n} = [s,s]$ are single points.

Damaschke made the useful observation that any Hamilton path in an interval graph can be reordered into a monotone one, in the following sense.

Lemma 1 ([11]). *If the interval graph G contains a Hamilton path, then it contains a Hamilton path from u_1 to u_n.*

We use Lemma 1 to rearrange certain path systems in G into a single path as follows. Let P be a path between u_1 and u_n and let $\mathcal{Q} = (Q_1, \ldots, Q_k)$ be a collection of paths, each of which contains u_1 or u_n as an end-vertex. Furthermore, P and all the paths of $\mathcal{Q}$ are assumed to be vertex-disjoint except for possible intersections at u_1 or u_n. Consider the path Q_1. By symmetry, it may be assumed to contain u_1. We apply Lemma 1 to $P \circ (Q_1 - u_n)$ and obtain a path P' between u_1 and u_n containing all the vertices of $P \cup Q_1$. Proceeding in a similar way for the paths $Q_2, \ldots, Q_k$, we obtain a path between u_1 and u_n on the same vertex set as $P \cup \bigcup_{j=1}^{k} Q_j$. We denote the resulting path by $\operatorname{merge}(P, Q_1, \ldots, Q_k)$ or simply by $\operatorname{merge}(P, \mathcal{Q})$.

Let G be an interval graph with all the notation as introduced above. In particular, the vertices of G are $u_1, \ldots, u_n$, we consider a clique path $C_1, \ldots, C_s$, and the start point and the end point of each u_i are $\ell_i = \min\{j : u_i \in C_j\}$ and $r_i = \max\{j : u_i \in C_j\}$, respectively, where $I_{u_1} = [1, 1]$ and $I_{u_n} = [s, s]$. We can obtain this representation of G by first executing the $O(n + m)$ time recognition algorithm of interval graphs due to Booth and Lueker [6] as their algorithm also produces a clique path $C_1, \ldots, C_s$ for input interval graphs.

Algorithm 1 is our $O(n + m)$ time algorithm for finding an optimal spanning stave between u_1 and u_n if it exists. It gradually builds up a set $\mathcal{P}$ of internally disjoint paths starting at u_1 and passing through vertices of $C_t \setminus C_{t+1}$ before moving to $C_t \cap C_{t+1}$ for $t = 1, \ldots, s - 1$. It is convenient to consider all these paths ordered from u_1 to their (temporary) end-vertices that we call *terminals*, and to use the terms *predecessor*, *successor*, and *descendant* of a fixed vertex v in one of the paths with the usual meaning of a vertex immediately before, immediately after, and somewhere after v in one of these paths, respectively.

We note that the path system $\mathcal{P}$ provided by Algorithm 1 is a valid stave. A routine check confirms that the following loop invariant holds at line 6: *the last vertices of paths from $\mathcal{P}$ all belongs to the clique C_t*. This is guaranteed by the computations at lines 10–18. At line 20 it also holds that *all vertices of $C_t \setminus C_{t+1}$ appear in the current $\mathcal{P} \cup \mathcal{Q}$*, as they have been included at line 8. When the loop terminates, the remaining vertices are incorporated at line 22. Thus the resulting path system $\mathcal{P}$ is a spanning stave.

In Theorem 2 we show that no spanning stave may consist of more than $2 - \operatorname{sc}(G)$ paths. On the other hand, we will also show that the k-stave found by Algorithm 1 can be supplied with a scattering set witnessing that $k \geq 2 - \operatorname{sc}(G)$. In other words this is an optimal scattering set whose existence also proves the optimality of the spanning stave. For this goal, we first develop some auxiliary terminology related to our algorithm.

We say that a vertex v has been *added to a path*, if, at some point in the execution of Algorithm 1, some path $R \in \mathcal{P}$ such that $v \notin V(R)$ has been extended to a longer path containing v (and possibly some other new vertices). If u_i has been processed by the algorithm and added to a path at lines 8 or 11 of Algorithm 1, we say that u_i has been *activated* at time a_i, and we assign a_i the current value of the variable t. Thus, we think of time steps $t = 1, \ldots, t = s$ during the execution of the algorithm. When at the same or a later stage a

```
Input: A clique-path $C_1, \ldots, C_s$ in an interval graph $G$.
Output: An optimal spanning stave $\mathcal{P}$ between $u_1$ and $u_n$, if it exists.
1  begin
2      let $p = \deg(u_1)$;
3      let $R_i = u_1$ for all $i = 1, \ldots, p$;
4      let $\mathcal{P} = \{R_1, \ldots, R_p\}$;
5      let $\mathcal{Q} = \emptyset$;
6      for $t := 1$ to $s - 1$ do
7          choose a $P \in \mathcal{P}$ whose terminal has the smallest end point among all
           terminals;
8          if $C_t \setminus (C_{t+1} \cup \bigcup(\mathcal{P} \cup \mathcal{Q})) \neq \emptyset$ then extend $P$ by attaching vertices of
           $C_t \setminus (C_{t+1} \cup \bigcup(\mathcal{P} \cup \mathcal{Q}))$ in an arbitrary order
9          for every path $R \in \mathcal{P}$ do
10             if the terminal of $R$ is not in $C_{t+1}$ then
11                 try to extend $R$ by a new vertex $u$ from $(C_t \cap C_{t+1}) \setminus \bigcup(\mathcal{P} \cup \mathcal{Q})$
                   with the smallest end point;
12                 if such $u$ does not exist then
13                     remove $R$ from $\mathcal{P}$;
14                     insert $R$ into $\mathcal{Q}$;
15                     decrement $p$;
16                     if $p = 0$ then report that $G$ has no spanning 1-stave
                       between $u_1$ and $u_n$ and quit
17                 end
18             end
19         end
20     end
21     choose any $P \in \mathcal{P}$;
22     extend $P$ by attaching vertices of $C_s \setminus \bigcup(\mathcal{P} \cup \mathcal{Q})$ in an arbitrary order;
23     let $P = \text{merge}(P, \mathcal{Q})$;
24     for every path $R \in \mathcal{P} \setminus P$ do extend $R$ by $u_n$;
25     report the optimal spanning $p$-stave $\mathcal{P}$.
26 end
```

Algorithm 1. Finding an optimal spanning stave

vertex u_j has been added as a successor of u_i to a path, we say that u_i has been *deactivated* at time d_i, and assign $d_i = a_j$. Hence, as soon as a_i and d_i have assigned values, we have $\ell_i \leq a_i \leq d_i \leq r_i$. Furthermore, any of the implied inequalities holds whenever both of its sides are defined. Note that any of these inequalities may be an equality; in particular, a vertex can be activated and deactivated at the same time.

If the involved parameters have assigned values, we consider the open (time) intervals (ℓ_i, a_i), (a_i, d_i) and (d_i, r_i), and we say that u_i is *free* during (ℓ_i, a_i) if this interval is nonempty, *active* during (a_i, d_i) if this interval is nonempty, and *depleted* during (d_i, r_i) if this interval is nonempty. In particular, note that the vertices that are added to a path at line 8 (if any) are from $C_t \setminus C_{t+1}$, so they satisfy $r_i = t$ and $a_i = t$. Such vertices will not be active or depleted during any (nonempty) time interval, but they are free during the time interval (ℓ_i, r_i) if this interval is nonempty.

For $1 \leq j \leq k \leq s$, we define $C_{j,k} = \left(\bigcup_{i=j}^{k} C_i\right)$.

The following lemma is crucial. (Its proof is omitted due to space restrictions.)

Lemma 2. *Suppose that Algorithm 1 terminates at line 16 or finishes an iteration of the loop at lines 6–20. Let the current value of the variable t be also denoted by t. If there is at least one depleted vertex during the interval $(t, t+1)$, then there exists an integer $t' < t$ with the following properties:*

(i) $C_{t'+1,t} \setminus (C_{t'} \cup C_{t+1}) \neq \emptyset$,
(ii) a unique vertex $u_i \in C_{t'} \cap C_{t+1}$ is active during $(t', t'+1)$ and is depleted during $(t, t+1)$,
(iii) all vertices that are active during $(t, t+1)$ are also active during $(t', t'+1)$, with the only possible exception of the last descendant of u_i (which we denote by v) that can be free during $(t', t'+1)$,
(iv) all vertices that are depleted during $(t, t+1)$ and distinct from u_i are also depleted during $(t', t'+1)$,
(v) all vertices that are active during $(t', t'+1)$ are also active during $(t, t+1)$, with the only exception of u_i, and
(vi) all vertices that are free during $(t', t'+1)$ are also free during $(t, t+1)$, with the only possible exception of v if it is active during $(t, t+1)$.

Now we are ready to state and prove the main structural result.

Theorem 2. *An interval non-complete graph G contains a spanning p-stave between u_1 and u_n if and only if* $\mathrm{sc}(G) \leq 2 - p$.

Proof. Let us first assume that $\mathcal{P} = (R_1 \ldots, R_p)$ is a spanning p-stave between u_1 and u_n. If G is complete, then the claim is trivial. Otherwise, let $S \subset V(G)$ be a scattering set. We claim that $u_1, u_n \notin S$. Suppose the contrary. Since the vertex u_1 is simplicial, i.e. its neighborhood induces a clique, we get that $c(G - S) \leq c(G - (S - \{u_1\}))$ and therefore $c(G - S) - |S| < c(G - (S - \{u_1\})) - |S - \{u_1\}|$, a contradiction with the choice of S. The argument for u_n is symmetric.

Since each path in $\mathcal{P}$ connects u_1 and u_n, the union of intervals corresponding to the internal vertices of such a path is the interval $[1, s]$. In other words, the internal vertices of each path in $\mathcal{P}$ dominate G. Hence, the vertex cut S contains an internal vertex from each path of $\mathcal{P}$. From each path R_i of $\mathcal{P}$, we choose a vertex $s_i \in S$ and set $S' = \{s_1, \ldots, s_p\}$.

Consider the spanning subgraph G' of G induced by the edges of $\mathcal{P}$. Observe that $G' - S'$ has two components. If we remove the remaining vertices of $S \setminus S'$ one by one, then with each vertex we remove, the number of components of the remaining graph can increase by at most one as $u_1, u_n \notin S$. Hence $c(G - S) \leq c(G' - S) \leq 2 + |S| - p$ and $\mathrm{sc}(G) \leq 2 - p$, proving the forward implication of the statement.

For the other direction, let us assume that G does not have a spanning p-stave between u_1 and u_n. If $\deg(u_1) < p$, then let S be the set of neighbors of u_1. Because G is not a complete graph, $u_n \notin S$, i.e., S is a vertex cut and $c(G - S) \geq 2$. Then $\mathrm{sc}(G) \geq c(G - S) - |S| \geq 2 - |S| > 2 - p$. Otherwise, if

$\deg(u_1) \geq p$, then during the execution of Algorithm 1, at some stage the value set at line 15 becomes smaller than p. Suppose t_1 is the value of the variable t at this moment. We will complete the proof by constructing a scattering set S and showing that for this set $c(G-S) - |S| > 2 - p$.

We repeatedly use Lemma 2 and find a finite sequence $t_1, t_2, \ldots, t_k$, such that $t_{i+1} = (t_i)'$ as long as there are depleted vertices during (t_i, t_i+1) for $i < k$. Notice that there are no depleted vertices during $(1,2)$, i.e., this process stops and we have no depleted vertices during (t_k, t_k+1). We choose $S = \bigcup_{i=1}^{k}(C_{t_i} \cap C_{t_{i+1}})$ and prove that $G-S$ has at least $|S| - p + 3$ components.

The subgraphs $G[C_{1,t_k}] - S$ and $G[C_{t_1+1,s}] - S$ contain u_1 and u_n, respectively; in particular, they have at least one component each. By property (i) in Lemma 2, $G[C_{t_{i+1}+1,t_i}] - S$ has at least one component for each $i \in \{1, \ldots, k-1\}$. Since all these components are distinct components of $G-S$, the graph $G-S$ has at least $k+1$ components.

By properties (ii), (v) and (vi) in Lemma 2, $(C_{t_{i+1}} \cap C_{t_{i+1}+1}) \setminus (C_{t_i} \cap C_{t_i+1})$ contains only vertices that are depleted during $(t_{i+1}, t_{i+1}+1)$ for each $i \in \{1, \ldots, k-1\}$. Further, $C_{t_1} \cap C_{t_1+1}$ has no vertices that are free during $(t, t+1)$, because at least one path is not extendable at time t_1. Also this set has at most $p-1$ vertices that are active during $(t, t+1)$. Hence, the remaining vertices are depleted. By properties (ii) and (iv) in Lemma 2, for each $i \in \{1, \ldots, k-1\}$, exactly one vertex that is depleted during (t_i, t_{i+1}) has a different status during $(t_{i+1}, t_{i+1}+1)$ and is active. It follows that $|S| \leq (p-1) + (k-1) = k + p - 2$ as required. □

Recall that the scattering number can be determined in $O(n+m)$ time by an algorithm of Hung and Chang [16] if the scattering number is positive. Then, by analyzing Algorithm 1, we get the following result:

Corollary 1. *The scattering number as well as a scattering set of an interval graph can be computed in $O(n+m)$ time.*

The only operation whose time complexity has not been discussed is merge(P, $\mathcal{Q}$) at line 23. We refer to Damaschke's proof of Lemma 1 to verify that this can be implemented in $O(n+m)$ time. Our proof of Theorem 2 provides a construction of a scattering set that can be straightforwardly implemented in linear time.

3 Hamilton-Connectivity

In this section we prove our contribution to Theorem 1, which is the following.

Theorem 3. *For all $k \geq 0$, an interval graph G is k-Hamilton-connected if and only if* $\mathrm{sc}(G) \leq -(k+1)$.

Proof. Let $k \geq 0$ and G be an interval graph with leftmost and rightmost vertices u_1 and u_n as defined before. The statement of Theorem 3 is readily seen to hold when G is a complete graph. Hence we may assume without loss of generality that G is not complete.

First suppose that G is k-Hamilton-connected. Then G has at least $k+3$ vertices. We claim that $G-R$ is traceable for every subset $R \subset V(G)$ with $|R| \leq k+2$. In order to see this, suppose that $R \subseteq V(G)$ with $|R| \leq k+2$. We may assume without loss of generality that $|R| = k+2$. Let s and t be two vertices of R. By definition, $G^* = G - (R \setminus \{s,t\})$ has a Hamilton path with end-vertices s and t. Hence $G - R = G^* - \{s,t\}$ is traceable. Below we apply this claim twice.

Because G is not complete, G has a scattering set S. By definition, S is a vertex cut. Hence $S = \{s_1, \ldots, s_\ell\}$ for some $\ell \geq k+3$, as otherwise $G-S$ would be traceable, and thus connected, due to our claim. Let $T = \{s_1, \ldots, s_{k+2}\}$ and let $U = \{s_{k+3}, \ldots, s_\ell\}$. By our claim, $G' = G - T$ is traceable implying that $\mathrm{sc}(G') \leq 1$ [25]. Because $c(G'-U) = c(G-S) \geq 2$, we find that U is a vertex cut of G'. We use these two facts to derive that $1 \geq \mathrm{sc}(G') \geq c(G'-U) - |U| = c(G-T-U) - |T| - |U| + |T| = c(G-S) - |S| + |T| = \mathrm{sc}(G) + |T| = \mathrm{sc}(G) + k + 2$, implying that $\mathrm{sc}(G) \leq 1 - (k+2) = -(k+1)$, as required.

Now suppose that $\mathrm{sc}(G) \leq -(k+1)$. First let $k = 0$. By Theorem 2, there exists a spanning 3-stave $\mathcal{P} = (P, Q, R)$ between u_1 and u_n. Let v, w be an arbitrary pair of vertices of G. We distinguish four cases in order to find a Hamilton path between v and w.

Case 1: $v = u_1$ and $w = u_n$. In this case, $\mathrm{merge}(P, Q, R)$ is the desired Hamilton path.

Case 2: $v = u_1$ and $w \neq u_n$. Assume without loss of generality that $w \in R$. We split R before w into the subpaths R_1 and R_2, i.e., w becomes the first vertex of R_2 and it does not belong to R_1. Then $\mathrm{merge}(P, Q, R_1) \circ R_2^{-1}$ is the desired path. The case with $v \neq u_1$ and $w = u_n$ is symmetric.

Case 3: $v \neq u_1$ and $w \neq u_n$ belong to different paths, say $v \in Q$ and $w \in R$. We split Q after v into Q_1 and Q_2, and we also split R before w, as above. Then $Q_1^{-1} \circ \mathrm{merge}(P, Q_2, R_1) \circ R_2^{-1}$ is the desired path.

Case 4: $v \neq u_1$ and $w \neq u_n$ belong to the same path, say Q. Without loss of generality, assume that both $v \neq u_1$ and $w \neq u_n$ appear in this order on Q. We split Q after v and before w into three subpaths Q_1, Q_2, Q_3. If v and w are consecutive on Q, i.e., when Q_2 is empty, then $Q_1^{-1} \circ \mathrm{merge}(P, R) \circ Q_3^{-1}$ is the desired path. Otherwise, let z be any vertex on R that is a neighbor of the first vertex of Q_2. Such z exists since the path R dominates G. We split R after z into R_1 and R_2. By the choice of z, R_1 and Q_2 can be combined through z into a valid path R' containing exactly the same vertices as R_1 and Q_2 and starting at u_1. Then we choose $Q_1^{-1} \circ \mathrm{merge}(P, R', R_2) \circ Q_3^{-1}$.

Now let $k \geq 1$. Let S be a set of vertices with $|S| \leq k$. We need to show that $G-S$ is Hamilton-connected. Let T be a scattering set of $G-S$ and let $S^* = S \cup T$. Because T is a scattering set of $G-S$, we find that S^* is a vertex cut of G. We use this to derive that $\mathrm{sc}(G-S) = c(G-S-T) - |T| = c(G-S^*) - |S^*| + |S^*| - |T| \leq \mathrm{sc}(G) + k - 0 \leq -1$. Then, by returning to the case $k = 0$ with $G-S$ instead of G, we find that $G-S$ is Hamilton-connected, as required. This completes the proof of Theorem 3. $\square$

4 Future Work

We conclude our paper by posing a number of open problems. We start with recalling two open problems posed in the literature. First of all, Damaschke's question [11] on the complexity status of the 2-Hamilton Path problem is still open. Our results imply that we may restrict ourselves to interval graphs with scattering number equal to 0 or 1. This can be seen as follows. Let G be an interval graph that together with two of its vertices u and v forms an instance of 2-Hamilton Path. We apply Corollary 1 to compute $\mathrm{sc}(G)$ in $O(n+m)$ time. If $\mathrm{sc}(G) < 0$, then G is Hamilton-connected by Theorem 1. Then, by definition, there exists a Hamilton path between u and v. If $\mathrm{sc}(G) > 1$, then G is not traceable, also due to Theorem 1. Hence, there exists no Hamilton path between u and v.

Second, Asdre and Nikolopoulos [3] asked about the complexity status of the ℓ-Path Cover problem on interval graphs. This problem generalizes 1-Path Cover and is to determine the size of a smallest path cover of a graph G subject to the additional condition that every vertex of a given set T of size ℓ is an end-vertex of a path in the path cover. The same authors show that both ℓ-Path Cover and 2-Hamilton Path can be solved in $O(n+m)$ time on proper interval graphs [2].

The Spanning Stave problem is that of computing the minimum value of p for which a given graph has a spanning p-stave. Because a Hamilton path of a graph is a spanning 1-stave and Hamilton Path is NP-complete, this problem is NP-hard. What is the computational complexity of Spanning Stave on interval graphs?

Kratsch, Kloks and Müller [20] gave an $O(n^3)$ time algorithm for solving Toughness on interval graphs. Is it possible to improve this bound to linear on interval graphs just as we did for Scattering Number?

Finally, can we extend our $O(n + m)$ time algorithms for Hamilton Connectivity and Scattering Number to superclasses of interval graphs such as circular-arc graphs and cocomparability graphs? The complexity status of Hamilton Connectivity is still open for both graph classes, although Hamilton Cycle can be solved in $O(n^2 \log n)$ time on circular-arc graphs [25] and in $O(n^3)$ time on cocomparability graphs [14]. It is known [20] that Scattering Number can be solved in $O(n^4)$ time on circular-arc graphs and in polynomial time on cocomparability graphs of bounded dimension.

Acknowledgement. We thank referees for valuable references to related works.

References

1. Arikati, S.R., Pandu Rangan, C.: Linear algorithm for optimal path cover problem on interval graphs. Inf. Proc. Let. 35, 149–153 (1990)
2. Asdre, K., Nikolopoulos, S.D.: A polynomial solution to the k-fixed-endpoint path cover problem on proper interval graphs. Theor. Comp. Sci. 411, 967–975 (2010)

3. Asdre, K., Nikolopoulos, S.D.: The 1-fixed-endpoint path cover problem is polynomial on interval graphs. Algorithmica 58, 679–710 (2010)
4. Bertossi, A.A.: Finding hamiltonian circuits in proper interval graphs. Inf. Proc. Let. 17, 97–101 (1983)
5. Bertossi, A.A., Bonucelli, M.A.: Hamilton circuits in interval graph generalizations. Inf. Proc. Let. 23, 195–200 (1986)
6. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. J. Com. Sys. Sci. 13, 335–379 (1976)
7. Chang, M.-S., Peng, S.-L., Liaw, J.-L.: Deferred-query: An efficient approach for some problems on interval graphs. Networks 34, 1–10 (1999)
8. Chen, C., Chang, C.-C.: Connected proper interval graphs and the guard problem in spiral polygons. In: Deza, M., Manoussakis, I., Euler, R. (eds.) CCS 1995. LNCS, vol. 1120, pp. 39–47. Springer, Heidelberg (1996)
9. Chen, C., Chang, C.-C., Chang, G.J.: Proper interval graphs and the guard problem. Disc. Math. 170, 223–230 (1997)
10. Chvátal, V.: Tough graphs and hamiltonian circuits. Disc. Math. 5, 215–228 (1973)
11. Damaschke, P.: Paths in interval graphs and circular arc graphs. Disc. Math. 112, 49–64 (1993)
12. Dean, A.M.: The computational complexity of deciding hamiltonian-connectedness. Congr. Num. 93, 209–214 (1993)
13. Deogun, J.S., Kratsch, D., Steiner, G.: 1-tough cocomparability graphs are Hamiltonian. Disc. Math. 170, 99–106 (1997)
14. Deogun, J.S., Steiner, G.: Polynomial algorithms for hamiltonian cycle in cocomparability graphs. SIAM J. Comp. 23, 520–552 (1994)
15. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co. Ltd. (1979)
16. Hung, R.-W., Chang, M.-S.: Linear-time certifying algorithms for the path cover and hamiltonian cycle problems on interval graphs. Appl. Math. Lett. 24, 648–652 (2011)
17. Ioannidou, K., Mertzios, G.B., Nikolopoulos, S.D.: The longest path problem has a polynomial solution on interval graphs. Algorithmica 61, 320–341 (2011)
18. Jung, H.A.: On a class of posets and the corresponding comparability graphs. J. Comb. Th. B 24, 125–133 (1978)
19. Keil, J.M.: Finding hamiltonian circuits in interval graphs. Inf. Proc. Let. 20, 201–206 (1985)
20. Kratsch, D., Kloks, T., Müller, H.: Measuring the vulnerability for classes of intersection graphs. Disc. Appl. Math. 77, 259–270 (1997)
21. Kužel, R., Ryjáček, Z., Vrána, P.: Thomassen's conjecture implies polynomiality of 1-hamilton-connectedness in line graphs. J Graph Th. 69, 241–250 (2012)
22. Li, P., Wu, Y.: A linear time algorithm for solving the 1-fixed-endpoint path cover problem on interval graphs, draft
23. Lehel, J.: The path partition of cocomparability graphs (1991) (manuscript)
24. Manacher, G.K., Mankus, T.A., Smith, C.J.: An optimum $\Theta(n \log n)$ algorithm for finding a canonical hamiltonian path and a canonical hamiltonian circuit in a set of intervals. Inf. Proc. Let. 35, 205–211 (1990)
25. Shih, W.K., Chern, T.C., Hsu, W.L.: An $O(n^2 \log n)$ time algorithm for the hamiltonian cycle problem on circular-arc graphs. SIAM J. Comp. 21, 1026–1046 (1992)

Equilateral L-Contact Graphs

Steven Chaplick[1,⋆], Stephen G. Kobourov[2,⋆⋆], and Torsten Ueckerdt[3]

[1] Dept. Applied Mathematics, Charles University, Prague, Czech Republic
[2] Dept. of Computer Science, University of Arizona, Tucson AZ, USA
[3] Dept. of Mathematics, Karlsruhe Istitute of Technology, Karlsruhe, Germany

Abstract. We consider *L-graphs*, that is contact graphs of axis-aligned L-shapes in the plane, all with the same rotation. We provide several characterizations of L-graphs, drawing connections to Schnyder realizers and canonical orders of maximally planar graphs. We show that every contact system of L's can always be converted to an equivalent one with equilateral L's. This can be used to show a stronger version of a result of Thomassen, namely, that every planar graph can be represented as a contact system of square-based cuboids.

We also study a slightly more restricted version of equilateral L-contact systems and show that these are equivalent to homothetic triangle contact representations of maximally planar graphs. We believe that this new interpretation of the problem might allow for efficient algorithms to find homothetic triangle contact representations, that do not use Schramm's monster packing theorem.

1 Introduction

A *contact graph* is a graph whose vertices are represented by geometric objects (such as curves, line segments, or polygons), and edges correspond to two objects touching in some specified fashion. There is a large body of work about representing planar graphs as contact graphs. An early result is Koebe's 1936 theorem [16] that all planar graphs can be represented by touching disks.

In 1990 Schnyder showed that maximally planar graphs contain rich combinatorial structure [18]. With an angle labeling and a corresponding edge labeling, Schnyder shows that maximally planar graphs can be decomposed into three edge disjoint spanning trees. This combinatorial structure, called Schnyder realizer, can be transformed into a geometric structure to produce a straight-line crossing-free planar drawing of the graph with vertex coordinates on the integer grid. While Schnyder realizers were defined for maximally planar graphs [18], the notion generalizes to 3-connected planar graphs [9]Fusy's transversal structures [11] for irreducible triangulations of the 4-gon also provide combinatorial structure that can be used to obtain geometric results. Later, de Fraysseix *et al.* [7] show how to use Schnyder realizer to produce a representation of planar graphs as T-contact graphs (vertices are axis-aligned T's and edges correspond to point contact between T's) and triangle contact graphs.

Recently, a similar combinatorial structure, called *edge labeling*, was identified for the class of planar Laman graphs, and this was used to produce a representation of such graphs as L-contact graphs, with L-shapes in all four rotations [15]. Planar Laman

⋆ Supported by the ESF GraDR EUROGIGA grant as project GACR GIG/11/E023.
⋆⋆ Research supported in part by NSF grants CCF-1115971 and DEB 1053573.

graphs contain several large classes of planar graphs (e.g., series-parallel graphs, outer-planar graphs, planar 2-trees) and are also of interest in structural mechanics, chemistry and physics, due to their connection to rigidity theory [13].

Planar bipartite graphs can be represented by axis-aligned segment contacts [6,17]. Triangle-free planar graphs can be represented via contacts of segments with only three slopes [5]. They can also be represented by contact axis-aligned line segments, L-shapes, and Γ-shapes [4].

Planar graphs have also been considered as intersection graphs of geometric objects. One major result is the proof of Scheinerman's conjecture that all planar graphs are intersection graphs of line segments in the plane [3]. Recently the *k-bend Vertex intersection graphs of Paths in Grids (B_k-VPG)*were introduced and it was shown that planar graphs are B_3-VPG [1]. It was recently shown that planar graphs are B_2-VPG [4], where the authors also conjectured that all planar graphs are intersection graphs of one fixed rotation of axis-aligned L-shapes (a special case of B_1-VPG).

In the 3D case Thomassen [20] shows that any planar graph has a proper contact representation by touching cuboids (axis-alligned boxes). Felsner and Francis [10] show that any planar graph has a (not necessarily proper) representation by touching cubes. In a *proper contact representation of cuboids* contacts must always have non-zero area and *cubes* are special cuboids where all sides have the same length. Recently Bremner *et al.* [2] described two new proofs of Thomassen's result: one based on canonical orders [8] and the other based on Schnyder's realizers [18].

Our Contributions: In this paper we consider contact graphs of L-shapes in only one fixed rotation, so-called L-graphs. In Section 2 we briefly review Schnyder realizers, T-contact representations, triangle contact representations, and canonical orders. In Section 3 we characterize L-graphs in terms of canonical orders, Schnyder realizers, and edge labelings. We also show how to recognize L-graphs in polynomial time. In Section 4 we show that every L-representation has an equivalent one with only equilateral L-shapes. Using this we strengthen the result of Thomassen [20] and Bremner *et al.* [2], by showing that every planar graph admits a proper contact representation with square-based cuboids. Finally, we consider a special class of equilateral L-representations, drawing connections to homothetic triangle contact representations of maximally planar graphs and contact representations with cubes.

2 Preliminaries

Schnyder realizers for maximally planar graphs were originally described in 1990 [18] and have played a central role in numerous problems for planar graphs.

Definition 1 ([18]). *Let $G = (V, E)$ be a maximally planar graph with a fixed plane embedding. Let v_1, v_2, v_n be the outer vertices in clockwise order. A* Schnyder realizer *of G is an orientation and coloring of the inner edges of G with colors* 1 *(red),* 2 *(blue) and n (green), such that:*

(i) Around every inner vertex v in clockwise order there is one outgoing red edge, a possibly empty set of incoming green edges, one outgoing blue edge, a possibly empty set of incoming red edges, one outgoing green edge, a possibly empty set of incoming blue edges.

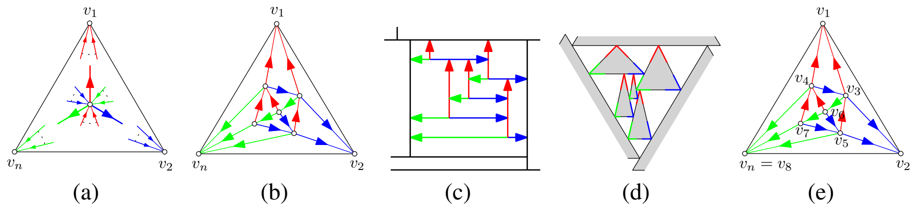


Fig. 1. (a) The Schnyder rules for inner and outer vertices. (b) A maximally planar graph G with a Schnyder realizer (S_1, S_2, S_n). (c) A T-contact representation of G w.r.t. (S_1, S_2, S_n). (d) A triangle contact representation of G w.r.t. (S_1, S_2, S_n). (e) A canonical order of G w.r.t. S_1, S_2.

(ii) All inner edges at outer vertices are incoming and edges at v_1 are colored red, edges at v_2 are colored blue, edges at v_n are colored green.

Schnyder realizer have several useful properties; see Fig. 1. For example, if S_1, S_2 and S_n are the sets of red, blue and green edges, then for $i = 1, 2, n$ we have that S_i is a directed tree spanning all inner vertices plus v_i, where each edge is oriented towards v_i. This way the orientation of edges can be recovered from their coloring and hence we denote a Schnyder realizer simply by the triple (S_1, S_2, S_n). For $i = 1, 2, n$ let S_i^{-1} be the set S_i with the orientation of every edge reversed. It is well-known that for every Schnyder realizer $S_1 \cup S_2 \cup S_n^{-1}$ is an acyclic set of directed edges.

Schnyder realizers are often used to show that planar graphs admit certain contact representations. In a *T-contact representation* of a maximally planar graph $G = (V, E)$ the vertices are assigned to interior disjoint axis-aligned upside down T-shapes, so that two T-shapes touch in a point if and only if the corresponding vertices are joined by an edge in G. For a vertex $v \in V$ let T_v be the corresponding T-shape. From every T-contact representation we get a Schnyder realizer by coloring an edge uv red (respectively blue and green) if the top (respectively left and right) endpoint of T_u is contained in T_v; see Fig. 1(c).

Similarly to T-contact representations, de Fraysseix *et al.* [7] consider triangle contact representations. In a *triangle contact representation* of a maximally planar graph $G = (V, E)$ the vertices are assigned to interior disjoint triangles, so that two triangles touch in a point if and only if the corresponding vertices are joined by an edge in G. We can indeed assume w.l.o.g. all triangles are isosceles with horizontal bases and the tip above. For a vertex $v \in V$ let Δ_v be the corresponding triangle. We again get a Schnyder realizer by coloring an edge uv red (respectively blue and green) if the top (respectively left and right) corner of Δ_u is contained in Δ_v; see Fig. 1(d).

Theorem 1 ([7]). *Let G be a maximally planar graph with a fixed embedding. Then:*

- *Every T-contact representation defines a Schnyder realizer and vice versa.*
- *Every triangle contact representation defines a Schnyder realizer and vice versa.*

A *homothetic triangle representation* is a triangle contact representation in which every triangle is a translate and/or uniform scaling of a fixed triangle. It has been noticed by Gonçalves, Lévêque and Pinlou [12], that a result of Schramm [19] implies the following.

Theorem 2 ([12]). *Every 4-connected maximally planar graph admits a homothetic triangle representation.*

Canonical orders were first introduced by De Fraysseix, Pach and Pollack in 1990 [8]. For maximally planar graphs Schnyder realizers and canonical orders are very closely related, as shown in Lemma 1 below.

Definition 2 ([8]). *Let $G = (V, E)$ be a biconnected planar graph with a fixed embedding and some distinguished outer edge v_1v_2. A* canonical order of G *is a permutation $(v_1, v_2, v_3, \ldots, v_n)$ of the vertices of G, such that:*

(i) For each $i \geq 3$ the induced subgraph G_i of G on $\{v_1, \ldots, v_i\}$ is biconnected, and the boundary of its outer face is a cycle C_i containing the edge v_1v_2.
(ii) For each $i \geq 4$ the vertex v_i lies in the outer face of G_{i-1}, and its neighbors in G_{i-1} form a subpath of $C_i \setminus v_1v_2$.

The outer edge v_1v_2 of G is then called the base edge of the canonical order.

Lemma 1. *If G is a maximally planar graph with Schnyder realizer (S_1, S_2, S_n), then every topological ordering of $S_1 \cup S_2 \cup S_n^{-1}$ defines a canonical order of G. Moreover, every canonical order of G is a topological order of $S_1 \cup S_2 \cup S_n^{-1}$ for some Schnyder realizer (S_1, S_2, S_n).*

We call a canonical order that is a topological order of $S_1 \cup S_2 \cup S_n^{-1}$ a *canonical order w.r.t.* S_1, S_2. See Fig. 1(e) for an example. Note that the same Schnyder realizer may give rise to several canonical orders as for example swapping the order of v_4 and v_5 in Fig. 1(e) results in a different canonical order w.r.t. S_1, S_2.

Another vertex order that can be defined for any graph is the so-called k-degenerate order. For an n-vertex graph G and a number $k \in \mathbb{N}$ $(v_1, \ldots, v_n)$ is a *k-degenerate order of G* if for each $i = 1, \ldots, n$ the vertex v_i has no more than k neighbors in the induced subgraph G_{i-1} of G on $\{v_1, \ldots, v_{i-1}\}$. Moreover, a *maximally k-degenerate order of* G further requires v_i to have exactly $\min\{i-1, k\}$ neighbors in G_i. A graph is *(maximally) k-degenerate* if it admits some (maximally) k-degenerate order. A very important subclass of maximally k-degenerate graphs are k-trees. A maximally k-degenerate graph G is a *k-tree* if in some k-degenerate order of G the neighborhood of v_i is a clique in G_{i-1}, $i = 1, \ldots, n$. Equivalently, k-trees are exactly the inclusion-maximal graphs of tree-width k; i.e., adding any edge to a k-tree results in a graph with tree-width $k+1$.

3 Contact L-Graphs: Characterization and Recognition

An *L-contact representation*, or *L-representation* for short, of a graph $G = (V, E)$ is a set of interior disjoint axis-aligned L-shapes, one for each vertex, such that two L-shapes touch in a point if and only if the corresponding vertices in G are adjacent. Unless stated otherwise we allow only one of the four possible rotations of L-shapes here. An L-representation is *degenerate* if two endpoints of L-shapes or an endpoint and a bend coincide, and *non-degenerate* otherwise. A graph is an *L-contact graph* or simply *L-graph* if it admits an L-representation. Since one can remove any contact in

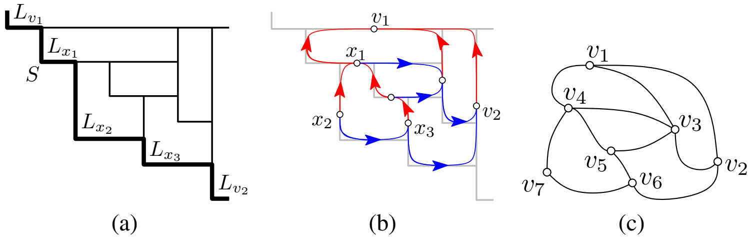


Fig. 2. (a) An L-representation with base edge v_1v_2 and outer staircase $S \subset L_{v_1} \cup L_{x_1} \cup L_{x_2} \cup L_{x_3} \cup L_{v_2}$ drawn thick. (b) The corresponding embedded L-graph with the corresponding edge labeling. (c) A corresponding 2-canonical order of the graph.

an L-representation by shortening one L, L-graphs are closed under taking subgraphs. Throughout this section we consider *maximal L-graphs* only, that is, L-graphs (with $n \geq 2$ vertices), that are not proper subgraphs of another L-graph (with n vertices).

For a fixed L-representation we denote the L-shape corresponding to a vertex v by L_v. The vertex for the L-shape with topmost horizontal leg and the vertex for the L-shape with rightmost vertical leg is denoted by v_1 and v_2, respectively. The edge v_1v_2 is called the *base edge of the L-representation*. Every L-representation defines a plane embedding of the underlying L-graph G. Each inner face of G corresponds to a rectilinear polygon whose boundary lies in the union of L-shapes for the vertices of that face. The L-shapes whose bends lie in at most one such rectilinear polygon correspond to the outer vertices of G. The maximal rectilinear path S containing all bends of these L-shapes is called the *outer staircase of the L-representation*. The L-shapes appear along S starting with L_{v_1} and ending with L_{v_2} in the same order as the outer vertices of G along the outer face starting with v_1 and ending with v_2; see Fig. 2.

For a maximally planar graph G and a Schnyder realizer (S_1, S_2, S_n) of G we define $G \setminus S_n$ as the graph $(V \setminus v_n, E \setminus S_n)$.

Lemma 2. *For every maximal L-graph G with base edge v_1v_2 there is a maximally planar graph H with a Schnyder realizer (S_1, S_2, S_n), such that $G = H \setminus S_n$.*

Proof. We consider any L-representation of G with base edge v_1v_2. We introduce a T-shape T_{v_n} whose vertical leg lies to the left of L_{v_1} and whose horizontal leg lies below L_{v_2}. We obtain a T-representation by adding a left leg to every L-shape so that its endpoint touches some vertical leg but is interior disjoint from any other leg. Let H be the maximally planar graph with that T-representation and (S_1, S_2, S_n) be the corresponding Schnyder realizer. Then $G = H \setminus S_n$. $\square$

Recall from Definition 2 that if $(v_1, \dots, v_n)$ is a canonical order of some biconnected graph G, then for every $i \in \{3, \dots, n\}$ the subgraph $G_i = G[v_1, \dots, v_i]$ is also biconnected, which implies that for each $i = 3, \dots, n$ the vertex v_i has degree at least two in $G[v_1, \dots, v_i]$. A 2*-canonical order* is a canonical order for which each v_i has degree exactly two in G_i. In particular a 2-canonical order is a special 2-degenerate order of a planar graph that depends on the chosen embedding. Note that there are planar maximal 2-degenerate graphs that admit no 2-canonical order; see Fig. 3(a) and (b). Note also that the graph in Fig. 3(a) admits a 2-degenerate order in which every vertex is put into the outer face of the graph induced by vertices of smaller index.

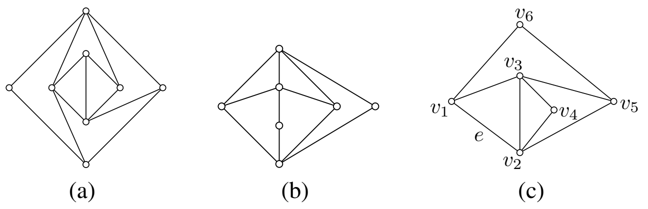


Fig. 3. (a),(b) Planar maximal 2-degenerate graphs that admit no 2-canonical order. (c) A graph with a 2-canonical order with base edge $e = v_1v_2$.

Lemma 3. *If a graph admits a 2-canonical order with base edge v_1v_2 then it admits an L-representation with base edge v_1v_2. Moreover, given a 2-canonical order an L-representation can be found in linear time.*

Proof. We use induction on the number of vertices, where the base case of just two vertices trivially holds. So let G be a graph on at least three vertices. Assume that G admits a 2-canonical order and let x be the last vertex in the order. Applying induction to $G \setminus x$ – a graph with a 2-canonical order in which both neighbors of x lie on the outer face – we obtain an L-representation of $G \setminus x$. The L-shapes for the two neighbors, u and v, of x appear on the outer staircase S. It is now possible to add an L-shape L_x, making contact with L_u and L_v, and this way obtain an L-representation of G. □

For a graph G with a fixed plane embedding and distinguished outer edge v_1v_2 we define an *edge labeling of G with base edge v_1v_2* to be an orientation and coloring of the edges of G different from v_1v_2 with colors 1 (red) and 2 (blue), such that:

(i) Around every inner vertex v in clockwise order there is one outgoing red edge, one outgoing blue edge, a possibly empty set of incoming red edges, a possibly empty set of incoming blue edges.
(ii) All non-base edges at v_1 (v_2) are incoming at v_1 (v_2) and colored red (blue).
(iii) Reversing all edges of color 1 gives an acyclic graph; e.g., every monochromatic path ends at either v_1 or v_2.

The labeling defined above is a special case of the edge labeling in [15], which characterizes contact L-representations with L-shapes in all four rotations.

Theorem 3. *For every graph G with a plane embedding and distinguished outer edge v_1v_2 the following are equivalent:*

(C1) G admits an L-representation with base edge v_1v_2.
(C2) $G = H\setminus S_n$ for some maximally planar graph H and Schnyder realizer (S_1, S_2, S_n).
(C3) G admits an edge labeling with base edge v_1v_2.
(C4) G admits a 2-canonical order with base edge v_1v_2.

Proof. (C1) $\Longrightarrow$ (C2): This is Lemma 2.
(C2) $\Longrightarrow$ (C3): Follows immediately from the definition of a Schnyder realizer.
(C3) $\Longrightarrow$ (C4): Consider an orientation and coloring of $E(G) \setminus v_1v_2$ with the above properties. We do induction on the number of vertices of G. For $|V(G)| = 2$ there

is nothing to show. For $|V(G)| \geq 3$ consider the path $P = x_0, x_1, \ldots, x_k, x_{k+1}$ on the outer face of G not containing the edge v_1v_2, where $x_0 = v_1$ and $x_{k+1} = v_2$. Since the edges x_0x_1 and x_kx_{k+1} are oriented towards x_0 and x_{k+1}, respectively, for some $i \in \{1, \ldots, k\}$ the edges $x_{i-1}x_i$ and x_ix_{i+1} are outgoing at x_i. Since every vertex different from v_1 and v_2 has one outgoing red and one outgoing blue edge, we find a directed red path from x_i to v_1 and a directed blue path from x_i to v_2. No vertex $v \neq x_i$ lies on both of these paths, since otherwise we would have a directed cycle after reversing all red edges. It follows that $x_{i-1}x_i$ is colored red and x_ix_{i+1} is blue. From the local coloring around x_i we see that x_i has no incoming edge. Applying induction to $G \setminus x_i$ we obtain a 2-canonical order of $G \setminus x_i$ and putting x_i at the end of this order gives a 2-canonical order of G.

(C4) $\Longrightarrow$ (C1): This is Lemma 3. □

The remainder of this section deals with the recognition problem of maximal L-graphs. From Theorem 3, every maximal L-graph is necessarily 2-degenerate and planar. Moreover, both planarity [14] and 2-degeneracy can be tested in linear time. For the maximal 2-degeneracy test, we simply iteratively remove a vertex of smallest degree. Clearly, if every vertex removed has degree exactly two, then G is maximal 2-degenerate. The correctness of this method follows from the fact that no pair of degree two vertices are adjacent in a maximal 2-degenerate graph. This test is easily implemented in linear time via a pre-processing bucket sort of the vertices by degree and adjusting the "bucket membership" of each vertex with each vertex deletion. Thus, to recognize maximal L-graphs we will focus on the planar 2-degenerate graphs.

We now demonstrate a linear time test to determine whether G has a 2-canonical order with a given base edge $e = v_1v_2$.

Lemma 4. *Let G be planar 2-degenerate with an edge $e = v_1v_2$. For every vertex v of G, in every 2-degenerate order starting from e, the neighbors of v that precede v are the same. Let $\overrightarrow{G_e}$ denote the orientation of G according to the precedence order with base edge e.*

Suppose we are given an edge $e = v_1v_2$ and need to determine whether G has a 2-canonical order starting from e. We first construct a 2-degenerate order σ. If no such order exists, we reject e. Otherwise, by Lemma 4, we use σ to construct $\overrightarrow{G_e}$.

We initialize the L-representation $\mathbf{L} = \{L_{v_1}, L_{v_2}\}$ where L_{v_1} is the "top-most" L-shape and L_{v_2} is the "right-most" L-shape. We also initialize the admissible vertices A as the vertices that could be added next according to $\overrightarrow{G}$ (i.e., A contains the vertices adjacent to both v_1 and v_2).

We now describe the main loop of our algorithm. Consider any admissible vertex u_1 and let x and y be $u_1's$ neighbors with $L_x, L_y \in \mathbf{L}$. Moreover, let $u_2, ..., u_k$ be the other admissible vertices adjacent to both x and y. Notice that in order to add every L_{u_i}, we need an appropriate visibility between L_x and L_y in $\mathbf{L}$. However, we delay testing this until the end of the algorithm to save time. Observe the following properties of $u_1, \ldots, u_k$. The L-shapes corresponding to these vertices will be "stacked" on top of each other. This means that, if e is the base edge of an L-representation of G, no pair u_i, u_j can belong to the same connected component of $G \setminus \{x, y\}$. Thus, we let H_i be the connected component of $G \setminus \{x, y\}$ which contains u_i. We now consider two cases. First, if (wlog) H_1 contains v_1, then L_{u_1} must be "lowest" L-shape among $L_{u_1}, \ldots, L_{u_k}$ in any representation since it requires a path of L-shapes that reaches L_{v_1} while avoiding L_x and L_y. In particular, for each $i \in \{2, \ldots, k\}$, we need $G_i =$ ($G[H_i \cup \{x, y\}]$ together with the edge xy) to have an L-representation $\mathbf{L_i}$ with xy as the base edge. Moreover, if H_1 does not contain v_1, we also need such an $\mathbf{L_1}$ for G_1. We recursively construct these $\mathbf{L_i}$'s then insert them into $\mathbf{L}$. If any recursive call fails, we know e was not a good base edge for G. If H_1 contained v_1, we add the an L-shape for u_1 to $\mathbf{L}$, and update the admissible vertices with respect to u_1 (note: we don't need to update with respect to $u_2, \ldots, u_k$ since we have already processed their entire connected components). From here we repeat this main loop until we have exhausted all vertices or we have found a contradiction. After exhausting the vertices we check whether our constructed representation is correct. This completes the description of the algorithm and it is easy to see that it runs in polynomial time.

4 Equilateral L-Representations and Related Representations

Every L-representation of G with base edge v_1v_2 induces an edge labeling of G with base edge v_1v_2, by orienting an edge uv from u to v if an endpoint of L_u is contained in the interior of L_v, and coloring it red (blue) if it is the top (right) endpoint of L_u. We say that two L-representations are *equivalent* if they induce the same edge labeling. An L-shape is *equilateral* if its horizontal and vertical leg are of the same length. An *equilateral L-representation* is one with only equilateral L-shapes.

Theorem 4. *Every L-representation has an equivalent equilateral L-representation.*

Proof. For a given L-representation with base edge v_1v_2, consider the induced edge labeling and fix one corresponding 2-canonical order $(v_1, v_2, \ldots, v_n)$. We construct an equivalent L-representation with equilateral L-shapes along this 2-canonical order, i.e., by a variant of the algorithm given in Lemma 3. We maintain the following invariant:

Invariant: *There is a line ℓ of slope -1 that intersects every segment of the outer staircase in an interior point.*

In the beginning we fix the line ℓ arbitrarily – say $\ell = \{(r, 1 - r) \mid r \in \mathbb{R}\}$. We keep ℓ fixed throughout the entire construction. In the base case one can easily define the L-shapes L_{v_1} and L_{v_2} so that all four legs intersect ℓ in an interior point – say L_{v_1} and L_{v_2} have top endpoint $(1, 2)$ and $(3, -1)$, respectively, and right endpoint $(4, -1)$ and $(5, -3)$, respectively; see Fig. 4(a). In general we have an L-representation of $G_i = G[v_1, \ldots, v_i]$ in which the invariant is maintained.

Consider what happens when we insert a new L-shape for v_{i+1}. Let u and v be the two neighbors of v_{i+1} in G_{i+1}. W.l.o.g. u comes before v when going counterclockwise around the outer face of G_i starting at v_1. Let s_u and s_v be the horizontal segment and vertical segment of the outer staircase which are contained in L_u and L_v, respectively. Note that by the invariant, if we would choose the points $\ell \cap s_u$ and $\ell \cap s_v$ as top and right endpoint of the newly inserted L-shape, then this would be equilateral. However, we do not insert $L_{v_{i+1}}$ exactly there as this would break the invariant. Instead, we insert a slightly smaller L-shape in such a way that the corresponding two new segments of the outer staircase intersect ℓ in the interior; see Fig. 4(b). □

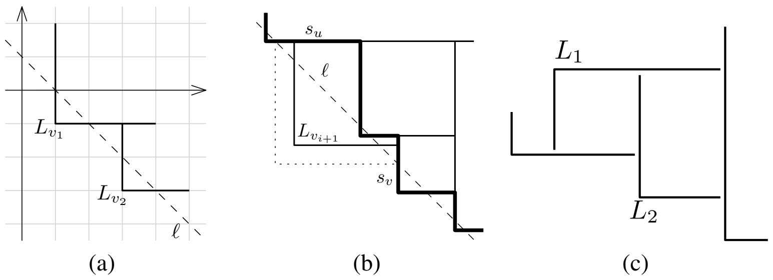


Fig. 4. (a) The definition of L_{v_1} and L_{v_2}. (b) Introducing the L-shape for v_{i+1} maintaining the invariant. (c) A contact L-representation with L-shapes in two different rotations without equivalent equilateral representation for both L_1 and L_2.

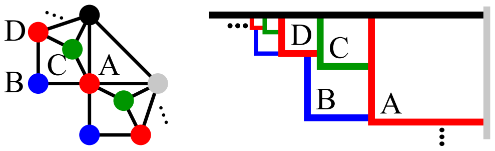


Fig. 5. An equilateral L-representation requiring an exponential size grid. Notice that D is less than half the size of A; i.e., this requires a grid with height $\Omega(2^{n/6})$. Special thanks to an anonymous referee for this observation.

We remark that the equilateral L-representation constructed in Theorem 4 requires an exponential size grid. Moreover, one can show that an exponential sized grid is required for some plane graphs (e.g., see Figure 5). Further we remark that with more than one of the four possible rotations in an L-representation, it is no longer true that every L-representation has an equivalent equilateral one. Consider the L-representation in Fig. 4(c): in every equivalent representation the horizontal leg of L_1 is longer than the horizontal leg of L_2 and the vertical leg of L_1 is shorter than the vertical leg of L_2. Thus L_1 and L_2 cannot be both equilateral.

For a maximally planar graph G with Schnyder realizer (S_1, S_2, S_n) and an inner vertex v we define $\sigma_i(v)$ to be the outgoing neighbor of v in S_i, $i = 1, 2, n$. For convenience, let $\sigma_n(v_1) = \sigma_n(v_2) = \sigma_n(v_n) = v_{n+1}$ for a dummy vertex $v_{n+1} \notin V(G)$.

Definition 3 (cuboid representation). *Let $G = (V, E)$ be a maximally planar graph, (S_1, S_2, S_n) a Schnyder realizer of G, $\{L_v \mid v \neq v_n\}$ an L-representation of $G \setminus S_n$, and $h(v)$ a number for every vertex $v \in V \cup v_{n+1}$. For $v \neq v_n$ let (x^r_v, y^r_v) and (x^t_v, y^t_v) be the right and top endpoint of L_v, respectively. Define an L-shape L_{v_n} with right endpoint $(x^r_{v_n}, y^r_{v_n}) := (x^t_{v_2}, y^r_{v_2})$ and top endpoint $(x^t_{v_n}, y^t_{v_n}) := (x^t_{v_1}, y^r_{v_1})$. Then for every $v \in V$ its* cuboid *is defined as:*

$$Q_v := [x^t_v, x^r_v] \times [y^r_v, y^t_v] \times [h(\sigma_n(v)), h(v)]$$

Note that for any v the projection of Q_v onto the xy-plane gives a rectangle, two sides of which form the L-shape L_v. The number $h(v)$ corresponds to the "height", i.e., z-coordinate, of the top side of the cuboid Q_v; see Fig. 6. A *cuboid representation* of a graph G is a set of interior disjoint cuboids, one for each vertex, so that two cuboids intersect exactly if the corresponding vertices are adjacent in G. A cuboid representation is *proper* if every non-empty intersection of two cuboids is a 2-dimensional rectangle.

Proposition 1. *The cuboids given by Definition 3 form a cuboid representation of G whenever $h(v_{n+1}) < h(v_n)$ and for every inner vertex v of G we have*

$$h(\sigma_1(v)) > h(v) \quad \textit{and} \quad h(\sigma_2(v)) > h(v) \quad \textit{and} \quad h(\sigma_n(v)) < h(v). \tag{1}$$

Further, a non-degenerate L-representation implies a proper cuboid representation.

Proof. Note that conditions (1) imply that along the edges of $S_1 \cup S_2 \cup S_n^{-1}$ the h-values are non-decreasing. It is easy to show that the cuboids for the outer three vertices are mutually touching with proper side contacts. So let uv be an inner edge of G. First assume $v = \sigma_i(u)$, i.e., $uv \in S_i$, for some $i \in \{1, 2\}$. Looking at the L-representation we see that projecting Q_u and Q_v onto the xy-plane gives two rectangles with non-empty intersection or a proper side contact in the non-degenerate case, which is horizontal if $i = 1$ and vertical if $i = 2$. Projecting Q_u and Q_v onto the z-axis gives intervals $[h(\sigma_n(u)), h(u)]$ and $[h(\sigma_n(v)), h(v)]$, respectively. Since there is a directed path from $\sigma_n(v)$ to u in $S_1 \cup S_2 \cup S_n^{-1}$ we get from (1) that $h(\sigma_n(v)) < h(u) < h(v)$. Thus Q_u and Q_v overlap non-trivially.

Next assume $v = \sigma_n(u)$, i.e., $uv \in S_n$. Looking at the L-representation we see that projecting Q_u and Q_v onto the xy-plane gives two rectangles that intersect or overlap non-trivially in the non-degenerate case. Projecting Q_u and Q_v onto the z-axis gives intervals $[h(\sigma_n(u)), h(u)] = [h(v), h(u)]$ and $[h(\sigma_n(v)), h(v)]$, respectively. Thus $Q_u \cap Q_v \neq \emptyset$ or is a rectangle parallel to the xy-plane in the non-degenerate case.

Finally let u and v be non-adjacent. If the rectangles defined by L_u and L_v do not overlap, i.e., can be separated by a horizontal or vertical line, then in 3-space Q_u and Q_v are separated by a plane parallel to the yz-plane or xz-plane. If the rectangles do overlap, there is a path on at least two edges in S_n starting and ending in u and v, respectively. From (1) and the definition of the z-component of cuboids follows that Q_u and Q_v can separated by a plane parallel to the xy-plane. □

Theorem 5. *Planar graphs have proper contact representation by square-based cuboids.*

Proof. As every planar graph is an induced subgraph of some maximally planar graph we may assume w.l.o.g. that $G = (V, E)$ is a maximally planar graph. We fix any Schnyder realizer (S_1, S_2, S_n) of G, consider any non-degenerate equilateral L-representation of $G \setminus S_n$, which exists by Theorem 4. Further we let $(v_1, v_2, \ldots, v_n)$ be any canonical order of G w.r.t. S_1, S_2 and define $h(v_i) = -i$ for $i = 1, \ldots, n$ and $h(v_{n+1}) = -(n + 1)$. Clearly, (1) holds for these h-values. Hence by Proposition 1 the cuboids given by Definition 3 form a proper cuboid representation of G, and since the L-representation is equilateral every cuboid has a square base. □

We remark that a square-based cuboid representation can be found efficiently with an iterative approach, when the L-representation and the cuboids are defined along a single sweep of the chosen canonical order. This approach is illustrated in Fig. 6.

Next we address the question when the cuboids from Definition 3 are actually cubes. This is clearly the case exactly if the chosen L-representation is equilateral and for every vertex v we set $h(v) = h(\sigma_n(v)) + |L_v|$, where $|L_v|$ is the length of a leg of L_v.

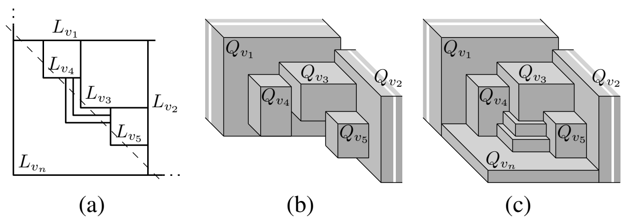


Fig. 6. (a) An equilateral L-representation of $G \setminus S_n$ together with an L-shape for the vertex v_n. (b)–(c) The cuboids can be defined along a canonical order w.r.t. S_1, S_2: The projection of each Q_v onto the xy-plane is a rectangle spanned by L_v. The maximum and minimum z-coordinate of Q_v is given by (the negative of) the index in the canonical order of v and $\sigma_n(v)$, respectively.

For a given equilateral L-representation we call this set of h-values the *cubic heights*. We remark that in any L-representation we can choose the vertical leg of L_{v_1} and the horizontal leg of L_{v_2} (keeping the rest unchanged), so that L_{v_1} and L_{v_2} are equilateral. The cubic heights clearly satisfy $h(\sigma_n(v)) < h(v)$, but in general (1) is not satisfied and we are not guaranteed by Proposition 1 to obtain a cuboid representation. However, as we show next we can sometimes choose the equilateral L-representation (and implicitly the Schnyder realizer) more carefully so that (1) is satisfied for the cubic heights.

Consider a fixed L-representation and let P be the set of all endpoints and bends of L-shapes. For a vertex v let ℓ_v be the line through the top and right endpoint of L_v. A *segment s of an L-shape L_v* is a connected component of $L_v \setminus P$, i.e., $s \subset L_v$, each endpoint of s is a point from P and no further point from P is contained in s. Let $C \subset P$ be the set of contact points between any two L-shapes. We call an L-representation *Square-L, or SL-representation* if for every $p \in C$ the vertical segment whose right end is p and the horizontal segment whose top end is p have the same length; see Fig. 6(a).

Lemma 5. *Consider a maximally planar graph G, a Schnyder realizer (S_1, S_2, S_n), and an SL-representation of $G \setminus S_n$. Then for every $v \in V(G)$ the line ℓ_v has slope -1 and contains the bends of L-shapes corresponding to vertices w with $\sigma_n(w) = v$.*

Proof. Consider any vertex $v \neq v_1, v_2$ and the corresponding L-shape L_v. Let S_v be the staircase that connects the top and right endpoint of L_v and contains the bends of L-shapes corresponding to vertices w with $\sigma_n(w) = v$. If $s_1, \ldots, s_{2k}$ are the segments along S_v, then by assumption s_{2i-1} and s_{2i} are of the same length, $i = 1, \ldots, k$. Equivalently, all bends on S_v lie on ℓ_v, and ℓ_v has slope -1. □

Corollary 1. *Let $\{L_v \mid v \in V\}$ be an SL-representation. Then it is equilateral and $\{\Delta_v := \operatorname{conv}(L_v) \mid v \in V\}$ is a homothetic triangle representation of G. Further, the cubic heights satisfy* (1) *and Proposition 1 yields a contact cube representation of G.*

Not every L-representation has an equivalent SL-representation, since not every planar graph admits a homothetic triangle representation. But homothetic triangle representations exist for 4-connected maximally planar graphs (Theorem 2) and planar 3-trees. Felsner and Francis [10] observed that from Theorem 2 one obtains a cube representation for every planar graph. However, the only proof of Theorem 2 relies on Schramm's result [19], which gives no efficient computation of such representations.

We believe that our interpretation may help to find homothetic triangle representations and hence cube representations efficiently.

5 Conclusions and Open Problems

We investigated L-graphs, provided a characterization, showed relations to Schnyder realizers and canonical orders, and described a recognition algorithm. Moreover, we showed that every L-representation can be transformed into an equivalent equilateral one, thus proving that every planar graph admits a proper contact representation with square-based cuboids, strengthening results by Thomassen [20] and Bremner *et al.* [2]. Finally, we showed that a more restrictive version of equilateral L-representations is equivalent to contact representations with homothetic triangles. Many problems remain, including characterizing contact L-graphs with L's in two or three rotations, and the existence of linear time recognition algorithm for L-graphs.

Acknowledgments. Initial work began at the Barbados Computational Geometry workshop in Feb. 2012. We thank organizers and participants for great discussions and suggestions, and especially S. Felsner, M. Kaufmann, G. Liotta, and T. Mchedlidze.

References

1. Asinowski, A., Cohen, E., Golumbic, M.C., Limouzy, V., Lipshteyn, M., Stern, M.: Vertex intersection graphs of paths on a grid. J. Graph Algorithms Appl. 16(2), 129–150 (2012)
2. Bremner, D., Evans, W., Frati, F., Heyer, L., Kobourov, S.G., Lenhart, W.J., Liotta, G., Rappaport, D., Whitesides, S.H.: On representing graphs by touching cuboids. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 187–198. Springer, Heidelberg (2013)
3. Chalopin, J., Gonçalves, D.: Every planar graph is the intersection graph of segments in the plane. In: 41st ACM Symp. on Theory of Computing (STOC), pp. 631–638 (2009)
4. Chaplick, S., Ueckerdt, T.: Planar graphs as VPG-graphs. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 174–186. Springer, Heidelberg (2013)
5. de Castro, N., Cobos, F., Dana, J., Márquez, A., Noy, M.: Triangle-free planar graphs and segment intersection graphs. J. of Graph Algorithms and Applications 6(1), 7–26 (2002)
6. de Fraysseix, H., de Mendez, P.O., Pach, J.: Representation of planar graphs by segments. Intuitive Geometry 63, 109–117 (1991)
7. de Fraysseix, H., de Mendez, P.O., Rosenstiehl, P.: On triangle contact graphs. Combinatorics, Probability & Computing 3, 233–246 (1994)
8. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10(1), 41–51 (1990)
9. Felsner, S.: Lattice structures from planar graphs. Electronic Journal of Combinatorics 11, R15 (2004)
10. Felsner, S., Francis, M.C.: Contact representations of planar graphs with cubes. In: Proc. 27th Symposium on Computational Geometry, pp. 315–320 (2011)
11. Fusy, É.: Transversal structures on triangulations: A combinatorial study and straight-line drawings. Discrete Mathematics 309(7), 1870–1894 (2009)
12. Gonçalves, D., Lévêque, B., Pinlou, A.: Triangle contact representations and duality. In: Discrete & Computational Geometry, pp. 1–16 (2012)

13. Haas, R., Orden, D., Rote, G., Santos, F., Servatius, B., Servatius, H., Souvaine, D., Streinu, I., Whiteley, W.: Planar minimally rigid graphs and pseudo-triangulations. Computational Geometry 31, 31–61 (2005)
14. Hopcroft, J., Tarjan, R.: Efficient planarity testing. J. ACM 21(4), 549–568 (1974)
15. Kobourov, S., Ueckerdt, T., Verbeek, K.: Combinatorial and geometric properties of planar Laman graphs. In: 24th Symp. on Discrete Algorithms (SODA), pp. 1668–1678 (2013)
16. Koebe, P.: Kontaktprobleme der konformen Abbildung. Berichte über die Verhandlungen der Sächsischen Akad. der Wissenschaften zu Leipzig. Math.-Phys. Klasse 88, 141–164 (1936)
17. Rosenstiehl, P., Tarjan, R.: Rectilinear planar layouts and bipolar orientations of planar graphs. Discrete & Computational Geometry 1, 343–353 (1986)
18. Schnyder, W.: Embedding planar graphs on the grid. In: Proc. 1st ACM-SIAM Symposium on Discrete Algorithms, pp. 138–148 (1990)
19. Schramm, O.: Combinatorically prescribed packings and applications to conformal and quasiconformal maps. arXiv preprint arXiv:0709.0710 (2007)
20. Thomassen, C.: Interval representations of planar graphs. J. Comb. Theory, Ser. B 40(1), 9–20 (1986)

Parameterized and Approximation Algorithms for the MAF Problem in Multifurcating Trees

Jianer Chen, Jia-Hao Fan, and Sing-Hoi Sze

Texas A&M University, College Station TX 77843, USA
{chen,grantfan,shsze}@cs.tamu.edu

Abstract. We study parameterized algorithms and approximation algorithms for the maximum agreement forest problem, which, for two given leaf-labeled trees, is to find a maximum forest that is a subgraph of both trees. The problem was motivated by the research in phylogenetics. For parameterized algorithms, while the problem is known to be fixed-parameter tractable for binary trees, it was an open problem whether the problem is still fixed-parameter tractable for general trees. We resolve this open problem by developing an $O(3^k n)$-time parameterized algorithm for the problem on general trees. Our techniques on tree structures also lead to a polynomial-time approximation algorithm of ratio 3 for the problem, giving the first constant-ratio approximation algorithm for the problem on general trees.

1 Introduction

The evolutionary relationships between a set of species are usually represented by a phylogenetic tree in which each leaf is labeled by a distinct species. Phylogenetic trees can be constructed by different methods, which often lead to different trees. In order to facilitate the comparison of different phylogenetic trees, several distance metrics have been proposed for measuring their similarity [1,7,9,11,12]. In particular, the *tree-bisection-and-reconnection* (*TBR*) and the *subtree-prune-and-regraft* (*SPR*) distances [2,10,16] correspond to the size of the *maximum agreement forest* (abbr. *MAF*) on unrooted trees [1] and on rooted trees [5], respectively.

While most previous work on MAF is restricted to bifurcating (i.e., binary) trees, the problem and related problems on multifurcating (i.e., general) trees have drawn attention recently. While soft multifurcations correspond to ambiguities during the phylogenetic tree construction process, hard multifurcations represent simultaneous speciation events [3,15,16].

In this paper, we focus on algorithms for the MAF problem on unrooted general trees, which corresponds to the TBR distance on general phylogenetic trees with respect to hard multifurcations [1].

Review on Related Research. The problem to construct an MAF for two unrooted trees is NP-hard and MAX SNP-hard [1,4].

Approximation algorithms have been studied for the problem, mainly on binary trees. An approximation algorithm of ratio 3 for the problem on rooted binary trees was claimed by Hein et al. [11], who also claimed that the MAF problem on rooted binary trees corresponds to the SPR distance. Allen and Steel [1] showed that the claim in [11] on the relationship between MAF and SPR was not true, and, on the other hand, proved that the MAF problem on unrooted binary trees corresponds to the TBR distance. Rodrigues et al. [14] found a subtle error in [11], showed that the algorithm in [11] has ratio at least 4, and presented a new approximation algorithm which they claimed has ratio 3. Bonet et al. [3] provided a counterexample and showed that for the TBR distance, the algorithm in [11] has approximation ratio at least 5 while the algorithm in [14] has approximation ratio at least 4. Using very different methods, Chataigner [6] developed an approximation algorithm of ratio 8 for the TBR distance for two or more binary trees. Recently, Whidden et al. [16,17] presented a linear-time approximation algorithm of ratio 3 for the TBR distance on unrooted binary trees. This is the best known approximation algorithm for the TBR distance on binary trees. For general trees, to our knowledge, there are currently no known approximation algorithms for the TBR distance. For the SPR distance on rooted general trees, Rodrigues et al. [15] developed an approximation algorithm of ratio $d + 1$, where d is the maximum number of children a node in the input trees may have. There is also a line of research on the *maximum acyclic agreement forest* problem on general trees [13].

Parameterized algorithms for the MAF problem, parameterized by the number k of trees in an MAF, have also been studied. A problem is *fixed-parameter tractable* [8] if it is solvable in time $f(k)n^{O(1)}$, where k is the parameter and n is the input size. Allen and Steel [1] showed that the MAF problem on unrooted binary trees, which corresponds to the TBR distance, is fixed-parameter tractable. By branching on inconsistent quartets, Hallett and McCartin [10] developed an algorithm of time $O(4^k k^5 + n^{O(1)})$ for the problem. Whidden and Zeh [17,16] further improved the time complexity to $O(4^k n)$, which is currently the best known parameterized algorithm for the MAF problem on unrooted binary trees. For the MAF problem on rooted binary trees, Bordewich et al. [4] proposed a parameterized algorithm of time $O(4^k k^4 + n^3)$, and Whidden et al. [16,17] improved the time complexity to $O(2.42^k n)$. While there has been significant work showing the fixed-parameter tractability for the MAF problem on binary trees, it has been an open problem posted by several researchers [10,16] whether the MAF problem on general trees is fixed-parameter tractable.

Our Contributions. We study parameterized algorithms and approximation algorithms for the MAF problem on unrooted general trees. Our method is based on careful study of the graph structures that takes advantage of special relationships among sibling leaves in trees. We develop an $O(3^k n)$-time parameterized algorithm for the MAF problem on unrooted general trees, thus showing the fixed-parameter tractability of the problem and resolving the open problem posed in [10,16]. We also present a polynomial-time approximation algorithm of ratio 3 for the MAF problem on unrooted general trees. The ratio matches the best known result for the problem on unrooted binary trees [16,17], but our algorithm keeps the same constant ratio and works for general trees. The only previously known approximation algorithm for the MAF problem on general trees [15] is on rooted trees and has ratio $d+1$, where d is the maximum number of children a node in the trees may have. Our algorithm is the first constant-ratio approximation algorithm for the MAF problem, on unrooted general trees.

2 Preliminaries and Problem Reformulations

All graphs will be undirected. For a vertex v, an edge e, and an edge subset E' in a graph G, denote by $G - v$, $G - e$, and $G - E'$ the graphs obtained from G with v, e, and the edges in E' removed, respectively. All trees in our discussion are unrooted. A *leaf* of a tree is a vertex of degree less than 2. A *forest* is a collection of disjoint trees. A forest $\mathcal{F}$ is *leaf-labeled* over a label-set L if there is a one-to-one mapping from the leaves of $\mathcal{F}$ to the elements of L (with all non-leaf vertices in $\mathcal{F}$ *unlabeled*). The label for a leaf v is denoted by $\ell(v)$. For a subforest $\mathcal{F}'$ of $\mathcal{F}$, denote by $\ell(\mathcal{F}')$ the set of labels for the leaves in $\mathcal{F}'$.

Two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ over the same label-set L are *isomorphic* if there is a bijection function f between the vertex sets of $\mathcal{F}_1$ and $\mathcal{F}_2$ such that any two vertices u and v of $\mathcal{F}_1$ are adjacent if and only if $f(u)$ and $f(v)$ are adjacent in $\mathcal{F}_2$, and the corresponding leaves have the same label. The forests $\mathcal{F}_1$ and $\mathcal{F}_2$ are *homeomorphic* if they become isomorphic after smoothing all degree-2 vertices (*smoothing* a degree-2 vertex v is to replace the vertex v and its incident edges with a new edge connecting the two neighbors of v). Note that if a leaf-labeled forest $\mathcal{F}_1$ is homeomorphic to a subforest of a leaf-labeled forest $\mathcal{F}_2$, then there is a unique subforest of $\mathcal{F}_2$ that is homeomorphic to $\mathcal{F}_1$. Therefore, in this case, without any confusion, we can simply say that the forest $\mathcal{F}_1$ *is a subforest of* $\mathcal{F}_2$. An *agreement forest* for two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ over the same label-set L is a leaf-labeled forest $\mathcal{F}'$ over L such that $\mathcal{F}'$ is a subforest of both $\mathcal{F}_1$ and $\mathcal{F}_2$. A *maximum agreement forest* $\mathcal{F}^*$ (abbr. MAF) for $\mathcal{F}_1$ and $\mathcal{F}_2$ is an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$ such that the *size* of (i.e., the number of trees in) $\mathcal{F}^*$ is minimized over all agreement forests for $\mathcal{F}_1$ and $\mathcal{F}_2$ [10].

The two versions of the MAF problem studied in the current paper are

PARA-MAF. Given two leaf-labeled trees T_1 and T_2 over the same label-set L, and a parameter k, is there an agreement forest of size at most k for T_1 and T_2?

MAX-MAF. Given two leaf-labeled trees T_1 and T_2 over the same label-set L, construct an MAF for T_1 and T_2.

Our algorithms on two leaf-labeled trees T_1 and T_2 will proceed by removing edges in the tree T_2 to construct an agreement forest for T_1 and T_2. Removing edges in T_2 will result in a forest consisting of more than one tree. Therefore, our algorithms will really work on a pair of forests $\mathcal{F}_1$ and $\mathcal{F}_2$. However, the size of an agreement forest $\mathcal{F}'$ for two forests may not properly reflect the complexity of the construction of $\mathcal{F}'$: the size of $\mathcal{F}'$ also heavily depends on the sizes of the given forests $\mathcal{F}_1$ and $\mathcal{F}_2$. Thus, we need a careful reformulation of the problems that allows to apply more accurate analysis on the problem complexity.

An *L-partition* $\mathcal{P} = \{L_1, \ldots, L_r\}$ of a label-set L satisfies $L_i \neq \emptyset$, $\bigcup_{i=1}^{r} L_i = L$, and $L_i \cap L_j = \emptyset$, for all i, j, where each L_i is called a *label-subset.* A label-subset L_i is *unit* if $|L_i| = 1$. For a leaf-labeled forest $\mathcal{F} = \{T_1, \ldots, T_h\}$ over a label-set L, the L-partition $\{\ell(T_1), \ldots, \ell(T_h)\}$ is called the *label-partition* for $\mathcal{F}$. For a subset L' of L, denote by $\mathcal{F}[L']$ the subforest of $\mathcal{F}$ *induced* by L', that is, $\mathcal{F}[L']$ consists of all paths in $\mathcal{F}$ that connect pairs of leaves with labels in L'. An L-partition $\mathcal{P} = \{L_1, \ldots, L_r\}$ *induces* a subforest $\{\mathcal{F}[L_1], \ldots, \mathcal{F}[L_r]\}$ of $\mathcal{F}$ if $\mathcal{F}[L_1]$, ..., $\mathcal{F}[L_r]$ are vertex-disjoint trees in $\mathcal{F}$. An L-partition $\mathcal{P}$ is a *c-cut label-partition* for $\mathcal{F}$, where $c \geq 0$ is an integer, if there exists a minimum set E_c of c edges in $\mathcal{F}$ such that after removing the c edges in E_c, the label-partition of the resulting forest is $\mathcal{P}$. Note that the label-partition for the forest $\mathcal{F}$ is a 0-cut label-partition for $\mathcal{F}$.

An unlabeled vertex in a leaf-labeled forest $\mathcal{F}$ may have degree 2, and our algorithms may delete edges and make an unlabeled vertex to have degree even less than 2. *Contraction* is an operation on an unlabeled vertex v of degree less than 3, defined as follows: (1) if v has degree 2, then smooth v; and (2) if v has degree less than 2, then remove v. Contractions enable us to keep the leaves of a forest always labeled. Note that contracting an unlabeled vertex of degree less than 2 does not change the label-partition for a forest. For contracting a degree-2 vertex v, which replaces v and its two incident edges by a new edge e', it is easy to verify that the forest obtained by removing any one of the two edges incident to v in the old forest and the forest obtained by removing the edge e' in the new forest have the same label-partition. These observations give immediately the following lemma.

Lemma 1. *Let $\mathcal{F}$ be a leaf-labeled forest over a label-set L, and let $\mathcal{F}'$ be the forest obtained from $\mathcal{F}$ by applying an arbitrary sequence of contractions on $\mathcal{F}$. For any integer $c \geq 0$, an L-partition $\mathcal{P}$ is a c-cut label-partition for $\mathcal{F}$ if and only if $\mathcal{P}$ is a c-cut label-partition for $\mathcal{F}'$.*

Lemma 2. *Suppose that $\mathcal{P} = \{L_1, \ldots, L_r\}$ is a c-cut label-partition for a leaf-labeled forest $\mathcal{F}$ consisting of h trees. Then $r = h + c$.*

Lemma 2 directly implies the following corollary, which allows us to characterize agreement forests for two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ in terms of c-cut label-partitions for $\mathcal{F}_2$.

Corollary 1. *For an agreement forest $\mathcal{F}^* = \{T_1, \ldots, T_k\}$ for two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ over the same label-set L, where $\mathcal{F}_2$ consists of h trees, the L-partition $\{\ell(T_1), \ldots, \ell(T_k)\}$ is a $(k-h)$-cut label-partition for $\mathcal{F}_2$.*

When a c-cut label-partition $\mathcal{P}$ for a forest is given, it is easy to find the c edges whose removal results in a forest whose label-partition is $\mathcal{P}$.

Lemma 3. *Let L be the label-set for a forest $\mathcal{F}$, and let $\mathcal{P}$ be an L-partition. Let e be an edge in $\mathcal{F}$ whose removal splits a leaf-labeled tree in $\mathcal{F}$ into two leaf-labeled trees T_1 and T_2 such that no label-subset in $\mathcal{P}$ has labels in both T_1 and*

T_2. Then for any integer $c \geq 1$, $\mathcal{P}$ is a c-cut label-partition for $\mathcal{F}$ if and only if $\mathcal{P}$ is a $(c-1)$-cut label-partition for $\mathcal{F} - e$.

Corollary 1 and Lemma 3 suggest a formulation of the MAF problem in terms of c-cut label-partitions. We say that an L-partition $\mathcal{P}$ *induces* an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$ if the subforest induced by $\mathcal{P}$ in $\mathcal{F}_1$ and the subforest induced by $\mathcal{P}$ in $\mathcal{F}_2$ are homeomorphic. By definition, if an L-partition $\mathcal{P} = \{L_1, \ldots, L_k\}$ induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$, and if $\mathcal{F}_2$ consists of h_2 trees, then $\mathcal{P}$ is a $(k - h_2)$-cut label-partition for $\mathcal{F}_2$. This characterization and Lemma 3 provide a way for a branch-and-search process: once we know that an edge e is not on the path connecting any two leaves whose labels are in the same label-subset in the desired $(k - h_2)$-cut label-partition $\mathcal{P}$ for the forest $\mathcal{F}_2$, we can remove e, and recursively construct a $(k - h_2 - 1)$-cut label-partition in $\mathcal{F}_2 - e$.

Our study uses above characterization on the following problem formulations.

> PARA-MAF'. Given two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ over the same label-set L, and a parameter k, is there a k'-cut label-partition for the forest $\mathcal{F}_2$ that induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$, where $k' \leq k$?
>
> MAX-MAF'. Given two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ over the same label-set L, construct an L-partition $\mathcal{P}$ such that $\mathcal{P}$ is a k-cut label-partition for the forest $\mathcal{F}_2$ and that $\mathcal{P}$ induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$, with k minimized.

An L-partition $\mathcal{P}$ will be called a *solution* for the pair $(\mathcal{F}_1, \mathcal{F}_2)$ of leaf-labeled forests over the label-set L if $\mathcal{P}$ induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$. The *value* of the solution $\mathcal{P}$ is c if $\mathcal{P}$ is a c-cut label-partition for $\mathcal{F}_2$. By this definition, constructing a solution of value c for $(\mathcal{F}_1, \mathcal{F}_2)$ is to find c edges in $\mathcal{F}_2$ whose removal results in a subforest that is an agreement forest for $(\mathcal{F}_1, \mathcal{F}_2)$. In particular, by Lemma 2, a minimum-value solution for $(\mathcal{F}_1, \mathcal{F}_2)$ induces an MAF for $\mathcal{F}_1$ and $\mathcal{F}_2$.

3 Bottommost Sibling Sets and Reduction Rules

By Lemma 1, we will assume that there are no unlabeled vertices of degree less than 3. Moreover, if our algorithms create unlabeled vertices of degree less than 3 during their processing, then we will immediately contract these vertices and work on the resulting forests without unlabeled vertices of degree less than 3.

A tree is a *single-vertex tree* if it is a single vertex (which is a leaf). A tree is a *single-edge tree* if it is a single edge (which contains two leaves). The *parent* of a leaf v in a tree with at least three vertices is the unique unlabeled vertex adjacent to v. Two leaves are *siblings* if they either share the same parent, or are the two leaves of a single-edge tree. A *sibling set* is a set of leaves that are all siblings. A *bottommost sibling set* (abbr. *BSS*) is a maximal sibling set X such that either the degree of their parent is at most $|X| + 1$, or X is the leaf set of a single-edge tree. By definition, the leaf of a single-vertex tree is *not* a BSS. Since we assume an unlabeled vertex has degree at least 3, a BSS contains at least two leaves, and a leaf-labeled tree that is not single-vertex must contain a BSS.

In the rest of this section, we fix two leaf-labeled forests $\mathcal{F}_1$ and $\mathcal{F}_2$ over the same label-set L, and consider their agreement forests. Note that if contraction is not applicable on an agreement forest $\mathcal{F}^*$ for $\mathcal{F}_1$ and $\mathcal{F}_2$, then each vertex in $\mathcal{F}^*$ corresponds to a unique vertex in $\mathcal{F}_1$ as well as to a unique vertex in $\mathcal{F}_2$. Therefore, it makes sense to say that two vertices in $\mathcal{F}^*$ are "adjacent in $\mathcal{F}_1$," or that a vertex in $\mathcal{F}^*$ is "the parent of a leaf in $\mathcal{F}_2$." Moreover, because of the one-to-one mapping between the leaf set and the label-set of a leaf-labeled forest, we can refer to a leaf by its label without confusions. Thus, we may say "a label ℓ is in the tree T in the forest $\mathcal{F}$," or "the parent of the label ℓ is the vertex v."

If $\mathcal{F}_1$ consists of only single-vertex trees, then $\mathcal{F}_1$ itself is the agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$. Therefore, in the following discussion, we assume that $\mathcal{F}_1$ contains at least one tree that is not a single-vertex tree. Thus, $\mathcal{F}_1$ always contains a BSS.

Lemma 4. *Let X_1 be a BSS in $\mathcal{F}_1$, and let $\mathcal{P}$ be a solution for $(\mathcal{F}_1, \mathcal{F}_2)$. Then $\mathcal{P}$ has at most one label-subset L_i that intersects $\ell(X_1)$ and $|L_i| > 1$.*

Let $\mathcal{P}$ be an L-partition, and let Y be a set of leaves in $\mathcal{F}_2$. Denote by $\mathcal{P}_Y$ the L-partition that consists of all label-subsets in $\mathcal{P}$ that do not intersect $\ell(Y)$, plus a label-subset that is the union of all label-subsets in $\mathcal{P}$ that intersect $\ell(Y)$.

Lemma 5. *Let X_1 be a BSS in $\mathcal{F}_1$, and let Y be a sibling set in $\mathcal{F}_2$ with $\ell(Y) \subseteq \ell(X_1)$. Let $\mathcal{P}$ be a solution for the pair $(\mathcal{F}_1, \mathcal{F}_2)$ such that a label-subset in $\mathcal{P}$ contains at least two labels in $\ell(Y)$. Then, the L-partition $\mathcal{P}_Y$ is also a solution for the pair $(\mathcal{F}_1, \mathcal{F}_2)$.*

The following lemma shows that a solution for the pair $(\mathcal{F}_1, \mathcal{F}_2)$ is symmetric with respect to two labels when certain conditions are enforced.

Lemma 6. *Let X_1 be a BSS in $\mathcal{F}_1$, and let Y be a sibling set in $\mathcal{F}_2$ with $\ell(Y) \subseteq \ell(X_1)$. For any solution $\mathcal{P}$ for $(\mathcal{F}_1, \mathcal{F}_2)$, swapping two labels of $\ell(Y)$ in $\mathcal{P}$ also results in a solution for $(\mathcal{F}_1, \mathcal{F}_2)$.*

Now we are able to state our main result in this section.

Theorem 1. *Let X_1 be a BSS in $\mathcal{F}_1$ and let Y be a sibling set in $\mathcal{F}_2$ with $\ell(Y) \subseteq \ell(X_1)$. For any $u_0 \in Y$, there is a maximum agreement forest $\mathcal{F}^*$ for $\mathcal{F}_1$ and $\mathcal{F}_2$ such that either (1) all labels in $\ell(Y)$ are in a single tree in $\mathcal{F}^*$, or (2) each label $\ell(u)$ in $\ell(Y)$, where $u \neq u_0$, is in a single-vertex tree in $\mathcal{F}^*$.*

We now present two reduction rules on a pair $(\mathcal{F}_1, \mathcal{F}_2)$ of leaf-labeled forests.
Reduction Rule 1. If a label ℓ is in a single-vertex tree in one of the forests $\mathcal{F}_1$ and $\mathcal{F}_2$, then remove the edge (if any) incident to the label ℓ in the other.

Lemma 7. *Let $(\mathcal{F}_1', \mathcal{F}_2')$ be the pair produced by Reduction Rule 1 on the pair $(\mathcal{F}_1, \mathcal{F}_2)$, then an L-partition $\mathcal{P}$ is a solution of value c for $(\mathcal{F}_1, \mathcal{F}_2)$ if and only if $\mathcal{P}$ is a solution of value c' for $(\mathcal{F}_1', \mathcal{F}_2')$, where $c' = c - 1$ if an edge in $\mathcal{F}_2$ is removed, and $c' = c$ otherwise.*

Let X be a sibling set in a forest. By *shrinking* X, we mean we delete all leaves in X, and introduce a new leaf v_X with a new label ℓ_X (e.g., we can use $\ell(X)$ for ℓ_X), and let v_X be adjacent to the common neighbor of the leaves in X if such a common neighbor exists.

If the sibling set X is the leaf set of a single-edge tree, then shrinking X gives a single-vertex tree with the vertex v_X. If X is the set of all leaves of a tree with a single non-leaf vertex v, then shrinking X makes v a degree-1 vertex adjacent to the new leaf v_X, and v will be contracted so that the resulting tree again becomes a single-vertex tree with the leaf v_X.

Reduction Rule 2. Let X_1 be a BSS in $\mathcal{F}_1$, and let X_2 be a set of leaves in $\mathcal{F}_2$ such that $\ell(X_1) = \ell(X_2)$. If X_2 is the leaf set of a single-edge tree in $\mathcal{F}_2$, or if X_2 is a sibling set whose parent has degree at most $|X_2| + 1$, then shrink X_1 in $\mathcal{F}_1$, and shrink X_2 in $\mathcal{F}_2$.

Note that after applying Reduction Rule 2, if a vertex v in either $\mathcal{F}_1$ or $\mathcal{F}_2$ is adjacent to the new leaf (there is at most one such vertex), then the degree of v is at most 2. In particular, if v is an unlabeled vertex, then v will be contracted.

Lemma 8. *Let $(\mathcal{F}_1', \mathcal{F}_2')$ be the pair produced by Reduction Rule 2 on the pair $(\mathcal{F}_1, \mathcal{F}_2)$, then the pair $(\mathcal{F}_1, \mathcal{F}_2)$ has a solution of value at most c if and only if the pair $(\mathcal{F}_1', \mathcal{F}_2')$ has a solution of value at most c.*

The following theorem follows from the definitions of Reduction Rules 1-2.

Theorem 2. *For a pair $(\mathcal{F}_1, \mathcal{F}_2)$ of leaf-labeled forests on which Reduction Rules 1-2 are not applicable, (1) a label is in a single-vertex tree in $\mathcal{F}_1$ if and only if it is in a single-vertex tree in $\mathcal{F}_2$; and (2) for any sibling set X_2 in $\mathcal{F}_2$ such that the set X_1 in $\mathcal{F}_1$ with $\ell(X_1) = \ell(X_2)$ is a BSS, the siblings in X_2 must have a parent and the parent has degree at least $|X_2| + 2$.*

4 MAF Is Fixed-Parameter Tractable

Let $(\mathcal{F}_1, \mathcal{F}_2; k)$ be an instance of the PARA-MAF' problem, for which we look for a solution of value at most k for the pair $(\mathcal{F}_1, \mathcal{F}_2)$ of leaf-labeled forests. Because of Lemmas 7 and 8, during our process, we will exhaustively apply Reduction Rules 1-2 on the pair of forests whenever the rules are applicable, and work on the reduced instance (note that by Lemma 7, in case we apply Reduction Rule 1 that removes an edge in $\mathcal{F}_2$, we also decrease the parameter k by 1). An instance is *strongly reduced* if none of the contraction operation and Reduction Rules 1-2 are applicable on the corresponding pair of forests. Therefore, throughout the discussion, we will assume that our instance $(\mathcal{F}_1, \mathcal{F}_2; k)$ is always a strongly reduced instance.

As explained in section 3, we can assume that there is a BSS X_1 in $\mathcal{F}_1$ such that $|X_1| \geq 2$. Let X_2 be the leaf set in $\mathcal{F}_2$ such that $\ell(X_2) = \ell(X_1)$.

Our algorithm is a branch-and-bound process. A branching rule is *safe* if the branching rule on an instance I for PARA-MAF' produces a set S of instances such that I is a yes-instance if and only if at least one of the instances in S is a yes-instance. We will examine all possible cases. For each case, we identify a set of edges, and show that at least one of the identified edges must be the "correct" edge to be removed, which ensures that branching on the identified edges is safe. Also, we say that a branching rule satisfies the recurrence relation $T(k) = T(k_1) + \cdots + T(k_r)$ if on an instance $(\mathcal{F}_1, \mathcal{F}_2; k)$ of PARA-MAF', the branching rule produces r instances $(\mathcal{F}_{1,1}, \mathcal{F}_{1,2}; k_1)$, …, $(\mathcal{F}_{r,1}, \mathcal{F}_{r,2}; k_r)$ for the problem. Moreover, we assume that the function $T(k)$ is non-decreasing. Therefore, if $k \leq k'$ then $T(k) \leq T(k')$.

Case 1. Leaves in X_2 are not in the same tree in $\mathcal{F}_2$.

Branching Rule 1. Let u_2 and v_2 be two leaves in X_2 that are in different trees in $\mathcal{F}_2$, then decrease k by 1, and branch into two ways: $[W1]$ cut u_2 in $\mathcal{F}_2$; and $[W2]$ cut v_2 in $\mathcal{F}_2$.

Lemma 9. *Branching Rule 1 is safe, and satisfies* $T(k) = 2T(k-1)$.

Case 2. X_2 is a sibling set in $\mathcal{F}_2$.
Because the instance $(\mathcal{F}_1, \mathcal{F}_2; k)$ is strongly reduced, by Theorem 2, X_2 has a parent p_2 of degree $\geq |X_2| + 2$.

Branching Rule 2. If X_2 is a sibling set, fix a leaf u_2 in X_2, and let $E_2 = \{e_1, \ldots, e_h\}$, $h \geq 2$, be the set of edges that are incident to the parent p_2 of X_2 but not to any leaf in X_2. Then branch into $h+1$ ways: $[W(0)]$ remove all edges incident to X_2 except the one incident to u_2 and decrease k by $|X_2| - 1$; and, for each $1 \leq i \leq h$, $[W(i)]$ remove all edges in E_2 except e_i and decrease k by $h-1$.

Lemma 10. *Branching Rule 2 is safe, and satisfies* $T(k) = T(k - (|X_2| - 1)) + hT(k - (h-1))$.

Case 3. X_2 contains a sibling set Y_2 with $|Y_2| \geq 2$.
We consider this case when Cases 1-2 do not apply. Thus, X_2 contains a leaf z_2 that is in the same tree but not a sibling of the leaves in Y_2. Fix a u_2 in Y_2. Let P be the path between the parent p_2 of Y_2 and z_2. Let E_2 be the set of edges that are not on the path P but adjacent to a vertex $w \neq p_2$ on P. See Fig. 1(A) in appendix for an illustration. Note that E_2 cannot be empty because the path P consists of at least three vertices and by our assumption of the contraction operation, no unlabeled vertex has degree less than 3.

Branching Rule 3. In Case 3, branch into three ways: $[W1]$ cut all leaves in Y_2 except u_2 and decrease k by $|Y_2| - 1$; $[W2]$ cut z_2 and decrease k by 1; and $[W3]$ cut all edges in E_2 and decrease k by $|E_2|$.

Lemma 11. *Branching Rule 3 is safe and satisfies* $T(k) \leq 3T(k-1)$.

Case 4. No two leaves in X_2 are siblings in $\mathcal{F}_2$.
If none of Cases 1-3 apply, then all leaves in X_2 are in the same tree and no two leaves in X_2 are siblings. We split this case into three subcases based on the size

$|X_1|$ and the number of unlabeled vertices on the path $P = \{u_2, w_1, \ldots, w_h, v_2\}$ between two leaves u_2 and v_2 in X_2, where $h \geq 2$. Let $int(P) = \{w_1, \ldots, w_h\}$.

Subcase 4.1: The path P consists of at least five vertices, i.e., $h \geq 3$.

Branching Rule 4.1 In Subcase 4.1, branch into $h + 2$ ways: $[W1]$ cut u_2 and decrease k by 1; $[W2]$ cut v_2 and decrease k by 1; and, for each $1 \leq i \leq h$, $[W(2+i)]$ cut the edges incident to $int(P) \setminus \{w_i\}$ but not on the path P, and decrease k by the number of edges cut.

Lemma 12. *Branching Rule 4.1 is safe, and satisfies* $T(k) \leq 2T(k-1) + hT(k-(h-1))$.

Note that if $|X_2| \geq 3$ and no two leaves in X_2 are siblings, then there are always two leaves u_2 and v_2 in X_2 such that the path connecting u_2 and v_2 in $\mathcal{F}_2$ satisfies the condition in Subcase 4.1. Therefore, in the following, we will assume that X_2 contains exactly two leaves u_2 and v_2, the path P connecting u_2 and v_2 consists of exactly four vertices, of which two are unlabeled, and $int(P) = \{w_1, w_2\}$. Let E_2 be the set of edges that are incident to either w_1 or w_2 but not on the path P.

Subcase 4.2: $E_2 = \{e_1, \ldots, e_h\}$ with $h \geq 3$, see Fig. 1(B) in appendix.

Branching Rule 4.2 In Subcase 4.2, branch into $h+2$ ways: $[W1]$ cut u_2 and decrease k by 1; $[W2]$ cut v_2 and decrease k by 1; and, for $1 \leq i \leq h$, $[W(2+i)]$ cut all edges in E_2 except e_i and decrease k by $h-1$.

Lemma 13. *Branching Rule 4.2 is safe, and satisfies* $T(k) = 2T(k-1) + hT(k-(h-1))$.

Since u_2 and v_2 are not siblings, the path P has at least two unlabeled vertices. Since an unlabeled vertex has degree at least 3, $|E_2| \geq 2$. Thus, only the case $|E_2| = 2$ has not been covered by the above cases.

Subcase 4.3: $E_2 = \{e_1, e_2\}$, see Fig. 1(C) in appendix.

Branching Rule 4.3 In Subcase 4.3, decrease k by 1, and branch into three ways: $[W0]$ cut u_2 in $\mathcal{F}_2$; $[W1]$ cut e_1 in E_2; and $[W2]$ cut e_2 in E_2.

Lemma 14. *Branching Rule 4.3 is safe, and satisfies* $T(k) = 3T(k-1)$.

An instance $(\mathcal{F}_1, \mathcal{F}_2; k)$ of PARA-MAF' with $k \leq 0$ can be easily handled: if $k < 0$ then it is a no-instance; and if $k = 0$ then it is a yes-instance if and only if $\mathcal{F}_2$ is a subforest of $\mathcal{F}_1$. If $\mathcal{F}_1$ consists of only single-vertex trees, then the MAF for $\mathcal{F}_1$ and $\mathcal{F}_2$ is just $\mathcal{F}_1$. Excluding these cases, the forest $\mathcal{F}_1$ contains a BSS X_1 of at least two leaves. Under the assumptions that $\mathcal{F}_1$ contains a BSS X_1 and that the contraction operation and reduction rules are applied whenever they are applicable, Cases 1-4 cover all possible cases for a given instance $(\mathcal{F}_1, \mathcal{F}_2; k)$. Therefore, our parameterized algorithm just proceeds with each of the cases and applies the corresponding branching rule, as given in Fig. 2 in appendix.

We need the following lemma for the recurrence relations to our algorithm.

Lemma 15. *Let* $T_h(k)$ *be a positive-valued function satisfying* $T_h(k) = 2T_h(k-1) + hT_h(k-(h-1))$, *where* $h \geq 3$ *is a constant. Then* $2^k \leq T_h(k) \leq 3^k$.

We analyze the algorithm **Para-MAF** in the following theorem.

Theorem 3. *The problem* PARA-MAF' *can be solved in time $O(3^k n)$, thus is fixed-parameter tractable.*

It is straightforward to solve the original problem PARA-MAF by using the algorithm **Para-MAF**, where two leaf-labeled trees T_1 and T_2 and a parameter k are given. The problem can be regarded as an instance $(T_1, T_2; k-1)$ of the PARA-MAF' problem where we look for a solution of value at most $k-1$ for (T_1, T_2), i.e., a k'-cut label-partition for the forest (i.e., the tree) T_2, with $k' \leq k-1$, which induces an agreement forest (of $k'+1 \leq k$ trees) for T_1 and T_2.

Corollary 2. *The parameterized problem* PARA-MAF *is fixed-parameter tractable.*

Corollary 2 resolves an open problem posed in the literature [10,17].

5 A Constant-Ratio Approximation Algorithm for MAX-MAF

The analysis in previous sections based on BSS also motivates an approximation algorithm for the MAX-MAF' problem which, on a pair $(\mathcal{F}_1, \mathcal{F}_2)$ of leaf-labeled forests over the same label-set L, looks for a solution of the minimum value, i.e., an L-partition $\mathcal{P}$ that induces an agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$ and is a c-cut label-partition for $\mathcal{F}_2$ with the value c minimized. A solution of the minimum value will be called an *optimal solution*, whose value will be called the *optimal value* for the instance $(\mathcal{F}_1, \mathcal{F}_2)$. Recall that a maximum agreement forest for $\mathcal{F}_1$ and $\mathcal{F}_2$ is induced by an optimal solution for $(\mathcal{F}_1, \mathcal{F}_2)$.

Again, our instances are assumed strongly reduced, which means that none of the contraction operation and Reduction Rules 1-2 are applicable on the instances.

An *edge-removal meta-step* of an algorithm is a collection of consecutive steps that on an instance $(\mathcal{F}_1, \mathcal{F}_2)$ of MAX-MAF' removes a set of edges in $\mathcal{F}_2$.

Definition 1. *An edge-removal meta-step M keeps ratio r if on an instance $(\mathcal{F}_1, \mathcal{F}_2)$ of* MAX-MAF'*, M removes a set E_M of edges in $\mathcal{F}_2$ such that $|E_M| \leq r(c - c')$, where c and c' are the optimal values for the instances $(\mathcal{F}_1, \mathcal{F}_2)$ and $(\mathcal{F}_1, \mathcal{F}_2 - E_M)$, respectively.*

For example, if Reduction Rule 1 removes an edge e in $\mathcal{F}_2$, then it is an edge-removal meta-step that keeps ratio 1 because $|E_M| = |\{e\}| = 1$, and by Lemma 7, the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - e)$ is one less than that for $(\mathcal{F}_1, \mathcal{F}_2)$. Also by definition, a meta-step that neither removes edges in $\mathcal{F}_2$ nor changes the optimal value for the forest pair (e.g., Reduction Rule 1 removes an edge in $\mathcal{F}_1$) keeps ratio r for any $r \geq 0$.

Before we present our algorithm, we observe the following:

Lemma 16. *Let $(\mathcal{F}_1, \mathcal{F}_2)$ be an instance of* MAX-MAF' *and let e_2 be any edge in $\mathcal{F}_2$. Then the optimal value for $(\mathcal{F}_1, \mathcal{F}_2 - e_2)$ is at most the optimal value for $(\mathcal{F}_1, \mathcal{F}_2)$.*

In the rest of this section, we fix an instance $(\mathcal{F}_1, \mathcal{F}_2)$ of MAX-MAF', assuming that $(\mathcal{F}_1, \mathcal{F}_2)$ is strongly reduced and that $\mathcal{F}_1$ contains a BSS X_1. Let X_2 be the leaf set in $\mathcal{F}_2$ with $\ell(X_2) = \ell(X_1)$.

As we did in the parameterized algorithm, we consider different cases based on the structure of X_2 in $\mathcal{F}_2$. For each case, we apply a meta-step that removes a set of edges in $\mathcal{F}_2$ and we verify that the meta-step keeps a ratio at most 3.

Case 1. Leaves in X_2 are not in the same tree in $\mathcal{F}_2$.

Meta-Step 1. Let $u_2, v_2 \in X_2$ be in different trees in $\mathcal{F}_2$, then remove the edges incident to u_2 and v_2.

Lemma 17. *Meta-Step 1 keeps ratio 2.*

Case 2. X_2 is a sibling set in $\mathcal{F}_2$.
Because $(\mathcal{F}_1, \mathcal{F}_2)$ is strongly reduced, by Theorem 2, X_2 has a parent p_2 of degree $\geq |X_2| + 2$. Let E'' be the set of edges that are incident to p_2 but not incident to X_2. We apply the following meta-steps based on the difference between the sizes of X_2 and E''. See Fig. 3(A) in appendix.

Subcase 2.1: $|E''| > |X_2|$.

Meta-Step 2.1. If $|E''| > |X_2|$, then pick any set E_1 of $|X_2| - 1$ edges incident to the leaves in X_2, and pick any set E_2 of $|X_2|$ edges in E'', remove $E_1 \cup E_2$.

Lemma 18. *Meta-Step 2.1 keeps ratio* 3.

Subcase 2.2: $|E''| \leq |X_2|$.

Meta-Step 2.2. If $|E''| \leq |X_2|$, then pick any set E_1' of $|E''| - 1$ edges incident to the leaves in X_2, and remove all edges in $E_1' \cup E''$.

Lemma 19. *Meta-Step 2.2 keeps ratio* 3.

Case 3. X_2 contains a sibling set Y_2 with $|Y_2| \geq 2$.
Because of Case 2, here we assume that X_2 itself is not a sibling set.

Meta-Step 3 Let $u_2, v_2 \in Y_2$, and let w_2 be a leaf in X_2 that is not a sibling of u_2 and v_2. Let e be an edge incident to the parent of w_2 but not on the path between u_2 and w_2. Then remove the edge e, the edge e_u incident to u_2, and the edge e_w incident to w_2. See Fig. 3(B) in appendix.

Lemma 20. *Meta-Step 3 keeps ratio* 3.

When none of the cases 1-3 hold true, all leaves in X_2 are in the same tree in $\mathcal{F}_2$ but no two are siblings. For this last case, we first prove the following lemma.

Lemma 21. *If all leaves in X_2 are in the same tree but no two are siblings, there is a leaf in X_2 whose parent has degree 2 in the induced subforest $\mathcal{F}_2[\ell(X_2)]$.*

Case 4. All leaves in X_2 are in the same tree in $\mathcal{F}_2$ but no two are siblings.

Meta-Step 4 Let $u_2, v_2 \in X_2$ such that the parent p_2 of u_2 has degree 2 in $\mathcal{F}_2[\ell(X_2)]$. Let e be an edge in $\mathcal{F}_2$ that is incident to p_2 but not on the path P_{uv} between u_2 and v_2. Then, cut the edge e, the edge e_u incident to u_2, and the edge e_v incident to v_2. See Fig. 3(C) in appendix.

Lemma 22. *Meta-Step 4 keeps ratio* 3.

We present our approximation algorithm for this problem in Fig. 4 in appendix.

Theorem 4. Apx-MAF *is an approximation algorithm for the* MAX-MAF' *problem that runs in time $O(n^2)$ and has an approximation ratio at most* 3.

The original MAX-MAF problem on two leaf-labeled trees T_1 and T_2 over the label-set L asks to construct a maximum agreement forest for T_1 and T_2. Suppose that a maximum agreement forest for T_1 and T_2 consists of c leaf-labeled trees. Then the L-partition that is an optimal solution for (T_1, T_2) consists of c label-subsets, i.e., the optimal value for (T_1, T_2) is $c-1$. We can apply the **Apx-MAF** algorithm on the instance (T_1, T_2), which will return an L-partition $\mathcal{P}$ that is a solution of value at most $3(c-1)$ for (T_1, T_2). Therefore, the solution $\mathcal{P}$ induces an agreement forest of at most $3c-2$ trees for (T_1, T_2). The ratio $(3c-2)/c < 3$ shows that **Apx-MAF** can also be used as an approximation algorithm for the problem MAX-MAF that has an approximation ratio bounded by 3.

When applied on binary trees, the algorithm **Apx-MAF** with its ratio of 3 matches the best previous known ratio for the problem on binary trees [17]. The only previously known approximation algorithm for the MAF problem on general trees is on rooted trees and has a ratio of $d+1$, where d is the maximum number of children a node in the forests may have [15]. Our algorithm is the first constant-ratio approximation algorithm for the MAF problem on general trees, where the trees are unrooted.

6 Conclusions

We have presented a parameterized algorithm and an approximation algorithm for the MAF problem on unrooted general trees, which corresponds to the TBR distance on multifurcating phylogenetic trees. For general trees, our parameterized algorithm is the first fixed-parameter tractable algorithm and our approximation algorithm is the first constant-ratio approximation algorithm. Our algorithms are based on the concept of a bottommost sibling set in one tree and the structure of the corresponding leaf set in the other tree. The methods based on sibling sets have been used by other researchers [16], but the special structure of the bottommost sibling set enables us to deal with operations on general trees more effectively. The techniques should be useful for the study on parameterized and approximation algorithms for other problems related to phylogenetic similarity, such as those related to the SPR distance, the rooted SPR distance, and the hybridization distance.

References

1. Allen, B., Steel, M.: Subtree transfer operations and their induced metrics on evolutionary trees. Ann. Comb. 5, 2001 (2000)
2. Amir, A., Keselman, D.: Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms. SIAM J. Comput. 26(6), 1656–1669 (1997)
3. Bonet, M., John, K., Mahindru, R., Amenta, N.: Approximating subtree distances between phylogenies. J. Comput. Biol. 13(8), 1419–1434 (2006)
4. Bordewich, M., McCartin, C., Semple, C.: A 3-approximation algorithm for the subtree distance between phylogenies. J. Discrete Alg. 6(3), 458–471 (2008)
5. Bordewich, M., Semple, C.: On the computational complexity of the rooted subtree prune and regraft distance. Ann. Comb. 8(4), 409–423 (2005)
6. Chataigner, F.: Approximating the maximum agreement forest on k trees. Inf. Process. Lett. 93(5), 239–244 (2005)
7. DasGupta, B., He, X., Jiang, T., Li, M., Tromp, J.: On the linear-cost subtree-transfer distance between phylogenetic trees. Algorithmica 25(2-3), 176–195 (1999)
8. Downey, R., Fellows, M.: Parameterized Complexity. Springer, New York (1999)
9. Farach, M., Thorup, M.: Fast comparison of evolutionary trees. In: SODA, pp. 481–488 (1994)
10. Hallett, M., McCartin, C.: A faster FPT algorithm for the maximum agreement forest problem. Theory Comput. Syst. 41(3), 539–550 (2007)
11. Hein, J., Jiang, T., Wang, L., Zhang, K.: On the complexity of comparing evolutionary trees. Discrete Appl. Math. 71(1-3), 153–169 (1996)
12. Hon, W.-K., Lam, T.-W.: Approximating the nearest neighbor interchange distance for evolutionary trees with non-uniform degrees. In: Asano, T., Imai, H., Lee, D.T., Nakano, S.-i., Tokuyama, T. (eds.) COCOON 1999. LNCS, vol. 1627, pp. 61–70. Springer, Heidelberg (1999)
13. Linz, S., Semple, C.: Hybridization in nonbinary trees. IEEE/ACM Transactions on Computational Biology and Bioinformatics 6, 30–45 (2009)
14. Rodrigues, E.M., Sagot, M.-F., Wakabayashi, Y.: Some approximation results for the maximum agreement forest problem. In: Goemans, M.X., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX-RANDOM 2001. LNCS, vol. 2129, pp. 159–169. Springer, Heidelberg (2001)
15. Rodrigues, E., Sagot, M., Wakabayashi, Y.: The maximum agreement forest problem: Approximation algorithms and computational experiments. Theor. Comput. Sci. 374(1-3), 91–110 (2007)
16. Whidden, C., Beiko, R., Zeh, N.: Fixed-parameter and approximation algorithms for maximum agreement forests. CoRR, abs/1108.2664 (2011)
17. Whidden, C., Zeh, N.: A unifying view on approximation and FPT of agreement forests. In: Salzberg, S.L., Warnow, T. (eds.) WABI 2009. LNCS, vol. 5724, pp. 390–402. Springer, Heidelberg (2009)

Linear Separation of Total Dominating Sets in Graphs*

Nina Chiarelli[1] and Martin Milanič[1,2]

[1] University of Primorska, UP FAMNIT, Glagoljaška 8, SI6000 Koper, Slovenia
`nina.chiarelli@student.upr.si`

[2] University of Primorska, UP IAM, Muzejski trg 2, SI6000 Koper, Slovenia
`martin.milanic@upr.si`

Abstract. A total dominating set in a graph is a set of vertices such that every vertex of the graph has a neighbor in the set. We introduce and study graphs that admit non-negative real weights associated to their vertices so that a set of vertices is a total dominating set if and only if the sum of the corresponding weights exceeds a certain threshold. We show that these graphs, which we call total domishold graphs, form a non-hereditary class of graphs properly containing the classes of threshold graphs and the complements of domishold graphs. We present a polynomial time recognition algorithm of total domishold graphs, and obtain partial results towards a characterization of graphs in which the above property holds in a hereditary sense. Our characterization in the case of split graphs is obtained by studying a new family of hypergraphs, defined similarly as the Sperner hypergraphs, which may be of independent interest.

Keywords: total domination, total domishold graph, split graph, dually Sperner hypergraph, threshold hypergraph.

Math. Subj. Class. (2010): 05C69

1 Introduction and Background

A possible approach for dealing with the intractability of a given decision or optimization problem is to identify restrictions on input instances under which the problem can still be solved efficiently. One generic framework for describing a kind of such restrictions for graph problems is the following: Given a graph G, does G admit non-negative integer weights on its vertices (or edges, depending on the problem) and a set T of integers such that a subset X of its vertices (or edges) has property P if and only if the sum of the weights of elements of X belongs to T? Property P can denote any of the desired substructures we

* The research presented in this paper is work towards the Ph.D. of the first author, who is partly co-financed by the European Union through the European Social Fund. Co-financing is carried out within the framework of the Operational Programme for Human Resources Development for 2007-2013, 1. development priorities Promoting entrepreneurship and adaptability; priorities 1. 3: Scholarship Scheme. The work of the second author is supported in part by the Slovenian Research Agency, research program P1-0285 and research projects J1-4010, J1-4021 and N1-0011: GReGAS, supported in part by the European Science Foundation.

are looking for, such as matchings, cliques, stable sets, dominating sets, etc. If weights as above are integer and given with the graph, and the set T is given by a membership oracle, then a dynamic programming algorithm can be employed to find a subset with property P of either maximum or minimum cost (according to some given cost function on the vertices) in $O(nM)$ time and with M calls of the membership oracle, where n is the number of vertices (or edges) of G and M is a given upper bound for T [18].

The advantages of the above framework depend both on the choice of property P and on the constraints (if any) imposed on the structure of the set T. For example, if P denotes the property of being a stable (independent) set and the set T is restricted to be an interval unbounded from below, we obtain the class of *threshold graphs* [5], which is very well understood and admits several characterizations, as well as linear time algorithms for recognition and for several optimization problems [17]. If P denotes the property of being a dominating set and T is an interval unbounded from above, we obtain the class of *domishold graphs* [1], which enjoy similar properties as threshold graphs. On the other hand, if P is the property of being a *maximal* stable set and T is restricted to consist of a single number, we obtain the class of *equistable graphs* [19], for which the recognition complexity is open (see, e.g., [16]), no structural characterization is known, and the maximum size of a stable set in an equistable graph is hard to approximate [18].

As the above examples show, the resulting class of graphs can be either *hereditary* (that is, closed under vertex deletion)—as in the case of threshold or domishold graphs—, or non-hereditary—as in the case of equistable graphs. When the resulting graph class is not hereditary, it is natural to consider the hereditary version of the property, in which the requirement (the existence of weights and the set T) is extended to all induced subgraphs of the given graph.

In this paper, we introduce and study the case when P is the property of being a total dominating set and T is an interval unbounded from above. Given a graph $G = (V, E)$, a *total dominating set* (a TD set, for short) is a subset S of the vertices of G such that every vertex of G has a neighbor in S. For surveys of the literature on the subject of total domination, see [11–14].

Definition 1. *A graph $G = (V, E)$ is said to be* total domishold *(TD for short) if there exists a pair (w, t) where $w : V \to \mathbb{R}_+$ is a weight function and $t \in \mathbb{R}_+$ is a threshold such that for every subset $S \subseteq V$, $w(S) := \sum_{x \in S} w(x) \geq t$ if and only if S is a total dominating set in G. A pair (w, t) as above will be referred to as a* total domishold structure *of G.*

We remark that for convenience, the above definition allows G to have isolated vertices. Every graph with an isolated vertex is total domishold, even though it does not have any TD sets.

Example 1. The complete graph of order n is total domishold. Indeed, a subset $S \subseteq V(K_n)$ is a total dominating set of K_n if and only if S is of size at least two, and consequently the pair $(w, 2)$ where $w(x) = 1$ for all $x \in V(K_n)$ is a total domishold structure of K_n. On the other hand, the 4-cycle C_4 is not a total domishold graph (cf. Proposition 5 in Section 4).

It is easy to see that adding a new vertex to the 4-cycle and connecting it to exactly one vertex of the cycle results in a total domishold graph. Therefore, contrary to the classes of threshold and domishold graphs, the class of total domishold graphs is not hereditary. This motivates the following definition:

Definition 2. *A graph G is said to be* hereditary total domishold *(HTD for short) if every induced subgraph of it is total domishold.*

Our results. We initiate the study of TD and HTD graphs. We identify several operations preserving the class of TD graphs, which, together with results from the literature [1, 5], imply that the class of HTD graphs properly contain the classes of threshold graphs and the complements of domishold graphs (Section 3). We obtain the following partial results towards a characterization of HTD graphs (this is done in Section 4): (1) We identify a set of 13 forbidden induced subgraphs for the class of HTD graphs, which implies that every HTD graphs is a $(1, 2)$-polar chordal graph. (2) Split graphs form a well known class of $(1, 2)$-polar chordal graphs. As our main result, we characterize the HTD split graphs. The characterization is obtained by studying a new family of hypergraphs, defined similarly as the Sperner hypergraphs, which might be of independent interest. Finally, we show that TD graphs can be recognized in polynomial time, and develop a simple polynomial time algorithm to find a minimum total dominating set in a given TD graph (this is done in Section 5).

2 Preliminaries and Notation

Graphs and Graph Classes. A graph G is *chordal* if it does not contain any induced cycle of order at least 4, *split* if its vertex set can be partitioned into a clique and an independent set, and $(1, 2)$-*polar* if it admits a partition of its vertex set into two (possibly empty) parts K and L such that K is a clique and L induces a subgraph of maximum degree at most 1. For a set of graphs $\mathcal{F}$, a graph G is said to be $\mathcal{F}$-*free* (or just F-*free* if $\mathcal{F} = \{F\}$), if it does not contain any induced subgraph isomorphic to a member of $\mathcal{F}$. Every member of $\mathcal{F}$ is said to be a *forbidden induced subgraph* for the (hereditary) set of $\mathcal{F}$-free graphs. The neighborhood of a vertex v in a graph will be denoted by $N_G(v)$, and its closed neighborhood by $N_G[v] := N_G(v) \cup \{v\}$, omitting the subscript G if the graph is clear from the context. A vertex in a graph G is *universal* if it is adjacent to every other vertex in G and *isolated* if it is of degree 0. By $G + H$ we will denote the disjoint union of graphs G and H. The *join* of graphs G and H is the graph obtained from the disjoint union $G + H$ by adding all edges of the form $\{uv \mid u \in V(G),\ v \in V(H)\}$. For a graph G, we denote by $2G$ the disjoint union of two copies of G. As usual, we denote by K_n, P_n and C_n the complete graph, the path and the cycle on n vertices.

Boolean Functions. A Boolean function $f : \{0, 1\}^n \to \{0, 1\}$ is *positive* if $f(x) \leq f(y)$ holds for every two vectors $x, y \in \{0, 1\}^n$ such that $x \leq y$ (that is, $x_i \leq y_i$ for all $i \in \{1, \dots, n\}$). A positive Boolean function $f : \{0, 1\}^n \to \{0, 1\}$

is *threshold* if there exist non-negative real weights $w = (w_1, \dots, w_n)$ and a non-negative real number t such that for every $x \in \{0,1\}^n$, $f(x) = 0$ if and only if $\sum_{i=1}^{n} w_i x_i \leq t$ (see, e.g., [7]). Such a pair (w, t) is called a *separating structure* of f. Every threshold positive Boolean function admits an integral separating structure [7].

Threshold Boolean functions have been characterized by Chow [4] and Elgot [8], as follows. For $k \geq 2$, a positive Boolean function $f : \{0,1\}^n \to \{0,1\}$ is said to be *k-summable* if, for some $r \in \{2, \dots, k\}$, there exist r (not necessarily distinct) false points of f, say, $x^1, x^2, \dots, x^r$, and r (not necessarily distinct) true points of f, say $y^1, y^2, \dots, y^r$, such that $\sum_{i=1}^{r} x^i = \sum_{i=1}^{r} y^i$. (A *false point* of f is an input vector $x \in \{0,1\}^n$ such that $f(x) = 0$; a *true point* is defined analogously.) Function f is said to be *k-asummable* if it is not k-summable, and it is *asummable* if it is k-asummable for all $k \geq 2$.

Theorem 1 (Chow [4], Elgot [8], see also Theorem 9.14 in [7]). *A positive Boolean function f is threshold if and only if it is asummable.*

The problem of determining whether a positive Boolean function given by its complete DNF is threshold is solvable in polynomial time, using dualization and linear programming. This result is summarized in the following theorem.

Theorem 2 (Peled and Simeone [20], see also Theorem 9.16 in [7]). *There exists a polynomial time algorithm that determines, given the complete DNF of a positive Boolean function $f(x_1, \dots, x_n)$, whether f is threshold. If this is the case, the algorithm also computes an integral separating structure of f.*

Hypergraphs. A *hypergraph* is a pair $H = (V, E)$ where V is a finite set of *vertices* and E is a set of subsets of V, called *(hyper)edges* [2]. A hypergraph $H = (V, E)$ is *threshold* if there exist a weight function $w : V \to \mathbb{R}_+$ and a threshold $t \in \mathbb{R}_+$ such that for all subsets $X \subseteq V$, it holds $w(X) \leq t$ if and only if X contains no edge of H [10]. Reformulating the characterization of threshold positive Boolean functions given by Theorem 1 in the language of hypergraphs, we obtain the following characterization of threshold hypergraphs.

Theorem 3 (Chow [4], Elgot [8]). *A hypergraph $H = (V, E)$ is* not *threshold if and only if there exists an integer n with $n \geq 2$ and n (not necessarily distinct) subsets $A_1, \dots, A_n$ of V, each containing an edge of H, and n (not necessarily distinct) subsets $B_1, \dots, B_n$ of V, each containing no edge of H, such that for every vertex $v \in V$,*

$$|\{i \mid v \in A_i\}| = |\{i \mid v \in B_i\}| . \tag{1}$$

For a positive integer n, we will use the notation $[n]$ for the set $\{1, \dots, n\}$.

3 Basic Properties of TD Graphs

In this section, we establish some basic properties of TD graphs.

Proposition 1. *Every graph with an isolated vertex is TD.*

Proof. If G has an isolated vertex, then G does not have any TD sets, and hence the pair $(w, |V(G)| + 1)$, where $w(x) = 1$ for all $x \in V(G)$ is a total domishold structure of G. □

As shown by the 4-cycle, TD graphs are not closed under join. However, they are closed under join with K_1, that is, under adding a universal vertex. This is stated formally in Proposition 2 and proved using the following auxiliary lemma.

Lemma 1. *Every TD graph admits a total domishold structure in which all weights are positive.*

Proof. Let (w, t) be a total domishold structure of a TD graph $G = (V, E)$. The value of

$$\delta = t - \max\{w(S) \mid S \in \mathcal{P}(V) \setminus \mathcal{T}\}\,,$$

where $\mathcal{P}(V)$ denotes the power set of V and $\mathcal{T}$ denotes the set of all total dominating sets of G, is well defined and positive. Let $w' : V \to \mathbb{R}_+ \setminus \{0\}$ and $t' \in \mathbb{R}$ be defined as: $w'(x) = |V|w(x) + \delta/2$ for all $x \in V$, and $t' = |V|t$. We claim that (w', t') is a total domishold structure of G. On the one hand, if $S \in \mathcal{T}$, then $w'(S) = |V|w(S) + \delta|S|/2 \geq |V|t = t'$. On the other hand, if $S \in \mathcal{P}(V) \setminus \mathcal{T}$, then $w(S) + \delta/2 < t$ and consequently $w'(S) = |V|w(S) + \delta|S|/2 \leq |V|(w(S) + \delta/2) < |V|t = t'$. □

Proposition 2. *Let G be a graph, and let G' be the graph obtained from G by adding to it a vertex adjacent to all vertices of G. Then, G is TD if and only if G' is TD.*

Proof. The proof will follow from the observation that the sets $\mathcal{T}$ and $\mathcal{T}'$ of total dominating sets of G and G', respectively, are related as follows:

$$\mathcal{T}' = \mathcal{T} \cup \{\{v\} \cup S \mid \emptyset \neq S \subseteq V(G)\}\,,$$

where v is the added vertex.

Suppose first that G is TD. By Lemma 1, G admits a total domishold structure (w, t) with $w(x) > 0$ for all $x \in V(G)$. Let $w' : V(G') \to \mathbb{R}_+$ be defined as follows: for all $x \in V(G)$, let $w'(x) = w(x)$; let $w'(v) = t - \min\{w(x) \mid x \in V(G)\}$. We claim that (w', t) is a total domishold structure of G'. Indeed, if $S \in \mathcal{T}'$ then we consider two cases. If $v \notin S$, then $S \in \mathcal{T}$ and $w'(S) = w(S) \geq t$. If $v \in S$, then $\{x, v\} \subseteq S$ for some $x \in V(G)$, and hence $w'(S) \geq w'(x) + w'(v) = w(x) + t - \min\{w(z) \mid z \in V(G)\} \geq t$. Similarly, if $w'(S) \geq t$, we consider two cases. If $v \notin S$, then $w(S) \geq t$ and therefore $S \in \mathcal{T} \subseteq \mathcal{T}'$. If $v \in S$, then $S \cap V(G) \neq \emptyset$ (since otherwise we would have $w'(S) = w'(v) < t$ by the positivity of w), and thus $S \in \mathcal{T}'$.

The other direction is straightforward. Since $\mathcal{T}' \cap \mathcal{P}(V(G)) = \mathcal{T}$, any pair (w, t) such that (w', t) is a total domishold structure of G' and w is the restriction of w' to $V(G)$, is a total domishold structure of G. □

Corollary 1. *Every threshold graph is HTD.*

Proof. Chvátal and Hammer proved in [5] that the class of threshold graphs is hereditary, and that every threshold graph contains either an isolated vertex or a universal vertex. Therefore, an induction on the number of vertices together with Propositions 1 and 2 shows that every threshold graph is TD. Since the class of threshold graphs is hereditary, every threshold graph is HTD. □

In general, TD graphs are not closed under disjoint union: the path P_3 is TD, but the graph $2P_3$ is not (cf. Proposition 5 in Section 4). However, they are closed under adding a (TD) graph with a *unique* (inclusion-wise) minimal TD set. Due to space limitations, the proof of the following proposition is omitted.

Proposition 3. *Let G and H be graphs such that H has a unique minimal TD set. Then, $G + H$ is TD if and only if G is TD.* □

Corollary 2. *Let G be a graph, and let $G' = G + K_2$. Then, G is TD if and only if G' is TD.*

A graph G is said to be *co-domishold* if its complement is domishold. Since threshold graphs are exactly the domishold co-domishold graphs [1,5], the following result generalizes Corollary 1.

Corollary 3. *Every co-domishold graph is HTD.*

Proof. This can be proved similarly as Corollary 1, using Corollary 2 in addition to Propositions 1 and 2, and the facts that: (1) the class of co-domishold graphs is hereditary (this is because the class of domishold graph is hereditary [1]); (2) every co-domishold graph contains either an isolated vertex, a universal vertex, or a connected component isomorphic to K_2 [1]. □

Note that not every HTD graph is co-domishold. For example, the 4-vertex path P_4 is easily verified to be HTD but it is not domishold [1], and hence also not co-domishold.

As observed in the introduction, the set of TD graphs is not hereditary. We now strengthen this observation by showing that the set of TD graphs is not contained in any nontrivial hereditary class of graphs (even if we disallow graphs with isolated vertices).

Proposition 4. *For every graph G there exists a TD graph G' without isolated vertices such that G is an induced subgraph of G'.*

Proof. Let G be a graph. First, add to $G = (V, E)$ a new vertex, say v, and connect v only to isolated vertices of G. Second, add a new private neighbor to every vertex of the resulting graph. Denoting by G' the obtained graph, it is clear that G is an induced subgraph of G'. By construction, the set $V \cup \{v\}$ is the unique minimal total dominating set in G'. Therefore, the pair (w, t), where $w : V(G') \to \mathbb{R}_+$ is given by $w(x) = 1$ if $x \in V \cup \{v\}$ and $w(x) = 0$, otherwise, and $t = |V| + 1$, is a total domishold structure of G'. □

We conclude this section with a characterization of TD graphs in terms of the thresholdness of a derived Boolean function, a characterization that will turn out useful in proofs in later sections.

We first fix some terminology and notations. Given a set V and a binary vector $x \in \{0,1\}^V$, the *support set* of a vector $x \in \{0,1\}^V$ is the set $S(x) = \{v \in V \mid x_v = 1\}$. Also, by $\overline{x}$ we denote the vector $\overline{x} \in \{0,1\}^V$ given by $(\overline{x})_i = 1 - x_i$ for all $i \in V$. Given a graph $G = (V, E)$, its *neighborhood function* is the positive Boolean function $f_G : \{0,1\}^V \to \{0,1\}$ that takes value 1 precisely on vectors $x \in \{0,1\}^V$ whose support set $S(x)$ contains the neighborhood of some vertex of G. Formally, $f_G(x) = \bigvee_{v \in V} \bigwedge_{u \in N(v)} x_u$ for every vector $x \in \{0,1\}^V$. (If $N(v) = \emptyset$ then $\bigwedge_{u \in N(v)} x_u = 1$.)

Lemma 2. *A graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ is total domishold if and only if its neighborhood function f_G is threshold. Moreover, if $(w_1, \ldots, w_n, t)$ is an integral separating structure of f_G, then $(w; \sum_{i=1}^n w_i - t)$ with $w(v_i) = w_i$ for all $i \in [n]$ is a total domishold structure of G.*

Proof. First, recall that a positive Boolean function $f(x_1, \ldots, x_n)$ is threshold if and only if its dual function $f^d(x) = \overline{f(\overline{x})}$ is threshold, and that if $(w_1, \ldots, w_n, t)$ is an integral separating structure of f, then $(w_1, \ldots, w_n, \sum_{i=1}^n w_i - t - 1)$ is a separating structure of f^d [7]. Therefore, it suffices to argue that G is total domishold if and only if f_G^d is threshold.

Let $x \in \{0,1\}^V$ and let $S(x)$ be the support set of x. By definition, $f_G^d(x) = 0$ if and only if $f(\overline{x}) = 1$, which is the case if and only if $V \setminus S$ contains the neighborhood of some vertex. In other words, $f_G^d(x) = 0$ if and only if S is not a total dominating set. Hence, if the dual function f_G^d is threshold with an integral separating structure $(w_1, \ldots, w_n, t)$, then $(w, t+1)$ with $w(v_i) = w_i$ for all $i \in [n]$ is a total domishold structure of G, and conversely, if (w, t) is an integral total domishold structure of G, then $(w_1, \ldots, v_n, t - 1)$ with $w_i = w(v_i)$ for all $i \in [n]$ is a separating structure of f_G^d.

Finally, if $(w_1, \ldots, w_n, t)$ is an integral separating structure of f_G, then $(w_1, \ldots, w_n, \sum_{i=1}^n w_i - t - 1)$ is a separating structure of f_G^d and hence $(w; \sum_{i=1}^n w_i - t)$ with $w(v_i) = w_i$ for all $i \in [n]$ is a total domishold structure of G. □

4 Partial Characterizations of HTD Graphs

In this section, we obtain some partial results towards a characterization of hereditary total domishold graphs. We start by identifying 13 forbidden induced subgraphs for the set of HTD graphs.

Proposition 5. *Every HTD graph is $\{F_1, \ldots, F_{13}\}$-free, where $F_1, \ldots, F_{13}$ are the graphs depicted in Fig. 1.*

Proof. We only need to verify that none of the graphs $F_1, \ldots, F_{13}$ is TD. Arguing by contradiction, assume that there is a graph $F \in \{F_1, \ldots, F_{13}\}$ that is TD.

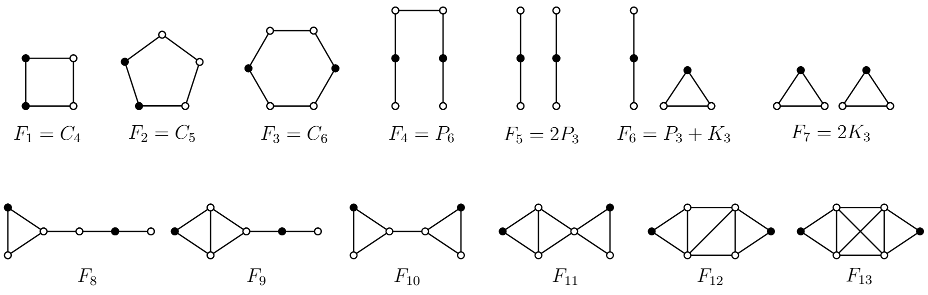


Fig. 1. Graphs $F_1, \ldots, F_{13}$

Take two vertices of degree 2 in F, say a and b, such that a and b have disjoint neighborhoods (e.g., the two black vertices in the depiction of F in Fig. 1). Denote their respective neighbors by a_1, a_2 and b_1, b_2. (If $F \in \{F_1, F_2\}$, then let $b_1 = a$ and $a_2 = b$). It is easy to see that the union of the two neighborhoods $N(a) \cup N(b)$ can be partitioned into two disjoint sets, namely $A = \{a_1, b_1\}$ and $B = \{a_2, b_2\}$, none of which contains the neighborhood of another vertex in the graph. For a set $S \subseteq V(F)$, let $x^S \in \{0,1\}^{V(F)}$ denote the characteristic vector of S, that is,

$$x_i^S = \begin{cases} 1, \text{ if } i \in S; \\ 0, \text{ otherwise.} \end{cases}$$

Clearly, $x^{N(a)}$ and $x^{N(b)}$ are true points of f_F, and x^A and x^B are false points of f_F. Since $x^A + x^B = x^{N(a)} + x^{N(b)}$, the neighborhood function f_F is not 2-asummable. By Theorem 1, function f_F is not threshold, which by Lemma 2 implies that F is not TD, a contradiction. $\square$

Proposition 5 implies a nice structural feature of HTD graphs. Recall that a graph is said to be $(1,2)$-*polar* if it admits a partition of its vertex set into two (possibly empty) parts K and L, such that K is a clique and L induces a subgraph of maximum degree at most 1. The following result is an immediate consequence of Proposition 5 and the forbidden induced subgraph characterization of $(1,2)$-polar graphs due to Gagarin and Metelskiǐ [9] (see also [21]).

Corollary 4. *Every HTD graph is a* $(1,2)$*-polar chordal graph.*

Notice that the converse of Corollary 4 does not hold: graphs $F_8, F_9, \ldots, F_{13}$ are $(1,2)$-polar chordal graphs that are not TD.

In the rest of this section, we give a complete characterization of HTD graphs within a well known subclass of $(1,2)$-polar chordal graphs – the split graphs. The characterization will be based on a new family of hypergraphs, defined similarly as the Sperner hypergraphs. Recall that a hypergraph $H = (V, E)$ is said to be *Sperner* (or: a *clutter*) if no edge of H contains another edge, or, equivalently, if for every two distinct edges e and f of H, it holds that $\min\{|e \setminus f|, |f \setminus e|\} \geq 1$. This motivates the following definition.

Definition 3. *A hypergraph $H = (V, E)$ is said to be* dually Sperner *if for every two distinct edges e and f of H, it holds that*

$$\min\{|e \setminus f|, |f \setminus e|\} \leq 1\,.$$

Lemma 3. *Every dually Sperner hypergraph is threshold.*

Proof. Suppose for a contradiction that there exists a dually Sperner hypergraph $H = (V, E)$ that is not threshold. By Theorem 3, there exists an integer $n \geq 2$ and n (not necessarily distinct) subsets $A_1, \ldots, A_n$ of V, each containing an edge of H, and n (not necessarily distinct) subsets $B_1, \ldots, B_n$ of V, each containing no edge of H, such that for every vertex $v \in V$, condition (1) holds. For every $i \in [n]$, let e_i be an edge of H contained in A_i. Let $i^* \in [n]$ be such that $|e_{i^*}| \leq |e_i|$ for all $i \in [n]$. In particular, this implies that for every $i \in [n]$, it holds that $|e_{i^*} \setminus e_i| \leq |e_i \setminus e_{i^*}|$, which, since H is dually Sperner, implies

$$|e_{i^*} \setminus e_i| \leq 1 \tag{2}$$

for every $i \in [n]$. On the other hand, since no B_i contains the edge e_{i^*}, we have, for all $i \in [n]$, the inequality

$$1 \leq |e_{i^*} \setminus B_i|\,. \tag{3}$$

Adding up the inequalities (3) for all $i \in [n]$, we obtain $n \leq \sum_{i\in[n]} |e_{i^*} \setminus B_i|$. This implies the following contradicting chain of equations and inequalities

$$\begin{aligned} n &\leq \sum_{i\in[n]} |e_{i^*} \setminus B_i| = \sum_{i\in[n]} \sum_{v\in e_{i^*}\setminus B_i} 1 = \sum_{v\in e_{i^*}} \sum_{i\,:\,v\notin B_i} 1 = \sum_{v\in e_{i^*}} (n - |\{i \,:\, v \in B_i\}|) \\ &= \sum_{v\in e_{i^*}} (n - |\{i \,:\, v \in A_i\}|) = \sum_{v\in e_{i^*}} \sum_{i\,:\,v\notin A_i} 1 = \sum_{i\in[n]} \sum_{v\in e_{i^*}\setminus A_i} 1 = \sum_{i\in[n]} |e_{i^*} \setminus A_i| \\ &\leq \sum_{i\in[n]} |e_{i^*} \setminus e_i| = \sum_{\substack{i\in[n]\\ i\neq i^*}} |e_{i^*} \setminus e_i| \leq \sum_{\substack{i\in[n]\\ i\neq i^*}} 1 = n - 1\,. \end{aligned}$$

The first equality in the second line follows from condition (1), while the first inequality in the third line follows from the fact that $e_i \subseteq A_i$, which implies $e_{i^*} \setminus A_i \subseteq e_{i^*} \setminus e_i$. The last inequality follows from (2).

This contradiction completes the proof. □

Using Lemma 3, we can now derive the following characterization of split HTD graphs.

Theorem 4. *Let $G = (V, E)$ be a split graph. Then, the following statements are equivalent:*

1. *G is hereditary total domishold.*
2. *G is F_{13}-free (see Fig. 1).*

Proof. The implication (1) $\Rightarrow$ (2) follows immediately from Proposition 5.

For the implication (2) $\Rightarrow$ (1), let $G = (V, E)$ be an F_{13}-free split graph. Since the class of F_{13}-free split graphs is hereditary, it is enough to show that G is total domishold. We prove this by induction on $|V|$. For $|V| = 1$, the graph G is K_1 and hence TD. Let $|V| > 1$. By Proposition 2 and the inductive hypothesis, we may assume that G has no universal vertices. Let $V = K \cup I$ where K is a clique, I is an independent set, and $K \cap I = \emptyset$. By Lemma 2, it suffices to show that the neighborhood function $f_G(x) = \bigvee_{v \in V} \bigwedge_{u \in N(v)} x_u$ is threshold. Notice that since G has no universal vertices, for every vertex $v \in K$ there exists a vertex $u \in I$ such that $N(u) \subseteq N(v)$. In particular, this implies that the neighborhood function of G is logically equivalent to the function $g : \{0,1\}^V \to \{0,1\}$ given by $g(x) = \bigvee_{v \in I} \bigwedge_{u \in N(v)} x_u$. Consider the hypergraph $H = (K, \{N(v) \mid v \in I\})$. Since G is F_{13}-free, H is dually Sperner, and by Lemma 3, H is threshold. Therefore, there exist a weight function $w : K \to \mathbb{R}_+$ and a threshold $t \in \mathbb{R}_+$ such that for all subsets $X \subseteq K$, it holds $w(X) \le t$ if and only if X contains no neighborhood of a vertex in I. Let $w' : V \to \mathbb{R}_+$ be the extension of w that agrees with w on K and assigns 0 to every vertex in I. The definition of g implies that for all $x \in \{0,1\}^V$, we have $g(x) = 0$ if and only if the set $S(x) \cap K$, where $S(x)$ is the support set of K, contains no neighborhood of a vertex in I. Consequently, the pair (w', t) is a separating structure of g, which implies that g is threshold, and so is f_G. By Lemma 2, G is total domishold. $\square$

5 Algorithmic Aspects

In this section, we show that total domishold graphs can be recognized in polynomial time, and examine some algorithmic consequences of this result. A polynomial time algorithm for the recognition of total domishold graphs will be obtained by reducing the problem to the problem of recognizing threshold Boolean functions given by a positive DNF.

Theorem 5. *There exists a polynomial time algorithm for recognizing total domishold graphs. If the input graph G is total domishold, the algorithm also computes an integral total domishold structure of G.*

Proof. Theorem 2 and Lemma 2 imply that the following polynomial time algorithm correctly determines whether G is total domishold, and if this is the case, computes a total domishold structure of it. First, compute the complete DNF ϕ of the neighborhood function f_G of G. More specifically, let $\phi = \bigvee_{S \in \mathcal{N}} \bigwedge_{u \in S} x_u$ where $\mathcal{N}$ is the set of neighborhoods of vertices of G that do not properly contain any other neighborhood. Second, apply the algorithm given by Theorem 2 to the input ϕ. If the algorithm detects that f_G is not threshold, then G is not total domishold. Otherwise, the algorithm will compute an integral separating structure $(w_1, \ldots, w_n, t)$ of f_G, in which case Lemma 2 implies that $(w; \sum_{i=1}^n w_i - t)$ with $w(v_i) = w_i$ for all $i \in [n]$ is a total domishold structure of G. $\square$

Let us now examine some consequences of Theorem 5. The *total dominating set problem* is the problem of finding a minimum-sized total dominating set in a given graph. The problem is NP-complete in general, and remains NP-complete even for restricted graph classes such as bipartite graphs or split graphs (see [6]). On the positive side, polynomial time algorithms have been designed for several graph classes (see, e.g., [14, 15] for an overview). With the exception of dually chordal graphs [3] and DDP-graphs [14], all known polynomial time algorithms for the total dominating set problem (we are aware of) deal with hereditary graph classes. The following result provides another example of a non-hereditary graph class for which the problem is polynomial.

Proposition 6. *The total dominating set problem is solvable in polynomial time for total domishold graphs.*

Proof. Applying Theorem 5, we may assume that the input graph G is given together with an integral total domishold structure (w, t). A greedy approach can be now used to find a minimum-sized total dominating set S: Start with the empty set, $S = \emptyset$, and, as long as $w(S) < t$, keep adding to S vertices according to non-increasing weights. The correctness and the polynomial running time of this algorithm are immediate. □

While the total dominating set problem is NP-complete for chordal graphs (in fact, even for split graphs), Proposition 6 shows that the problem is polynomial in the class of HTD graphs, which, by Corollary 4, is a subclass of chordal graphs.

In conclusion, we would like to mention the following open questions related to total domishold graphs. Is every $\{F_1, \ldots, F_{13}\}$-free graph total domishold? What is the complexity of recognizing HTD graphs?

Note added in proof: Since writing this paper, we have been able to answer the above questions, by proving that every $\{F_1, \ldots, F_{13}\}$-free graph is total domishold.

Acknowledgements. We would like to thank Endre Boros for fruitful discussions and helpful comments, and an anonymous reviewer for useful comments regarding the proof of Proposition 5.

References

1. Benzaken, C., Hammer, P.L.: Linear separation of dominating sets in graphs. In: Bollobás, B. (ed.) Advances in Graph Theory, Annals of Discrete Mathematics, vol. 3, pp. 1–10. North-Holland (1978)
2. Berge, C.: Hypergraphs. North-Holland (1989)
3. Brandstädt, A., Chepoi, V.D., Dragan, F.F.: The algorithmic use of hypertree structure and maximum neighbourhood orderings. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903, pp. 65–80. Springer, Heidelberg (1995)
4. Chow, C.K.: Boolean functions realizable with single threshold devices. Proceedings of the IRE 49, 370–371 (1961)

5. Chvátal, V., Hammer, P.L.: Aggregation of inequalities in integer programming. In: Hammer, P.L., Johnson, E.L., Korte, B.H., et al. (eds.) Studies in Integer Programming (Proc. Worksh. Bonn 1975). Annals of Discrete Mathematics, vol. 1, pp. 145–162. North-Holland, Amsterdam (1975)
6. Corneil, D.G., Stewart, L.: Dominating sets in perfect graphs. Discrete Math. 86, 145–164 (1990)
7. Crama, Y., Hammer, P.L.: Boolean functions: Theory, Algorithms, and Applications. Cambridge University Press (2011)
8. Elgot, C.C.: Truth functions realizable by single threshold organs. In: IEEE Symposium on Switching Circuit Theory and Logical Design, pp. 225–245 (1961)
9. Gagarin, A.V., Metel'skiĭ, Y.M.: Characterization of (1,2)-polar graphs (in Russian) Vestsi Nats. Akad. Navuk Belarusi Ser. Fiz.-Mat. Navuk 3, 107–112 (1999)
10. Golumbic, M.C.: Algorithmic graph theory and perfect graphs. Ann. of Discrete Math., vol. 57. Elsevier (2004)
11. Haynes, T.W., Hedetniemi, S., Slater, P.: Fundamentals of Domination in Graphs. Marcel Dekker (1998)
12. Haynes, T.W., Hedetniemi, S., Slater, P.: Domination in Graphs: Advanced Topics. Marcel Dekker (1998)
13. Henning, M.A.: Recent results on total domination in graphs: A survey. Discrete Math. 309, 32–63 (2009)
14. Henning, M.A., Yeo, A.: Total Domination in Graphs. Springer (2013)
15. Kratsch, D.: Domination and total domination on asteroidal triple-free graphs. Discrete Applied Math. 99, 111–123 (2000)
16. Levit, V.E., Milanič, M., Tankus, D.: On the recognition of k-equistable graphs. In: Golumbic, M.C., Stern, M., Levy, A., Morgenstern, G. (eds.) WG 2012. LNCS, vol. 7551, pp. 286–296. Springer, Heidelberg (2012)
17. Mahadev, N.V.R., Peled, U.N.: Threshold graphs and related topics. Ann. of Discrete Math, vol. 56. North-Holland Publishing Co., Amsterdam (1995)
18. Milanič, M., Orlin, J., Rudolf, G.: Complexity results for equistable graphs and related classes. Ann. Oper. Res. 188, 359–370 (2011)
19. Payan, C.: A class of threshold and domishold graphs: equistable and equidominating graphs. Discrete Math. 29, 47–52 (1980)
20. Peled, U.N., Simeone, B.: Polynomial-time algorithms for regular set-covering and threshold synthesis. Discrete Applied Math. 12, 57–69 (1985)
21. Zverovich, I.E., Zverovich, O.I.: Independent domination in hereditary classes. Theoret. Comput. Sci. 352, 215–225 (2006)

Sparse Square Roots[⋆]

Manfred Cochefert[1], Jean-François Couturier[1], Petr A. Golovach[2], Dieter Kratsch[1], and Daniël Paulusma[3]

[1] Laboratoire d'Informatique Théorique et Appliquée, Université de Lorraine, 57045 Metz Cedex 01, France
{manfred.cochefert,jean-francois.couturier, dieter.kratsch}@univ-lorraine.fr

[2] Department of Informatics, University of Bergen, PB 7803, 5020 Bergen, Norway
petr.golovach@ii.uib.no

[3] School of Engineering and Computing Sciences, Durham University, Science Laboratories, South Road, Durham DH1 3LE, UK
daniel.paulusma@durham.ac.uk

Abstract. We show that it can be decided in polynomial time whether a graph of maximum degree 6 has a square root; if a square root exists, then our algorithm finds one with minimum number of edges. We also show that it is FPT to decide whether a connected n-vertex graph has a square root with at most $n-1+k$ edges when this problem is parameterized by k. Finally, we give an exact exponential time algorithm for the problem of finding a square root with maximum number of edges.

1 Introduction

Squares and square roots are classical concepts in graph theory. The square G^2 of the graph $G = (V_G, E_G)$ is the graph with vertex set V_G such that any two distinct vertices $u, v \in V_G$ are adjacent in G^2 if and only if u and v are of distance at most 2 in G. A graph H is a square root of G if $G = H^2$. Note that there exist graphs with no square root and that there exist graphs with many square roots. The characterization of those graphs that have a square root, or equivalently of those graphs that are the square of a graph, has already been studied in the 1960s. Mukhopadhyay [17] characterized squares of undirected graphs in 1967, whereas Geller [8] did the same for directed graphs in 1968. Neither characterization yields a polynomial time algorithm for recognizing squares. In fact, in 1994, Motwani and Sudan [16] showed that the problem of recognizing whether a given graph has a square root is NP-complete. As we will discuss, this fundamental result triggered a lot of research on the computational complexity of recognizing squares of graphs and computing square roots under the presence of additional structural assumptions. In particular, the following two recognition questions have attracted attention; here $\mathcal{G}$ denotes some fixed graph class.

(1) How hard is it to recognize the graphs that are the square of a graph from $\mathcal{G}$?
(2) How hard is is to recognize the graphs from $\mathcal{G}$ that have a square root?

[⋆] Supported by EPSRC (EP/G043434/1), ERC (267959) and ANR project AGAPE.

Ross and Harary [18] characterized those graphs that are the square of a tree. They proved that if a connected graph G has a tree square root, then this root is unique up to isomorphism. Moreover, they gave an algorithm for determining a tree that is a square root of any graph known to be the square of some tree. Lin and Skiena [13] obtained a linear time algorithm for deciding whether a graph is the square of a tree. They also proved that it can be decided in linear time whether a planar graph has a square root, and their algorithm finds such a square root if it exists. Lau and Corneil [10] gave a polynomial time algorithm for recognizing graphs that are the square of a proper interval graph, and they showed that the following three problems are NP-complete: recognizing the graphs that are the square of a chordal graph, the graphs that are the square of a split graph, and the chordal graphs that have a square root, respectively. Lau [9] gave a polynomial time algorithm that recognizes the graphs that are the square of a bipartite graph. Le and Tuy [11] obtained structural and algorithmic results for squares of block graphs that generalize the aforementioned results for squares of trees. In a later paper [12] they presented a quadratic time algorithm for recognizing the graphs that are the square of a strongly chordal split graph. Recently, Milanic and Schaudt [14] considered two other subclasses of chordal graphs, namely trivially perfect graphs and threshold graphs, and they gave linear time algorithms for recognizing the graphs from these two classes that have a square root. Adamaszek and Adamaszek [1] proved that if a graph has a square root of girth at least 6, then this square root is unique up to isomorphism. Farzad, Lau, Le and Tuy [7] gave a polynomial time algorithm for recognizing the graphs that have a square root of girth at least 6. They also showed that this problem is NP-complete for square roots of girth 4. The latter result was improved by Farzad and Karimi [6], who established the dichotomy by showing that the problem of recognizing the graphs that have a square root of girth 5 is NP-complete.

Our Results. In the first part of our paper (Section 3) we give a polynomial time algorithm that recognizes the graphs of maximum degree 6 that have a square root. If a square root exists, then our algorithm finds one with minimum number of edges and thus solves the following optimization problem for graphs of maximum degree 6.

Minimum Square Root
Input: a graph G and a positive integer s.
Question: does there exist a graph H with at most s edges such that $G = H^2$?

It can be shown that graphs of maximum degree at most 5 that have a square root have bounded pathwidth, which leads to a linear-time recognition algorithm of such graphs. However, this is not the case for graphs of maximum degree at most 6: consider the square of a wall with subdivided edges. Our approach is to preprocess a given graph G of maximum degree at most 6 in order to obtain a graph of bounded pathwidth.

In the second part of our paper (Section 4) we take a parameterized road to square roots; up to our knowledge, this has not been done so far. A problem with input size n and a parameter k is said to be *fixed parameter tractable* (or FPT) if it can be solved in time $f(k) \cdot n^{O(1)}$ for some function f that only depends on k. Because any square root of a connected n-vertex graph G is a connected spanning subgraph of G, every square root of G has at least $n-1$ edges. This means that $s \geq n-1$ for any yes-instance (G, s) of Minimum Square Root. As such, a natural choice for the parameter would be $k = s-(n-1)$. This leads to the following problem that we call the Tree $+k$ Edges Square Root problem: given a graph G and an integer k, has G a square root with at most $n-1+k$ edges? We show that this problem is FPT when parameterized by k.

In the third part of our paper (Section 5) we consider the Maximum Square Root problem, which is the problem of finding a square root with maximum number of edges. We present an exact exponential time algorithm for Maximum Square Root. In Section 5 we also observe that it is FPT to decide whether a square root can be obtained by at most k edge deletions.

2 Preliminaries and Structural Lemmas

We only consider finite undirected graphs without loops and multiple edges. We refer to the textbook by Diestel [4] for any undefined graph terminology. Let G be a graph. We denote the vertex set of G by V_G and the edge set by E_G. The subgraph of G induced by a subset $U \subseteq V_G$ is denoted by $G[U]$. The graph $G-U$ is the graph obtained from G by removing all vertices in U. If $U=\{u\}$, we also write $G-u$. A set S is a separator in a connected graph G if $G-S$ is disconnected. For two disjoint subsets of vertices X, Y in G, a set of vertices S is an *(X,Y)-separator*, if $G-S$ has no path connecting a vertex of X with a vertex of Y. An (X,Y)-separator S is *minimal*, if no proper subset of S is an (X,Y)-separator. The *distance* $\mathrm{dist}_G(u,v)$ between a pair of vertices u and v of G is the number of edges of a shortest path between them. The *open neighborhood* of a vertex $u \in V_G$ is defined as $N_G(u)=\{v \mid uv \in E_G\}$, and its *closed neighborhood* is defined as $N_G[u]=N_G(u) \cup \{u\}$. Two vertices u, v are said to be *true twins* if $N_G[u]=N_G[v]$, and u, v are *false twins* if $N_G(u)=N_G(v)$. A vertex u is *simplicial*, if $N_G(u)$ is a clique. The degree of a vertex $u \in V_G$ is denoted $d_G(u)=|N_G(u)|$. The maximum degree of G is $\Delta(G)=\max\{d_G(v) | v \in V_G\}$. A vertex of degree one is said to be a *pendant* vertex.

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 (called *bags*) of V_G such that the following three conditions hold: i) $\bigcup_{i \in V_T} X_i = V_G$, ii) for each edge $xy \in E_G$, $x, y \in X_i$ for some $i \in V_T$, and iii) for each $x \in V_G$ the set $\{i \mid x \in X_i\}$ induces a connected subtree of T. The *width* of a tree decomposition $(\{X_i \mid i \in V_T\}, T)$ is $\max_{i \in V_T}\{|X_i|-1\}$. The *treewidth* $\mathbf{tw}(G)$ of a graph G is the minimum width over all tree decompositions of G. If T restricted to be a path, then we say that (X,T) is a *path decomposition* of a graph G. The *pathwidth* $\mathbf{pw}(G)$ of G is the minimum width over all path decompositions of G.

In the remainder of this section we give some structural results about sparse square roots. We start with the following observation that we will frequently use.

Observation 1. *Let H be a square root of a connected graph G.*

i) If u is a pendant vertex of H, then u is a simplicial vertex of G.
ii) If u, v are pendant vertices of H adjacent to the same vertex, then u, v are true twins in G.
iii) If u, v are pendant vertices of H adjacent to different vertices, then u and v are not adjacent in G unless $H = K_2$.

We now state a number of lemmas, the proofs of which have been omitted due to space restrictions, although we note that the proof of Lemma 1 is straightforward. Moreover, Lemmas 1 and 2 can also be found implicitly in the paper of Ross and Harary [18]. Because Ross and Harary [18] consider tree square roots, whereas we are concerned with finding general square roots, we cannot apply their results directly, and as such we give explicit statements of these lemmas.

Lemma 1. *Let H be a square root of G. Let $\{u_1, \dots, u_r\} \subseteq V_H$ for some $r \geq 3$ induce a star in H with central vertex u_1. Let $u_3, \dots, u_r$ be pendant and $\{u_2\}$ be a $(\{u_1, u_3, \dots, u_r\}, V_H \setminus \{u_1, \dots, u_r\})$-separator. Then $\{u_1, \dots, u_r\}$ is a clique of G, and $\{u_1, u_2\}$ is a minimal $(\{u_3, \dots, u_r\}, V_G \setminus \{u_1, \dots, u_r\})$-separator of G.*

Lemma 2. *Let $\{u_1, \dots, u_r\}$, $r \geq 3$, be a clique in a connected graph G such that $\{u_1, u_2\}$ is a minimal $(\{u_3, \dots, u_r\}, V_G \setminus \{u_1, \dots, u_r\})$-separator. Let also $\{x_1, \dots, x_p\} = N_G(u_1) \setminus \{u_1, \dots, u_r\}$ and $\{y_1, \dots, y_q\} = N_G(u_2) \setminus \{u_1, \dots, u_r\}$. Let G be a graph having a square root.*

i) For any square root H of G, the following holds: $u_1u_2 \in E_H$ and, either $u_3u_1, .., u_ru_1 \in E_H$, $u_3u_2, \dots, u_ru_2 \notin E_H$, $u_1x_1, \dots, u_1x_p \notin E_H$, and $\{u_2\}$ is a minimal $(\{u_1, u_3, \dots, u_r\}, V_H \setminus \{u_1, \dots, u_r\})$-separator in H or, symmetrically, $u_3u_1, \dots, u_ru_1 \notin E_H$, $u_3u_2, \dots, u_ru_2 \in E_H$, $u_2y_1, \dots, u_2y_q \notin E_H$ and $\{u_1\}$ is a minimal $(\{u_2, .., u_r\}, V_H \setminus \{u_1, .., u_r\})$-separator in H.
ii) If u_1, u_2 are true twins in G, then either $u_1x_1, \dots, u_1x_p \in E_H$ or $u_2y_1, \dots, u_2y_q \in E_H$, G is the union of two complete graphs with vertex sets $\{u_1, \dots, u_r\}$ and $\{u_1, u_2, x_1, \dots, x_p\}$, and G has two isomorphic square roots with edge sets $\{u_1u_2, \dots, u_1u_r\} \cup \{u_2x_1, \dots, u_2x_p\}$ and $\{u_2u_1, u_2u_3, \dots, u_2u_r\} \cup \{u_1x_1, \dots, u_1x_p\}$ respectively.
iii) If $N_G[u_2] \setminus N_G[u_1] \neq \emptyset$, then $u_2u_1, \dots, u_ru_1 \in E_H$, $u_3u_2, \dots, u_ru_2 \notin E_H$, $u_1x_1, \dots, u_1x_p \notin E_H$, and G has a square root such that $\{u_1, \dots, u_r\}$ induces the star with central vertex u_1 such that $u_3, \dots, u_r$ are pendant in H; moreover, this square root can be obtained from any square root of G by the deletion of the edges u_iu_j for $i, j \in \{2, \dots, r\}$, $i \neq j$.

Lemma 3. *Let H be a square root of G. Suppose that H contains the graph F shown in Fig. 1 as a subgraph, $r \geq 3$, $u_4, \dots, u_r$ are pendant vertices of H, $d_H(u_2) = r - 1$, $u_1u_2u_3$ is an induced path in H that is not included in any cycle of length at most 6, $p, q \geq 1$, $\{x_1, \dots, x_p\} = N_H(u_1) \setminus \{u_2\}$ and $\{y_1, \dots, y_q\} = N_H(u_3) \setminus \{u_2\}$. Then $\{u_1, \dots, u_r\}$ is a clique in G such that either $r = 3$ or $\{u_1, u_2, u_3\}$ is a minimal $(\{u_4, \dots, u_r\}, V_G \setminus \{u_1, \dots, u_r\})$ separator in G and the following holds:*

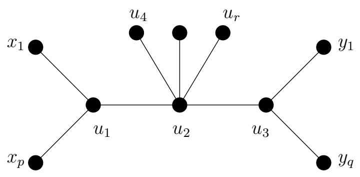


Fig. 1. The graph F

i) $\{x_1,\ldots,x_p\} = N_G(u_1) \cap N_G(u_2) \setminus \{u_3,\ldots,u_r\}$ *and* $\{y_1,\ldots,y_q\} = N_G(u_2) \cap N_G(u_3) \setminus \{u_1,u_4,\ldots,u_r\}$*;*
ii) $d_G(u_2) = p + q + r - 1$*;*
iii) $N_G(u_1) \cap N_G(u_3) = \{u_2,u_4,u_5,\ldots,u_r\}$*;*
iv) $x_1u_3,\ldots,x_pu_3 \notin E_G$*,* $y_1u_1,\ldots,y_qu_1 \notin E_G$ *and* $x_iy_j \notin E_G$ *for* $i \in \{1,\ldots,p\}$ *and* $j \in \{1,\ldots,q\}$*.*

Lemma 4. *Let* $\{u_1,\ldots,u_r\}$*,* $r \geq 3$*, be a clique in a connected graph* G *such that either* $r = 3$ *or* $\{u_1,u_2,u_3\}$ *is a minimal* $(\{u_4,\ldots,u_r\}, V_G \setminus \{u_1,\ldots,u_r\})$*-separator. Assume also that*

i) $\{x_1,\ldots,x_p\} = N_G(u_1) \cap N_G(u_2) \setminus \{u_3,\ldots,u_r\} \neq \emptyset$ *and* $\{y_1,\ldots,y_q\} = N_G(u_2) \cap N_G(u_3) \setminus \{u_1,u_4,\ldots,u_r\} \neq \emptyset$*;*
ii) $d_G(u_2) = p + q + r - 1$*;*
iii) $N_G(u_1) \cap N_G(u_3) = \{u_2,u_4,\ldots,u_r\}$*;*
iv) $x_1u_3,\ldots,x_pu_3 \notin E_G$*,* $y_1u_1,\ldots,y_qu_1 \notin E_G$ *and* $x_iy_j \notin E_G$ *for* $i \in \{1,\ldots,p\}$ *and* $j \in \{1,\ldots,q\}$*.*

Then for any square root H *of* G *(if there is one), the graph* F *shown in Fig. 1 is a subgraph of* H *such that* $d_H(u_2) = r - 1$*,* $\{x_1,\ldots,x_p\} = N_H(u_1) \setminus \{u_2\}$ *and* $\{y_1,\ldots,y_q\} = N_H(u_3) \setminus \{u_2\}$*. Moreover, if* H *is a square root of* G*, then the graph obtained from* H *by deleting the edges* u_iu_j *for* $i,j \in \{4,\ldots,r\}$*,* $i \neq j$*, is a square root of* G*, where* $u_4,\ldots,u_r$ *are pendant vertices in* H*.*

Lemma 5. *Let* u,v *be true twins in a connected graph* G *with at least three vertices. Let also* G' *be the graph obtained from* G *by the deletion of* v*. If* H' *is a square root of* G'*, then the graph* H *obtained from* H' *by adding* v *with* $N_H(v) = N_{H'}(u)$ *(i.e, by adding a false twin of* u*) is a square root of* G*. If* H *is a square root of* G *such that* u,v *are false twins in* H*, then the graph* H' *obtained by the deletion of* v *is a square root of* G'*.*

3 Square Roots for Graphs of Bounded Degree

In this section we show that the Minimum Square Root problem is polynomial-time solvable for graphs of maximum degree at most 6. We start with some additional terminology and lemmas, the proofs of which have been omitted.

Let G be a connected graph, and let $u \in V_G$. We let $L_0(u),\ldots,L_{s(u)}(u)$ denote the levels in the breadth-first search (BFS) from u, that is, $L_i(u) = \{v \in V_G \mid \text{dist}_G(u,v) = i\}$ for $i = 1,\ldots,s(u)$, where $s(u)$ is the number of levels in the decomposition. Hence $L_i = \emptyset$ if $i > s(u)$.

Lemma 6. *For a connected graph G and $u \in V_G$,*

$$\mathbf{pw}(G^2) \leq \max\{|L_i(u) \cup L_{i+1}(u) \cup L_{i+2}(u)| | 0 \leq i \leq s(u)\} - 1.$$

We define the following auxiliary problem.

Minimum Square Root with Labels
Input: a graph G, positive integer s and sets of edges $R, B \subseteq E_G$.
Question: does there exist a graph H with at most s edges such that $G = H^2$, $R \subseteq E_H$ and $B \cap E_H = \emptyset$?

We will use the following lemma.

Lemma 7. *The* Minimum Square Root with Labels *problem can be solved in time $O(f(t)n)$ for n-vertex graphs of treewidth at most t.*

Using Lemma 6 we show that if a square root of G has no induced paths with internal vertices of degree 2 that are parts of short cycles, then G has bounded pathwidth.

Lemma 8. *Let H be a square root of a graph G with $\Delta(G) \leq 6$ such that H has no induced path xyz with $d_H(y) = 2$, $d_H(x) \geq 2$ and $d_H(z) \geq 2$ that is not included in any cycle of length at most 6 in H. Then $\mathbf{pw}(G) \leq 71$.*

The following example shows that we cannot obtain an analog of Lemma 8 for graphs of maximum degree at most 7. Let H' be a cubic graph. We construct a graph H as follows. For each vertex $u \in V_{H'}$ with $N_{H'}(u) = \{v_1, v_2, v_3\}$, u is replaced by three pairwise adjacent vertices u_1, u_2, u_3, and the edges uv_1, uv_2, uv_3 are replaced by u_1v_1, u_2v_2, u_3v_3. We observe that H is cubic and that $\Delta(H^2) = 7$. However, not only $\mathbf{pw}(H^2)$ but also $\mathbf{tw}(H^2)$ is not bounded, because the treewidth of H' can be arbitrary.

We can now prove the main theorem of this section.

Theorem 1. Minimum Square Root *can be solved in time $O(n^5)$ for n-vertex graphs of maximum degree at most 6.*

Proof. The proof is constructive. Our algorithm has two stages. At the first stage we exclude induced paths from square roots with internal vertices of degree two that are not included in short cycles in roots.

Let G be a graph of maximum degree at most 6. We use two sets of edges R and B, and we are trying to find square roots that contain edges of R but that do not contain any edge of B, that is, we are solving Minimum Square Root with Labels. Initially, $R = \emptyset$ and $B = \emptyset$. We recursively apply the following rule. Here, we say that a sequence $u_1, \ldots, u_\ell$ is *maximal* if it cannot be extended by adding new vertices in the beginning or in the end in such a way that conditions i)–v) of step 1 are fulfilled for the modified sequence.

Path reduction rule.

1. Find a maximal sequence of vertices $u_1, \ldots, u_\ell$, $\ell \geq 3$ such that
   i) u_i, u_{i+1}, u_{i+2} are pairwise adjacent for $i \in \{1, \ldots, \ell-2\}$,
   ii) the sets $\{x_1, \ldots, x_p\} = N_G(u_1) \cap N_G(u_2) \setminus \{u_1, u_2, u_3\}$ and $\{y_1, \ldots, y_q\} = N_G(u_{\ell-1}) \cap N_G(u_\ell) \setminus \{u_{\ell-2}, u_{\ell-1}, u_\ell\}$ are not empty,
   iii) $d_G(u_2) = p + q + 2$ if $\ell = 3$, and $d_G(u_2) = p + 3$, $d_G(u_{\ell-1}) = q + 3$, $d_G(u_i) = 4$ for $i \in \{3, \ell-2\}$ if $\ell \geq 4$,
   iv) $N_G(u_i) \cap N_G(u_{i+2}) = \{u_{i+1}\}$ for $i \in \{1, \ldots, \ell-2\}$, and
   v) if $\ell \leq 4$, then $x_1 u_\ell, \ldots, x_p u_\ell \notin E_G$, $y_1 u_1, \ldots, y_q u_1 \notin E_G$, and if $\ell = 3$, then $x_i y_j \notin E_G$ for $i \in \{1, \ldots, p\}$ and $j \in \{1, \ldots, q\}$.
2. Set $R' = \{u_1u_2, u_2u_3, \ldots, u_{\ell-1}u_\ell\} \cup \{x_1u_1, \ldots, x_pu_1\} \cup \{y_1u_\ell, \ldots, y_qu_\ell\}$ and $B' = \{x_1u_2, \ldots, x_pu_2\} \cup \{y_1u_2, \ldots, y_qu_2\} \cup \{u_iu_{i+2} | 1 \leq i \leq \ell-2\} \cup \{zu_1 | z \in N_G(u_1) \setminus \{u_2, u_3, x_1, \ldots, x_p\}\} \cup \{zu_\ell | z \in N_G(u_\ell) \setminus \{u_{\ell-2}, u_{\ell-1}, y_1, \ldots, y_q\}\}$.
3. If $R \cap B' \neq \emptyset$ or $R' \cap B \neq \emptyset$, then stop and return `no`.
4. Delete vertices $u_2, u_4 \ldots, u_\ell$ from G, and also delete the edge u_1u_3 if $\ell = 3$. Set $R = (R \cup R') \cap E_G$ and $B = (B \cup B') \cap E_G$. Set $s = s - \ell + 1$.

To show that the path reduction rule is safe, consider an instance (G, R, B, s) of Minimum Square Root with Labels and assume that $u_1, \ldots, u_\ell$ is a sequence of vertices that satisfies i)–v) of step 1. By Lemma 4, for any square root H of G (if it exists), $R' \subseteq E_H$ and $B' \cap E_H = \emptyset$ for the sets R' and B' constructed at step 2. Hence, if $R \cap B' \neq \emptyset$ or $R' \cap B \neq \emptyset$, then we have a no-answer. Assume that we did not stop at step 3, and denote by $(\hat{G}, \hat{R}, \hat{B}, \hat{s})$ the instance of Minimum Square Root with Labels obtained at step 4. Let H be a solution for (G, R, B, s). Because $R' \subseteq E_H$ and $B' \cap E_H = \emptyset$ by Lemma 4, it is straightforward to check that the graph $\hat{H}$ obtained from H by the deletion of $u_2, \ldots, u_{\ell-1}$ is a solution for $(\hat{G}, \hat{R}, \hat{B}, \hat{s})$. From another side, if $\hat{H}$ is a solution for $(\hat{G}, \hat{R}, \hat{B}, \hat{s})$, then H obtained by joining u_1 and u_ℓ by a path of length $\ell - 1$ is a solution for (G, R, B, s).

We apply the path reduction rule recursively and as long as possible. Assume that we did not stop and returned `no`. To simplify notation, assume that (G, R, B, s) is the obtained instance of Minimum Square Root with Labels. Because we cannot apply the path reduction rule, by Lemma 3, we conclude that for any square root H of G, H has no induced path $u_1u_2u_3$ with $d_H(u_2) = 2$, $d_H(u_1) \geq 2$ and $d_H(u_3) \geq 2$ that is not included in any cycle of length at most 6 in H, as otherwise we could apply the rule for the maximal sequence that includes u_1, u_2, u_3. By Lemma 8, $\mathbf{pw}(G) \leq 71$ if G has a square root. Using Bodlaender's algorithm [3], we check whether $\mathbf{pw}(G) \leq 71$. If $\mathbf{pw}(G) > 71$, then we conclude that we have a no-answer. Otherwise, we solve Minimum Square Root with Labels using Lemma 7.

To conclude the proof, it remains to evaluate the complexity. Each application of the path reduction rule can be done in time $O(n^3 m)$ where m is the number of edges. We can check all triples u_1, u_2, u_3 of pairwise adjacent vertices in time $O(n^3)$. Then we can construct the sets $N_G(u_1) \cap N_G(u_2) \setminus \{u_1, u_2, u_3\}$, $N_G(u_1) \cap N_G(u_2) \setminus \{u_1, u_2, u_3\}$ and check conditions i)–v) in time $O(m)$. Observe that we possibly can extend the sequence $u_1, \dots, u_i$ for $i \geq 3$ only if $N_G(u_{i-1}) \cap N_G(u_i) \setminus \{u_{i-2}, u_{i-1}, u_i\}$ contains exactly one element. Hence, the total time needed to obtain a maximal sequence if u_1, u_2, u_3 are given is $O(m)$. Also the checking whether we have a no-answer at step 3 and construction of the new instance can be done in linear time. As the rule is applied at most n times, we conclude that the total time for this step is $O(n^5)$. It remains to observe that the Bodlaender's algorithm [3] runs in linear time, and Square Root with Labeled Edges is also can be solved in linear time for graphs of bounded treewidth. □

We observe that by the same approach we can solve other variants of square root problems for graphs of maximum degree at most 6. We can find, for example, a square root of maximum size. Also we can count all square roots.

4 The Tree $+k$ Edges Square Root Problem

In this section we prove the following theorem.

Theorem 2. *The* Tree $+k$ Edges Square Root *problem can be solved in time $2^{O(k^4)} + O(n^4 m)$ time on graphs with n vertices and m edges.*

Proof. Due the space restrictions, we only sketch the proof here.

We need the following auxiliary problem:

Tree $+k$ Edges Square Root with Labels
Input: an n-vertex graph G, a non-negative integer k and two subsets of edges $R, B \subseteq E_G$.
Parameter: k.
Question: does there exist a graph H with at most $n + k - 1$ edges such that $G = H^2$, $R \subseteq E_H$ and $B \cap E_H = \emptyset$?

In order to prove the theorem, we reduce Tree $+k$ Edges Square Root to Tree $+k$ Edges Square Root with Labels where the size of the graph in the obtained instance is bounded by a function of k. Then we solve Tree $+k$ Edges Square Root with Labels by a brute force algorithm.

Let G be a connected graph with n vertices and m edges, and let k be a positive integer. First, we check whether G has a tree square root using the algorithm by Lin and Skiena [13], and if we find one, then we stop and return a yes-answer. From now on we assume that any square root of G (if there is one) has cycles. Clearly, connected graphs that have square roots have no cut vertices. Hence, we also check whether G is 2-connected, and stop and return `no` otherwise. We introduce two sets of edges R and B. Initially $R = B = \emptyset$.

As in the algorithm of Lin and Skiena [13], we "trim" pendant edges in potential roots. Since the root we are looking for is not a tree, our trimming rule is more sophisticated and based on Lemmas 1 and 2.

Trimming Rule

1. Find a pair $S = \{u_1, u_2\}$ of two adjacent vertices such that one component of $G - S$ has a set of vertices $\{u_3, \dots, u_r\}$ such that $\{u_1, \dots, u_r\}$ is a clique.

2. If either $N_G[u_1] = N_G[u_2]$ or $N_G[u_1] \setminus N_G[u_2] \neq \emptyset$ and $N_G[u_2] \setminus N_G[u_1] \neq \emptyset$, then stop and return `no`.
3. If $N_G[u_1] \setminus N_G[u_2] \neq \emptyset$, then rename u_1 by u_2 and u_2 by u_1.
4. Set $R' = \{u_1u_2, \ldots, u_1u_r\}$ and $B' = \{u_iu_j | 2 \leq i < j \leq r\} \cup \{u_1x | x \in N_G(u_1) \setminus \{u_2, \ldots, u_r\}\}$.
5. If $R \cap B' \neq \emptyset$ or $R' \cap B \neq \emptyset$, then stop and return `no`. Otherwise, set $R = R \cup R'$, $B = B \cup B'$, delete $u_3, \ldots, u_r$ from G and delete the edges incident to these vertices from R and B.

We apply this rule recursively until we either stop and return `no` or else obtain an instance of Tree $+k$ Edges Square Root Labels such that we cannot apply the rule anymore. Suppose that we did not return `no`. To simplify notations, assume that (G, R, B) is the obtained instance. We need the set R constructed up to now. Let $R_0 = R$. We now apply the following rule, which is based on Lemmas 3 and 4.

Path Reduction Rule

1. Find a triple $S = \{u_1, u_2, u_3\}$ of pairwise adjacent vertices such that
   i) either $N_G(u_1) \cap N_G(u_2) \cap N_G(u_3) = \emptyset$ or $N_G(u_1) \cap N_G(u_2) \cap N_G(u_3) = \{u_3, \ldots, u_r\}$ is a clique of G and S is a $(\{u_4, \ldots, u_r\}, V_G \setminus \{u_1, \ldots, u_r\})$-separator,
   ii) $\{x_1, \ldots, x_p\} = N_G(u_1) \cap N_G(u_2) \setminus \{u_1, \ldots, u_r\}$ and $\{y_1, .., y_q\} = N_G(u_3) \cap N_G(u_2) \setminus \{u_1, \ldots, u_r\}$ are not empty,
   iii) $d_G(u_2) = p + q + r - 1$,
   iv) $N_G(u_1) \cap N_G(u_3) = \{u_2, u_4, \ldots, u_r\}$, and
   v) $x_1u_3, \ldots, x_pu_3 \notin E_G$, $y_1u_1, \ldots, y_qu_1 \notin E_G$ and $x_iy_j \notin E_G$ for $i \in \{1, \ldots, p\}$ and $j \in \{1, \ldots, q\}$.
2. Set $R' = \{u_2u_1, u_2u_3, \ldots, u_2u_r\}$ and $B' = \{x_1u_2, \ldots, x_pu_2\} \cup \{y_1u_2, .., y_qu_2\} \cup \{u_1u_3, \ldots, u_1u_r\} \cup \{u_3u_4, \ldots, u_3u_r\}$.
3. If $R \cap B' \neq \emptyset$ or $R' \cap B \neq \emptyset$, then stop and return `no`.
4. Delete the vertices $u_2, u_4 \ldots, u_r$ from G and delete all edges incident to these vertices from R and B. If $u_1u_3 \in B$, then delete u_1u_3 from B. Include u_1u_3 in R. Modify G by adding edges $x_1u_3, \ldots, x_pu_3$ and $y_1u_1, \ldots, y_qu_1$ in G. Put these edges in B.

We apply the rule recursively and as long as possible. Suppose that we did not stop and return `no`. As before, assume that (G, R, B) is the obtained instance. Recall that R_0 is the set of vertices placed in R by the trimming rule. Let $R_1 = R_0 \cap R$ and $R_2 = R \setminus R_1$. Now we are ready to describe the final reduction rule based on Observation 1 and Lemma 5.

Simplicial Vertex Reduction Rule

1. Find the set S of all simplicial vertices v of G such that v is not incident to the edges of R_2 and if v is incident to an edge of R_1, then all other edges incident to v are in B.
2. If $|V_G \setminus S| > 15k - 14$, then stop and return `no`.

3. Construct the partition $S_1, \dots, S_t$ of S such that any two vertices in each S_i are true twins, and vertices from S_i and S_j are not adjacent if $i \neq j$. Let $X_1, \dots, X_t$ be the sets of vertices incident to the edges of R_1 in $S_1, \dots, S_t$ respectively.
4. If $t > 15k - 14$, then stop and return `no`.
5. If for some $i \in \{1, \dots, t\}$, all the edges of R_1 incident to the vertices of X_i have no common end-point, then stop and return `no`.
6. For each $i \in \{1, \dots, t\}$, if $|X_i| > 1$, then delete arbitrary $|X_i| - 1$ vertices of X_i from G and S_i, and delete the edges of R, B incident to these vertices.
7. For each $i \in \{1, \dots, t\}$, if $|S_i| > 15k-13$, then delete arbitrary $|S_i|-15k+13$ vertices of $S_i \setminus X_i$ from G.

For these rules, we prove that if we stop while executing them, then the problem has no solution. If $(\hat{G}, \hat{R}, \hat{B})$ is the instance obtained by one application of the rules, then $\hat{G}$ is connected, Tree $+k$ Edges Square Root with Labels has a yes-answer for $(\hat{G}, \hat{R}, \hat{B})$ if and only if Tree $+k$ Edges Square Root with Labels has a yes-answer for (G, R, B), and $\hat{G}$ has at most $(15k-14)(15k-12)$ vertices.

To complete the proof of Theorem 2, it remains to solve the obtained reduced instance $(\hat{G}, \hat{R}, \hat{B})$ and evaluate running time. As the obtained graph has at most $(15k-14)(15k-12)$ vertices, it has at most $(15k-14)(15k-12)((15k-14)(15k-12)-1)/2$ edges. Therefore, we can solve Tree $+k$ Edges Square Root with Labels for the obtained instance in time $2^{O(k^4)}$ by brute force checking all edge subsets of size at most $|V_{\hat{H}}| + k - 1$. Now we observe that the trimming and path reduction rules are applied at most n times to construct $(\hat{G}, \hat{R}, \hat{B})$. Each application of the trimming rule can be done in time $O(n^2 m)$ and each application of the path reduction rule takes $O(n^3 m)$. Finally, the simplicial vertex reduction rule can be done in $O(nm)$. Hence, the total running time is $2^{O(k^4)} + O(n^4 m)$. □

Note we reduced Tree $+k$ Edges Square Root to Tree $+k$ Edges Square Root with Labels, i.e., we did not obtain a polynomial kernel. In fact, a polynomial kernel for Tree $+k$ Edges Square Root can be obtained by similar reduction rules, but the obtained graph would have more than $(15k-14)(15k-12)$ vertices.

5 Conclusions

We proved that Tree $+k$ Edges Square Root is FPT when parameterized by k. We also showed that Minimum Square Root can be solved in polynomial time for graphs of maximum degree at most 6. It would be interesting to know whether this degree restriction is tight. Is it possible to solve the problem in polynomial time for graphs of maximum degree at most Δ for some fixed $\Delta \geq 7$? Is there a fixed Δ such that Minimum Square Root is NP-complete for graphs of maximum degree at most Δ? This question is open even if we ask about the existence of any (not necessarily minimum) square root. Another interesting direction of research is to consider square roots of bounded degree. It is trivial to check whether a graph has a square root of maximum degree at most two. Can the existence of a subcubic square root be tested in polynomial time?

Is it possible to construct an exact algorithm for MINIMUM SQUARE ROOT that is better than the trivial exact algorithm for this problem? It can be noted that if we consider MAXIMUM SQUARE ROOT (i.e. we ask about a square root of maximum size), then such an algorithm exists. This algorithm is based on the simple observation that to construct a square root H from a given graph G, for every pair of adjacent edges not belonging to a triangle we have to delete at least one of these edges. Since the structure of the paths in G is crucial, the following auxiliary graph $\mathcal{P}(G)$ with vertex set E_G is useful: for any distinct edges $e_1 = xy$ and $e_2 = yz$ with a common end-point such that $xz \notin E_G$, e_1e_2 is an edge of $\mathcal{P}(G)$. Clearly, for a given graph G, $\mathcal{P}(G)$ is a subgraph of the line graph of G. This leads to the following lemma, the proof of which has been omitted.

Lemma 9. *Let H be a spanning subgraph of G. Then H is a square root of the graph G if and only if E_H is an independent set of $\mathcal{P}(G)$ and for all adjacent vertices u, v in G, u and v are at distance at most 2 in H.*

By using Lemma 9 we obtain an exact exponential time algorithm for MAXIMUM SQUARE ROOT.

Theorem 3. MAXIMUM SQUARE ROOT *can be solved by an exact exponential time algorithm of running time $O^*(3^{m/3})$, where m denotes the number of edges of the input graph.*

Proof. Let G be a graph. We compute the graph $\mathcal{P}(G)$, enumerate all maximal independent sets I of $\mathcal{P}(G)$, and verify for each $I \subseteq E$ whether G is the square of the graph $H_I = (V_G, I)$. Out of those graphs H_I that are square roots of G, return the one with maximum number edges; if no such graph H_I has been found, then G has no square roots. Correctness follows from Lemma 9. The graph $\mathcal{P}(G)$ can be computed in time $O(m^2)$. All the maximal independent sets of the m-vertex graph $\mathcal{P}(G)$ can be enumerated in time $O^*(3^{m/3})$ using the polynomial delay algorithm of Tsukiyama et al. [19], since $\mathcal{P}(G)$ has at most $3^{m/3}$ maximal independent sets [15]. Finally, for each maximal independent set I, we can check in time $O(nm)$ whether $(H_I)^2 = G$. Hence the overall running time of our algorithm is $O^*(3^{m/3})$. □

Lemma 9 also implies that it can be decided in time $O^*(2^k)$ whether a square root of a graph G can be obtained by deleting at most k edges. It is sufficient to check whether $\mathcal{P}(G)$ has a vertex cover C of size at most k such that $H = (V_G, E_G \setminus C)$ is a square root of G. All vertex covers of size at most k of a graph can be enumerated by adapting the standard $O^*(2^k)$ branching algorithm for the vertex cover problem (see e.g. [5]).

Aingworth, Motwani and Harary [2] proved that if H is a square root of a connected n-vertex graph $G \neq K_n$, then $|E_G \setminus E_H| \geq n - 2$. Trivially, a complete graph is its own square root. Hence, we conclude the paper with the following question: is it FPT to decide whether a connected n-vertex graph $G \neq K_n$ has a square root with at least $|E_G| - |V_G| - k + 2$ edges when parameterized by k? In particular, can it be decided in polynomial time whether a connected graph G has a square root with *exactly* $|E_G| - |V_G| + 2$ edges?

References

1. Adamaszek, A., Adamaszek, M.: Uniqueness of graph square roots of girth six. Electr. J. Comb. 18(1) (2011)
2. Aingworth, D., Motwani, R., Harary, F.: The difference between a graph and its square. Util. Math. 54, 223–228 (1998)
3. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25(6), 1305–1317 (1996)
4. Diestel, R.: Graph theory, 4th edn. Graduate Texts in Mathematics, vol. 173. Springer, Heidelberg (2010)
5. Downey, R.G., Fellows, M.R.: Parameterized complexity. Monographs in Computer Science. Springer, New York (1999)
6. Farzad, B., Karimi, M.: Square-root finding problem in graphs, a complete dichotomy theorem. CoRR abs/1210.7684 (2012)
7. Farzad, B., Lau, L.C., Le, V.B., Tuy, N.N.: Complexity of finding graph roots with girth conditions. Algorithmica 62(1-2), 38–53 (2012)
8. Geller, D.P.: The square root of a digraph. J. Combinatorial Theory 5, 320–321 (1968)
9. Lau, L.C.: Bipartite roots of graphs. ACM Transactions on Algorithms 2(2), 178–208 (2006)
10. Lau, L.C., Corneil, D.G.: Recognizing powers of proper interval, split, and chordal graph. SIAM J. Discrete Math. 18(1), 83–102 (2004)
11. Le, V.B., Tuy, N.N.: The square of a block graph. Discrete Mathematics 310(4), 734–741 (2010)
12. Le, V.B., Tuy, N.N.: A good characterization of squares of strongly chordal split graphs. Inf. Process. Lett. 111(3), 120–123 (2011)
13. Lin, Y.L., Skiena, S.: Algorithms for square roots of graphs. SIAM J. Discrete Math. 8(1), 99–118 (1995)
14. Milanic, M., Schaudt, O.: Computing square roots of trivially perfect and threshold graphs. Discrete Applied Mathematics (in press)
15. Moon, J.W., Moser, L.: On cliques in graphs. Israel J. Math. 3, 23–28 (1965)
16. Motwani, R., Sudan, M.: Computing roots of graphs is hard. Discrete Applied Mathematics 54(1), 81–88 (1994)
17. Mukhopadhyay, A.: The square root of a graph. J. Combinatorial Theory 2, 290–295 (1967)
18. Ross, I.C., Harary, F.: The square of a tree. Bell System Tech. J. 39, 641–647 (1960)
19. Tsukiyama, S., Ide, M., Ariyoshi, H., Shirakawa, I.: A new algorithm for generating all the maximal independent sets. SIAM J. Comput. 6(3), 505–517 (1977)

Completing Colored Graphs to Meet a Target Property

Kathryn Cook[1], Elaine M. Eschen[1], R. Sritharan[2], and Xiaoqiang Wang[1]

[1] Lane Dept. of CSEE, West Virginia University, Morgantown, WV 26506
elaine.eschen@mail.wvu.edu, xiaoqiang.wang.wvu@gmail.com
[2] Department of Computer Science, The University of Dayton, Dayton, OH 45469
rsritharan1@udayton.edu

Abstract. We consider the problem of deciding whether a k-colored graph can be completed to have a given property. We establish that, when k is not fixed, the completion problem for circular-arc graphs, even unit or proper circular-arc graphs, is NP-complete. When k is fixed, in the case of completion to circular-arc graphs and Helly circular-arc graphs, we fully classify the complexities of the problems. We also show that deciding whether a 3-colored graph can be completed to be strongly chordal can be done in $O(n^2)$ time. As a corollary of our results, the sandwich problem for Helly circular-arc graphs is NP-complete.

1 Introduction

The graphs we consider are simple and vertex colorings are proper. A *k-colored graph* is a graph properly colored with k colors. In the Π sandwich problem, given graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ on the same vertex-set with $E_1 \subseteq E_2$ and a property Π, the question is whether there is a graph $G = (V, E)$ that has property Π and $E_1 \subseteq E \subseteq E_2$ holds. The set E_1 contains the *required edges*, while the set $E_2 \backslash E_1$ contains the *optional edges*. It is seen that when $E_1 = E_2$ the sandwich problem is the same as the recognition problem for property Π. The sandwich problem was introduced in 1995 by Golumbic et al. in [10] where they studied the problem for the property of membership in several classes of perfect graphs. Since then, the sandwich problem has been studied for a variety of NP properties, and a number of published papers can be found on the topic.

In the *Π completion of a colored graph* problem, given a property Π and a graph $G = (V, E)$ with a proper vertex coloring $c : V \to \mathbb{Z}$, the question is whether there exists a supergraph $G' = (V, E')$ of G that has property Π and is properly colored by c. When such a G' exists, we say G' is a *Π completion of* G and G *admits a Π completion*. By taking the edges in E to be the required edges and $\{xy \mid x \in V, y \in V, xy \notin E, \text{and } c(x) \neq c(y)\}$ to be the set of optional edges, it is seen that the Π completion of a colored graph is a restriction of the sandwich problem for property Π. There are sandwich problems that are NP-complete (such as for comparability graphs), whose corresponding colored graph completion problem is trivial (as every complete k-partite graph is a comparability graph). However, there are also NP-complete sandwich problems whose colored graph completion version remains hard. *Π* completion of colored graphs has been studied inside and outside the context of the sandwich problem.

It was established in [17] that the perfect phylogeny problem is equivalent to problem of chordal completion of colored graphs. It was subsequently shown that this problem is NP-complete [4], which in turn implies that the sandwich problem for chordal graphs is also NP-complete. The sandwich problems for strongly chordal graphs and chordal bipartite graphs were two of the problems whose complexity was left as an open question in [10]. It was shown in [7,15] that these problems are NP-complete; these proofs established that the problems of strongly chordal completion of colored graphs and chordal bipartite completion of colored graphs are NP-complete.

The complexity of chordal completion of colored graphs has been studied when the input graph G is colored with k colors, where k is constant. For the case that $k = 3$, several linear-time algorithms are known [5,11,12]. It is also known that for every fixed k, the problem can be solved in polynomial time [14].

Due to potential applications to the problem of DNA physical mapping [3,9], the problem of interval completion of a colored graph, and its variants, have been studied extensively in the literature. It is known that the problems of interval completion of a colored graph [6] and the unit interval completion of a colored graph [9] are NP-complete. Several results are known for the case of interval and unit interval completions when the input is restricted. Let k be the number of colors used on the input graph. In contrast to the case for chordal graphs, the best known algorithm for interval completion, when $k = 3$, runs in $O(n^2)$ time [3]. In further contrast, the problem has been shown to be NP-complete for every fixed $k \geq 4$ [3]. In fact, the interval completion of a colored graph is NP-complete [1] even for 4-colored caterpillars. Similarly, it is known that the unit interval completion of a colored graph is solvable in polynomial time for caterpillars with hair length less than 2 [2], while it is NP-complete for caterpillars with hair length at least 2 [2].

Our main contributions in this paper are as follows: First, we show that the problems of completing a colored graph to be a circular-arc graph, a proper circular-arc graph, and a unit circular-arc graph are NP-complete. Then, we provide a full classification of the complexity of the problem of completing a colored graph to be a circular-arc graph when the number of colors used is fixed. Specifically, given a k-colored graph, we show that when $k = 2$, there is an $O(n)$-time algorithm for the problem, but when $k = 3$, the problem is NP-complete; in turn, the problem remains NP-complete for every fixed k, $k \geq 3$. We provide an identical classification for the problem of completing a k-colored graph to be a Helly circular-arc graph. To the best of our knowledge, these are the first instances of a colored graph completion problem on 3-colored graphs that are hard. We also show that deciding whether a 3-colored graph admits a strongly chordal completion can be decided in $O(n^2)$ time; our algorithm is based on a characterization of bi-connected 3-colored graphs that admit a strongly chordal completion. We conjecture that the corresponding problem for 4-colored graphs is NP-complete. It is known that the sandwich problems for interval and circular-arc graphs are NP-complete [10]. It follows from our results that the sandwich problem for Helly circular-arc graphs is NP-complete.

We note that the proofs omitted due to space constraints can be found in the full version of the paper.

2 Definitions

For graph $G = (V, E)$, we use $n = |V|$ and $m = |E|$. For $S \subseteq V$, $G[S]$ is the subgraph of G induced by S. Vertex x is *simplicial* if $N(x)$ is a clique. G is *chordal* if every cycle with at least 4 vertices in G has a chord. For $k \geq 1$, a *k-tree* is defined recursively as follows: a K_{k+1} is a k-tree. Given a k-tree T_n on n vertices, a k-tree on $n+1$ vertices can be constructed by adding a new vertex that is adjacent only to every vertex of a clique of size k in T_n. A *partial k-tree* is a subgraph of a k-tree. A sequence $v_1 v_2 \cdots v_n$ of all the vertices of a chordal graph is a *perfect elimination scheme* provided for every i, $1 \leq i < n$, v_i is simplicial in $G[\{v_{i+1}, \cdots, v_n\}]$. An *$n$-sun* is the graph on $2n$ vertices ($n \geq 3$) whose vertex set can be partitioned into $W = \{w_0, \ldots, w_{n-1}\}$ and $U = \{u_0 \ldots, u_{n-1}\}$ such that U is a clique, W is an independent set, and u_i is adjacent to w_j if and only if $i = j$ or $i = j + 1 \pmod{n}$. A graph is *strongly chordal* if it is chordal and does not contain an n-sun, $n \geq 3$. G is an *interval graph* if every $x \in V$ can be mapped to an interval I_x on the real line such that $xy \in E$ if and only if $I_x \cap I_y \neq \emptyset$. G is a *circular-arc graph* if every $x \in V$ can be mapped to an arc A_x on a circle such that $xy \in E$ if and only if $A_x \cap A_y \neq \emptyset$; $\{A_x | x \in V\}$ is a *model* for G. A set of pair-wise intersecting arcs in the model for a circular-arc graph has the *Helly property* if they all have a common point of intersection. A circular-arc graph is a *Helly circular-arc graph* if admits a model in which any set of arcs corresponding to a clique has the Helly property. It is well known that G is a Helly circular-arc graph if and only if there is a circular ordering of the cliques of G with the *consecutive ones property*, i.e., for any vertex x of G, the cliques containing x are consecutive in the ordering.

3 Circular-Arc Completions

3.1 Completing Arbitrarily Colored Graphs

Using the fact that the interval [3] and unit interval [9] completion of a colored graph problems are NP-complete and the following theorem due to Tucker, we prove Theorem 2.

Theorem 1. [16] *Suppose the vertex set of a circular-arc graph G can be partitioned into two cliques. Then in any circular-arc model for G there exist two points p_1 and p_2 such that each circular arc contains at least one of them.*

Theorem 2. *The circular-arc, proper circular-arc, and unit circular-arc completion of a colored graph problems are NP-complete.*

3.2 Completing 2-Colored Graphs

A *short caterpillar* is a tree obtained from a path on k vertices, $k \geq 1$, by and optionally attaching vertices of degree one to the vertices on the path. A *bug* is a graph obtained from a cycle on $2k$ vertices, $k \geq 2$, by optionally attaching vertices of degree one to the vertices on the cycle. A short caterpillar and a bug have unique 2-colorings (up to flipping color classes).

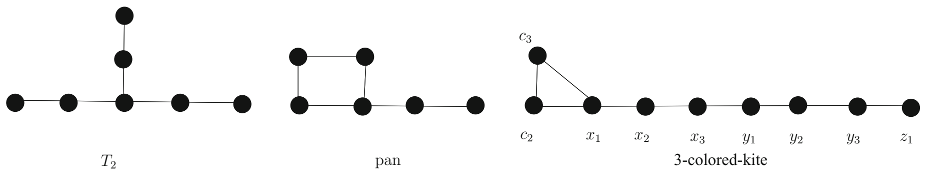


Fig. 1. Three relevant graphs

Observation 1. *The tree T_2 is not a circular-arc graph. Any tree that is not an interval graph must contain T_2 as an induced subgraph. A circular-arc graph cannot contain a pan as a subgraph.*

Lemma 1. *Let G be a 2-colored graph. G is a circular-arc graph if and only if G is either the disjoint union of one or more short caterpillars or G is a bug.*

Theorem 3. *Let G be a 2-colored graph. Then, the following are equivalent:*

(i) G admits a circular-arc completion.
(ii) G admits a Helly circular-arc completion.
(iii) G is either the disjoint union of short caterpillars or G is the disjoint union of a bug with isolated vertices of either color.

Corollary 1. *Whether a 2-colored graph admits a circular-arc completion (or Helly circular-arc completion) can be tested in $O(n)$ time.*

3.3 Completing 3-Colored Graphs

We show that the problem of deciding whether a 3-colored graph G admits a circular-arc completion (or Helly circular-arc completion) is NP-complete by a transformation from the strongly NP-complete problem [8] 3-partition. Our transformation uses ideas from [3]. However, the proof in [3] is for interval completion of a 4-colored graph; we require only three colors.

Problem *3-partition*
Instance: Integers $m \in \mathbb{N}$ and $Q \in \mathbb{N}$, a sequence $s_1, \cdots, s_{3m} \in \mathbb{N}$ such that $\sum_{i=1}^{3m} s_i = mQ$ and $\frac{1}{4}Q < s_i < \frac{1}{2}Q$, $1 \leq i \leq 3m$.
Question: Can the set $\{1, \cdots, 3m\}$ be partitioned into m disjoint sets $S_1, \cdots, S_m$ such that for $1 \leq j \leq m$, $\sum_{i \in S_j} s_i = Q$?

Note, if the answer is yes, then each set S_i is a 3-set. Given an instance I of the 3-partition problem, we describe the construction of 3-colored graph $G(I)$. Then we establish that I admits a 3-partition if and only if $G(I)$ admits a (Helly) circular-arc completion.

Suppose we are given an instance I of 3-partition $m, Q, s_1, s_2, \cdots, s_{3m} \in \mathbb{N}$. The graph $G(I)$ is constructed as follows:

C-cliques: Clique C_i consists of the vertices $c_{i,1}$, $c_{i,2}$, and $c_{i,3}$ where vertex $c_{i,j}$ is colored j. Add cliques $C_0, C_1, \cdots, C_{m-1}$ to $G(I)$.

Note that when clique C_{i+1} is referred to, and wherever appropriate, the arithmetic is done modulo m.

Tracks: Track-i consists of the path $d_{i,1}d_{i,2}\cdots d_{i,24Q}$ where vertex $d_{i,j}$ is colored 1 if j mod 3 = 1, colored 2 if j mod 3 = 2, and colored 3 if j mod 3 = 0. Thus, the color pattern 123123... repeats on the path. Construct track-0, track-1, $\cdots$, track-$m-1$. Identify vertex $c_{i,1}$ of C_i with vertex $d_{i,1}$ of track-i, for $0 \leq i \leq m-1$. Also, identify vertex $d_{i,24Q}$ of track-i with vertex $c_{(i+1),3}$ of C_{i+1}, for $0 \leq i \leq m-1$.

Thus, the part of $G(I)$ formed so far consists of cyclically ordered cliques $C_0, C_1, \cdots, C_{m-1}$ where C_i is connected to C_{i+1} by track-i.

W-paths: Corresponding to s_i, construct the path $W^i = e_{i,1}e_{i,2}\cdots e_{i,24s_i-2}$. Vertex $e_{i,j}$ is colored 2 if j mod 3 = 1, colored 3 if j mod 3 = 2, and colored 1 if j mod 3 = 0. Thus, the color pattern 231231... repeats on the path with each of the vertices $e_{i,1}$ and $e_{i,24s_i-2}$ colored 2. Add the paths $W^1, W^2, \cdots, W^{3m}$ to $G(I)$.

In summary, the 3-colored graph $G(I)$ consists of the m C-cliques cyclically connected by the m tracks in disjoint union with the $3m$ W-paths. We refer to the part of $G(I)$ consisting of the C-cliques connected by the tracks as the *circle-gadget*. The reader is referred to Figure 2 for a schematic.

In the transformation in [3] from 3-partition to the interval completion of 4-colored graphs the weight-gadgets are paths. The graph G in the constructed instance includes two cliques of size 4, and a vertex that is adjacent to exactly one vertex in each of these cliques and to one endpoint of each weight path. This forces the weight paths to lie between the cliques of size 4 in any interval model of an interval completion of G. In the case of completion to a circular-arc graph, this type of structure in the constructed graph would over-constrain the completion; hence we are able to construct a 3-colored graph.

Let *3-colored-kite* be the 3-colored graph given in Figure 1, where each vertex with subscript i has color i.

Proposition 1. *Let A be a model of any circular-arc completion of a 3-colored-kite. Then, the arcs corresponding to the vertices of the clique $\{x_1, c_2, c_3\}$ must have the Helly property in A.*

Proposition 2. *Let A be a model of any circular-arc completion of the circle-gadget. Then, A is a Helly circular-arc model that covers the circle.*

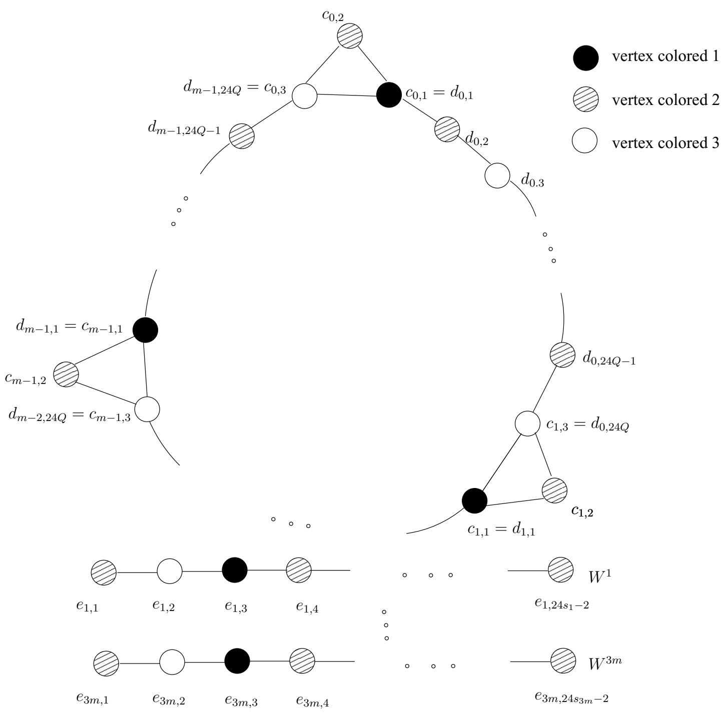


Fig. 2. A schematic of the transformation: the graph $G(I)$

Corollary 2. *Let A be a model of any circular-arc completion of $G(I)$. Then, A is a Helly circular-arc model.*

In the next two lemmata, we employ ideas from [3] where it is shown that the interval completion problem for 4-colored graphs is NP-complete.

Lemma 2. *Suppose I admits a 3-partition. Then, $G(I)$ admits a circular-arc completion. Further, any model for a circular-arc completion of $G(I)$ is a Helly circular-arc model.*

Lemma 3. *Suppose $G(I)$ admits a circular-arc completion. Then, any model for a circular-arc completion of $G(I)$ is a Helly circular-arc model. Further, I admits a 3-partition.*

Proof. Let H be a circular-arc completion of $G(I)$ and let A be a circular-arc model for H; it follows from Corollary 2 that A is a Helly circular-arc model. Consider a circular ordering R of cliques of H such that for any vertex x of H, the cliques containing x are consecutive in R. As the paths $\{W^j\}$ are disjoint from the circle-gadget in $G(I)$ and the vertices of a C_i are colored 1, 2, and 3, each endpoint of each edge of W^j is part of a clique that is between C_i and C_{i+1} in the clockwise direction, for some i, in R. As the arcs corresponding to the vertices of the circle-gadget cover the circle (ref. Proposition 2), every clique in R must contain a $c_{i,s}$ or a $d_{i,r}$ vertex that is colored 1, 2, or 3. It then follows that the endpoints of an edge $e_{j,k}e_{j,k+1}$ of W^j whose vertices are colored 3 and 1 must be part of a clique of R that contains a track vertex $d_{i,r}$ colored 2. Corresponding to each such vertex $d_{i,r}$ of track-i that is colored 2, there exists a segment S of R where each clique in S contains vertex $d_{i,r}$. Clearly, there are $8Q$ such segments in track-i, all the segments are between C_i and C_{i+1}, and the segments are pairwise disjoint.

Now, we claim that there is at most one clique in S corresponding to vertex $d_{i,r}$ that contains the endpoints of an edge $e_{j,k}e_{j,k+1}$ of W^j, for some j, where $e_{j,k}$ is colored 3 and $e_{j,k+1}$ is colored 1. Let K_a be the last clique in the counter-clockwise direction that contains the vertex $d_{i,r}$ and let K_b be the last clique in the clockwise direction that contains the vertex $d_{i,r}$. By way of contradiction, suppose there is another edge $e_{p,q}e_{p,q+1}$ of W^p with $e_{p,q}$ colored 3 and $e_{p,q+1}$ colored 1 such that clique K_c contains $d_{i,r}$, $e_{j,k}$, and $e_{j,k+1}$ and clique K_d contains $d_{i,r}$, $e_{p,q}$, and $e_{p,q+1}$. Suppose w.o.l.g. that the ordering of the cliques in the clockwise direction from K_a is: K_a, K_c, K_d, K_b. Note that it may be the case that $K_a = K_c$ or $K_b = K_d$. We will show that this implies that either the arc for $e_{j,k}$ is contained in the arc for $d_{i,r}$ or the arc for $e_{j,k+1}$ is contained in the arc for $d_{i,r}$. This yields a contradiction as each of $e_{j,k}$ and $e_{j,k+1}$ is adjacent to some vertex on W^j colored 2. Let K_{ap} be the first clique distinct from K_a encountered when moving in the counter-clockwise direction from K_a. Observe that K_{ap} must contain a vertex of color 1 or a vertex of color 3 from the circle-gadget (ref. Proposition 2). If K_{ap} contained a vertex of color 3, then the arc for $e_{j,k}$ is constrained to be contained in the arc for $d_{i,r}$; otherwise, the arc for $e_{j,k+1}$ is constrained to be contained in the arc for $d_{i,r}$.

Let B_i be the indices of paths from $\{W^j\}$ each of whose edges is part of a clique between C_i and C_{i+1} in the clockwise direction. Let $B_i = \{i_1, \cdots, i_r\}$; we may assume $r \geq 1$. As each W^j contains $8s_j - 1$ edges whose endpoints are colored 1 and 3, the total number of such edges corresponding to the W^j paths whose indices are in B_i is $(8s_{i_1} - 1) + \cdots + (8s_{i_r} - 1)$. It follows that $8(s_{i_1} + \cdots + s_{i_r}) - r \leq 8Q$. For each s_i, as $s_i \in \mathbb{N}$ and $s_i > \frac{Q}{4}$, we have $s_i \geq \frac{Q}{4} + \frac{1}{4}$. Thus, we have $8(\frac{Q}{4} + \frac{1}{4})r - r \leq 8Q$ and hence, $r \leq 3$. It follows from integrality that $(s_{i_1} + \cdots + s_{i_r}) \leq Q$. Thus, there is a partition of $\{1, \cdots, 3m\}$ into sets S_1, $\cdots$, S_m such that the sum of members of $\{s_j \mid j \in S_i\}$, for $1 \leq i \leq m$, is at most Q. However, as the sum of s_1 through s_{3m} is mQ, members of $\{s_j \mid j \in S_i\}$, for $1 \leq i \leq m$, sum to exactly Q and I admits a 3-partition. □

Theorem 4. *Given a 3-colored graph G, each of the following is NP-complete: (i) Does G admit a circular-arc completion? (ii) Does G admit a Helly circular-arc completion?*

Proof. Since circular-arc (Helly circular-arc) graphs can be recognized in polynomial time, the problems are in NP. As 3-partition is NP-complete in the strong sense, the transformation provided above, whose correctness follows from Lemma 2 and Lemma 3, is a polynomial-time transformation. □

Remark 1. A modification of the transformation from 3-partition given here can be used to prove that given a 3-colored graph G, deciding whether G admits a unit circular-arc completion is NP-complete and deciding whether G admits a proper circular-arc completion is NP-complete.

Corollary 3. *The Helly circular-arc sandwich problem is NP-complete.*

Let G be a k-colored graph. Construct graph H from G by adding a new vertex colored $k + 1$ that is adjacent to every vertex of G. It is seen that G admits a (Helly) circular-arc completion if and only if H does. Thus, we have the following:

Corollary 4. *For every fixed $k \geq 3$, deciding whether a k-colored graph admits a circular-arc completion or Helly circular-arc completion is NP-complete.*

4 Strongly Chordal Completions

We consider the problem of strongly chordal completion of a colored graph, when the input graph is k-colored. The problem is NP-complete when k is not a constant [7]. When k is fixed and $k = 2$, the problem is trivial. Next, we present an $O(n^2)$-time algorithm the problem when $k = 3$. The chordal completion of a 3-colored graph is well studied and multiple linear-time algorithms are known for it [5,11,12].

Observe that if a 3-colored chordal graph contains a sun, then it must be a 3-sun. Thus, given a 3-colored graph G, the problem of determining whether G admits a strongly chordal completion is equivalent to the problem of determining whether G admits a chordal completion that does not contain a 3-sun.

Proposition 3. [5,14] *Let G be a $k+1$-colored graph. G admits a chordal completion if and only if G admits a chordal completion that is a k-tree.*

We note that the corresponding statement for strongly chordal completion is not true. Construct G as follows: Start with the P_5 $abcde$ colored with colors 1 and 2 and add vertex f colored 3 so that f is adjacent to each of a, b, c, d, e. Finally, add vertex g colored 3 adjacent only to c. Every way G can be completed to be a 2-tree contains a 3-sun, and hence, G does not admit any strongly chordal completion that is a 2-tree. However, G clearly admits a strongly chordal completion, namely itself. However, when the input graph is bi-connected, the following shows that it is enough to consider completions to 2-trees.

Proposition 4. *If G is a k-connected $k + 1$-colored chordal graph with at least $k + 1$ vertices, then G is a k-tree.*

Let G be a 3-colored graph given as input. It is clear that G can be completed to be strongly chordal if and only if each bi-connected component of G can be completed to be strongly chordal. So, we assume that the input graph is bi-connected. Further, given Proposition 4, the problem reduces to testing whether G admits a 2-tree completion that avoids a 3-sun. Therefore, it is necessary that G is a partial 2-tree.

A *tree of cycles* [5] is a graph defined recursively as follows: A chordless cycle is a tree of cycles. A triangle is considered to be a chordless cycle. Given a tree of cycles T_k with k chordless cycles, a tree of cycles T_{k+1} with $k+1$ chordless cycles can be constructed by adding a new chordless cycle and identifying the end vertices of an edge of the new cycle with the end vertices of an edge in T_k. In a tree of cycles an *attach edge* is an edge shared by two or more chordless cycles.

Given a bi-connected partial 2-tree G, a *cell completion* G' of G [5] refers to adding certain edges to G that must be present in every completion of G to a 2-tree. A cell completion of a 3-colored graph G that is properly colored, if it exists, is called a *3-colored cell completion* of G.

Theorem 5. [5] *Bi-connected graph G is a partial 2-tree if and only if its cell completion G' is a tree of cycles. Further, the cell completion of G can be computed in linear time.*

Observation 2. *Let G be a bi-connected 3-colored graph. G admits a strongly chordal completion if and only if G has a 3-colored cell completion that admits a strongly chordal completion.*

In [5] a characterization was provided of the case when a cell completion (tree of cycles) of a bi-connected 3-colored graph admits a chordal completion: it was shown that the cell completion must be properly colored and every chordless cycle in the cell completion must use all three colors. We provide a similar characterization for the case of completion to strongly chordal graphs. It turns out that the structure of the tree of cycles becomes restricted as well as the completions of the individual chordless cycles.

Observation 3. *A 3-coloring of a 3-sun is unique (up to naming the colors) and it is impossible to destroy a 3-colored 3-sun by adding edges to it while maintaining chordality and a proper coloring.*

Lemma 4. *Let G' be a 3-colored cell completion of a bi-connected 3-colored graph G. Then, G' can be completed to be strongly chordal if and only if each chordless cycle C of G' can be completed to be strongly chordal without creating a 3-sun each of whose edges is either an attach edge or has both its endpoints on the same chordless cycle.*

Therefore, it is sufficient to complete each chordless cycle to a strongly chordal graph ensuring that when the completions of chordless cycles are glued together at the attach edges, no 3-sun will be created.

Observation 4. *Let C be 3-colored chordless cycle. Then, any chordal completion C' of C is a triangulation, and hence, is a 2-tree. Further, every edge of C is in a unique triangle in C'.*

Let C' be a triangulation of a 3-colored chordless cycle C. Triangle A in C' is a *border* triangle if it includes two edges of C; A is an *interior* triangle if it includes exactly one edge of C. Let B be a cycle in graph G. Edge e of B is in an *exterior* triangle of B if e is part of a triangle that has a vertex not on B.

In Lemma 5 and Corollaries 5 and 6, we assume a positive instance of the strongly chordal completion of a 3-colored graph problem; let G' be a 3-colored cell completion of a bi-connected 3-colored graph G and let H be a strongly chordal completion of G'.

Lemma 5. *Suppose C' is a triangulation of some cycle C in H.*

(i) If an edge e of C is in an exterior triangle of C, then it is in a border triangle of C'.
(ii) If each of the edges uv, vw on C is in an exterior triangle of C, then v is adjacent in C' to all the other vertices of C. Further, there cannot be another edge on C in an exterior triangle of C.
(iii) If C has edge e that is in an exterior triangle of C, then C' admits a linear ordering of the triangles in C' such that the ordering starts with a border triangle containing e, ends with a border triangle, each of the other triangles is an interior triangle, and for each vertex x of C, the triangles that contain x are consecutive in the ordering. Further, if C has edge $f \neq e$ that is also in an exterior triangle of C, then f must be in the terminal border triangle in the ordering.

Corollary 5. *Suppose C' is a triangulation of some cycle C in H. Then, C has at most two edges each of which is in an exterior triangle of C. Specifically, a chordless cycle in G' has at most two attach edges.*

Let G' be a 3-colored cell completion of a bi-connected 3-colored graph G. A chordless cycle C in G' is of *type-i*, $i = 0, 1, 2$, if C has i attach edges.

Corollary 6. *For a chordless cycle C in G', let C' be the triangulation of C in H. Then, C' is an interval graph that admits a linear ordering of its triangles such that the first and last are border triangles of C', the rest are interior triangles of C', and for every vertex x in C, the triangles containing x are consecutive in the ordering. Further,*

(i) if C is of type-2 with attach edges e and f, then e is in the first triangle and f is in the last triangle.
(ii) if C is of type-1 with the attach edge e, then e is in the first triangle and some edge in $E(C)$ is in the last triangle.
(iii) if C is of type-0 with no attach edges, then some edge in $E(C)$ is in the first triangle and some edge in $E(C)$ is in the last triangle.

Let C be a chordless cycle of type-i, $i = 0, 1, 2$, in a 3-colored cell completion G' of a bi-connected graph G. A triangulation C' of C is a *compatible interval completion* if C' satisfies the part of Corollary 6 that applies to C.

Theorem 6. *Let G' be a 3-colored cell completion of a bi-connected 3-colored graph G and H be an edge-supergraph of G' that is properly colored. Then, H is strongly chordal if and only if, for each chordless cycle C in G', C has at most two attach edges and the completion C' of C in H is a compatible interval completion.*

Algorithm 1. stc-completion

```
input: Bi-connected 3-colored graph $G$
output: yes if $G$ can be completed to be strongly chordal and no otherwise
if $G$ is not a partial 2-tree then
  output no; stop
else
  Compute any 2-tree completion $H$ of $G$
end if
Use $G$ and $H$ to compute the cell completion $G'$ of $G$
if $G'$ is not properly colored then
  output no; stop
else
  Construct list $L$ of chordless cycles in $G'$
end if
if some chordless cycle in $L$ has more than two attach edges then
  output no; stop
end if
for each chordless cycle $C$ in $L$ do
  if $C$ does not admit a compatible interval completion then
    output no; stop
  end if
end for
output yes
```

Theorem 7. *Algorithm stc-completion is correct and it can be implemented to run in $O(n^2)$ time. Thus, deciding whether a 3-colored graph can be completed to be strongly chordal can be done in $O(n^2)$ time.*

Proof. Since the cell completion adds edges that must be in every 2-tree completion of a bi-connected partial 2-tree, if the cell completion is not properly colored then the answer is *no*. The rest of the correctness of the algorithm follows from the lemmata, corollaries, and the discussions preceding it. Testing whether G is a partial 2-tree, and if so completing it to a 2-tree H, can be done in linear time [13]. It is shown in [5] that from G and H, the cell completion G' of G can be computed in linear time and a list of the chordless cycles in G' can be found in linear time. An algorithm to test whether a 3-colored chordless cycle admits a compatible interval completion in $O(n^2)$ time appears in [3]. □

We note that from details in the paper, when a 3-colored graph has a strongly chordal completion H, the graph H can be constructed in $O(n^2)$ time.

5 Conclusions

We have provided a full classification of the complexities of the problems of completing a colored graph to be a circular-arc graph, and completing a colored graph to be a Helly circular-arc graph, when the number of colors used is fixed. We have also shown that deciding whether a 3-colored graph admits a strongly chordal completion can be decided in $O(n^2)$ time; we conjecture that this problem for 4-colored graphs is NP-complete. It follows from our results that the sandwich problem for Helly circular-arc graphs is NP-complete. In the proof that deciding whether a 3-colored graph has a circular-arc completion is NP-complete, the graph that we construct is disconnected. We remark that the problem is unlikely to be easy for connected graphs.

References

1. Alvarez, C., Diaz, J., Serna, M.: The hardness of intervalizing four colored caterpillars. Discrete Math. 235, 245–253 (2001)
2. Alvarez, C., Serna, M.: The proper interval colored graph problem for caterpillar trees. Electron. Notes in Discrete Math. 17, 23–28 (2004)
3. Bodlaender, H.L., de Fluiter, B.: On intervalizing k-colored graphs for DNA physical mapping. Discrete Appl. Math. 71, 55–77 (1996)
4. Bodlaender, H.L., Fellows, M.R., Warnow, T.J.: Two strikes against perfect phylogeny. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 273–283. Springer, Heidelberg (1992)
5. Bodlaender, H.L., Kloks, T.: A simple linear time algorithm for triangulating three-colored graphs. J. Algorithms 15, 160–172 (1993)
6. Fellows, M.R., Hallett, M.T., Warcham, H.T.: DNA physical mapping: three ways difficult. In: Lengauer, T. (ed.) ESA 1993. LNCS, vol. 726, pp. 157–168. Springer, Heidelberg (1993)
7. de Figueiredo, C.M.H., Faria, L., Klein, S., Sritharan, R.: On the complexity of the sandwich problems for strongly chordal graphs and chordal bipartite graphs. Theoret. Comput. Sci. 381, 57–67 (2007)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to NP-Completness. Freeman, New York (1979)
9. Goldberg, P.W., Golumbic, M.C., Kaplan, H., Shamir, R.: Four strikes against physical mapping. J. Comput. Biol. 2, 139–152 (1995)
10. Golumbic, M.C., Kaplan, H., Shamir, R.: Graph sandwich problems. J. Algorithms 19, 449–473 (1995)
11. Idury, R., Schaffer, A.: Triangulating three-colored graphs in linear time and linear space. SIAM J. Discrete Math. 2, 289–293 (1993)
12. Kannan, S.K., Warnow, T.J.: Triangulating 3-colored graphs. SIAM J. Discrete Math. 5, 249–258 (1992)
13. Matousek, J., Thomas, R.: Algorithms finding tree-decompositions of graphs. J. Algorithms 12, 1–22 (1991)
14. McMorris, F.R., Warnow, T.J., Wimer, T.: Triangulaing vertex-colored graphs. SIAM J. Discrete Math. 7, 296–306 (1994)
15. Sritharan, R.: Chordal bipartite completion of colored graphs. Discrete Math. 308, 2581–2588 (2008)
16. Tucker, A.: An efficent test for circular arc graphs. SIAM J. Comput. 9, 1–24 (1980)
17. Warnow, T.J.: Combinatorial algorithms for constructing phylogenetic trees, Ph. D. Thesis, University of California, Berkeley (1991)

Colouring of Graphs with Ramsey-Type Forbidden Subgraphs*

Konrad K. Dabrowski[1], Petr A. Golovach[2], and Daniël Paulusma[3]

[1] ESSEC Business School, Av. B. Hirsch, 95021 Cergy Pontoise, France
dabrowski@essec.edu

[2] Department of Informatics, Bergen University,
PB 7803, 5020 Bergen, Norway
petr.golovach@ii.uib.no

[3] School of Engineering and Computing Sciences, Durham University,
Science Laboratories, South Road, Durham DH1 3LE, United Kingdom
daniel.paulusma@durham.ac.uk

Abstract. A colouring of a graph $G = (V, E)$ is a mapping $c : V \to \{1, 2, \ldots\}$ such that $c(u) \neq c(v)$ if $uv \in E$; if $|c(V)| \leq k$ then c is a k-colouring. The Colouring problem is that of testing whether a given graph has a k-colouring for some given integer k. If a graph contains no induced subgraph isomorphic to any graph in some family $\mathcal{H}$, then it is called $\mathcal{H}$-free. The complexity of Colouring for $\mathcal{H}$-free graphs with $|\mathcal{H}| = 1$ has been completely classified. When $|\mathcal{H}| = 2$, the classification is still wide open, although many partial results are known. We continue this line of research and forbid induced subgraphs $\{H_1, H_2\}$, where we allow H_1 to have a single edge and H_2 to have a single non-edge. Instead of showing only polynomial-time solvability, we prove that Colouring on such graphs is fixed-parameter tractable when parameterized by $|H_1| + |H_2|$. As a byproduct, we obtain the same result both for the problem of determining a maximum independent set and for the problem of determining a maximum clique.

1 Introduction

Graph colouring involves the labelling of the vertices of some given graph by integers called colours such that no two adjacent vertices receive the same colour. The Colouring problem is that of deciding whether or not a graph can be coloured with at most k colours for some given integer k. Because Colouring is NP-complete for any fixed $k \geq 3$, its computational complexity has been widely studied for special graph classes, see for example the surveys of Randerath and Schiermeyer [23] and Tuza [26]. In this paper, we consider the Colouring problem for graphs characterized by two forbidden induced subgraphs. Before we summarize related results and explain our new results, we first state the necessary terminology.

* Research Supported by EPSRC (EP/G043434/1), ERC (267959) and ANR (TODO ANR-09-EMER-010).

1.1 Basic Terminology

We only consider finite undirected graphs without loops or multiple edges. We refer to the textbook of Diestel [10] for any undefined graph terminology. Let $G = (V, E)$ be a graph. A *colouring* of G is a mapping $c : V \rightarrow \{1, 2, \ldots\}$ such that $c(u) \neq c(v)$ whenever $uv \in E$. We call $c(u)$ the *colour* of u and $\{u \in V \mid c(u) = i\}$ for some $i \geq 1$ a *colour class* of c. A *k-colouring* of G is a colouring c of G with $1 \leq c(u) \leq k$ for all $u \in V$. The smallest integer k for which G has a k-colouring is called the *chromatic number* of G, denoted $\chi(G)$; a $\chi(G)$-colouring is said to be *optimal*. The k-Colouring problem is that of deciding whether a given graph admits a k-colouring. Here, k is *fixed*, that is, not part of the input. If k is part of the input, then we denote the problem as Colouring.

Let $G = (V, E)$ be a graph. A graph H is an *induced subgraph* of G if H can be obtained from G by deleting zero or more vertices. In this case we write $H \subseteq_i G$. For a set $S \subseteq V$, we let $G[S] = (S, \{uv \in E \mid u, v \in S\})$ denote the subgraph of G *induced by* S. Let $\{H_1, \ldots, H_p\}$ be a set of graphs. We say that G is *$(H_1, \ldots, H_p)$-free* if G has no induced subgraph isomorphic to a graph in $\{H_1, \ldots, H_p\}$; if $p = 1$, we may write that G is H_1-free instead of (H_1)-free.

The *complement* of a graph $G = (V, E)$, denoted by $\overline{G}$, has vertex set V and an edge between two distinct vertices if and only if these vertices are not adjacent in G. The *disjoint union* of two graphs G and H with $V(G) \cap V(H) = \emptyset$ is the graph with vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$, and we denote this by $G + H$. We denote the disjoint union of r copies of G by rG.

For $r \geq 1$, the graph P_r denotes the *path* on r vertices, that is, $V(P_r) = \{u_1, \ldots, u_r\}$ and $E(P_r) = \{u_i u_{i+1} \mid 1 \leq i \leq r - 1\}$. Adding the edge $u_1 u_r$ to this graph yields the *cycle* on r vertices, denoted C_r. The graph sP_1 denotes the *independent set* on s vertices. The Independent Set problem is that of testing whether a given graph has an independent set of size at least k for some given integer k. The graph K_t denotes the *complete* graph on t vertices, that is, $V(K_t) = \{u_1, \ldots, u_t\}$ and $E(K_t) = \{u_i u_j \mid 1 \leq i < j \leq t\}$. The vertex set of a complete graph is called a *clique*. The Clique problem is that of testing whether a given graph has a clique of size at least k for some integer k. The graph $K_t - e$ denotes the graph obtained from K_t after removing exactly one edge.

1.2 Our Results

We show fixed-parameter tractability results for three problems, namely for Colouring, Independent Set and Clique, restricted to $(sP_1 + P_2, K_t - e)$-free graphs. In parameterized complexity theory, the problem input consists of a pair (I, p), where I is the problem instance and p is the parameter. A problem is *fixed-parameter tractable* (fpt) with parameter p if it can be solved in time $f(p) \cdot |I|^{O(1)}$ for some function f that only depends on p. In our case, a natural parameter is $s + t$. In Section 2 we show that Colouring is fixed-parameter tractable with parameter $s + t$ when restricted to $(sP_1 + P_2, K_t - e)$-free graphs, that is, can be solved in time $f(s + t)(n + k)^{O(1)}$ for some function f that only depends on $s + t$. In the same section, we show that the Independent Set and

Clique problems are also fixed-parameter tractable with parameter $s+t$ when restricted to (sP_1+P_2, K_t-e)-free graphs. However, the main motivation for our research comes from the area of graph colouring, as we explain in Section 1.3. In Section 3 we give some directions for future research. There, we also show that Colouring is polynomial-time solvable for $(2P_2, K_t - e)$-free graphs.

Finally, it should be noted that many classes of (H_1, H_2)-free graphs are known to have bounded clique-width (see Section 1.3), in which case it follows that Colouring is polynomial-time solvable [19]. The graph classes we consider only have bounded clique-width for small values of s and t; the proof of this claim has been omitted due to space restrictions. Thus our results (which are stronger than mere polynomial-time solvability) do not fall into this category.

1.3 Motivation and Related Work

The complexity of Maximum Independent Set restricted to H-free graphs has only been partially classified. For instance, the complexity status of this problem on P_5-free graphs is a notorious open case (see [24] for a subexponential algorithm). Because a graph has an independent set of size at least k if and only if its complement has a clique of size at least k, the complexity classification of the Clique problem on H-free graphs is also far from being settled. Our research on the Colouring problem, which is the main focus of our paper, is well embedded in the literature. As a starting point, Král' et al. [18] completely determined the computational complexity of Colouring for H-free graphs.

Theorem 1 ([18]). *Let H be a fixed graph. If H is a (not necessarily proper) induced subgraph of P_4 or of $P_1 + P_3$, then* Colouring *can be solved in polynomial time for H-free graphs; otherwise it is* NP*-complete for H-free graphs.*

Theorem 1 initiated a study of the computational complexity of the k-Colouring problem on H-free graphs. This classification is still open (see [12] for a survey). Due to Theorem 1, the Colouring problem restricted to graph classes characterized by *two* forbidden induced subgraphs has received a significant amount of attention as well. We survey known results below.

We observe that Theorem 1 implies that Colouring is polynomial-time solvable for (H_1, H_2)-free graphs if one of H_1, H_2 is an induced subgraph of P_4 or of $P_1 + P_3$. Another straightforward case is as follows. For positive integers s and t, the *Ramsey number* $R(s,t)$ is the smallest number n such that all graphs on n vertices contain an independent set of size s or a clique of size t. Ramsey's Theorem (see e.g. [10]) states that such a number exists for all positive integers s and t. As an immediate consequence, Colouring is polynomial-time solvable on (sP_1, K_t)-free graphs for all s and t. Hence, it is natural to consider graph classes that can be obtained by adding one edge to the first forbidden induced subgraph and removing one edge from the second. This leads to the class of $(sP_1 + P_2, K_t - e)$-free graphs; note that this class includes all $(sP_1, K_t - e)$-free graphs and all $(sP_1 + P_2, K_{t-1})$-free graphs. As explained in Section 1.2, we have fpt algorithms with parameter $s+t$ for solving Colouring on (sP_1+P_2, K_t-e)-

free graphs (and consequently for $(sP_1, K_t - e)$-free graphs and for $(sP_1 + P_2, K_t)$-free graphs). This result adds to the body of existing work on Colouring for (H_1, H_2)-free graphs, which we further discuss below.

The following result, which we will use later on, is due to Gyárfás [15].

Theorem 2 ([15]). *Let $\ell, t \geq 1$ be two integers. Then every (P_ℓ, K_t)-free graph can be coloured with at most $(\ell - 1)^{t-2}$ colours.*

Theorem 2 was slightly improved by Gravier, Hoáng and Maffray [14], who showed that every (P_ℓ, K_t)-free graph that is not a complete graph can be coloured with at most $(\ell - 2)^{t-2}$ colours. Each of these two results implies that Colouring is polynomial-time solvable on (F, K_t)-free graphs, whenever F is the disjoint union of one or more paths such that k-Colouring is polynomial-time solvable on F-free graphs for all $k \geq 1$. Combining this observation with such existing results for k-Colouring [6,9,16] gives us a number of polynomial-time solvable cases [11]. Also, the fact that Colouring can be solved in polynomial time on graphs of bounded clique-width [19] directly leads to polynomial-time results for Colouring restricted to $(K_{1,3}, C_3 + P_1)$-free graphs [1], $(P_1 + P_4, \overline{P_1 + P_4})$-free graphs [3], $(P_5, \overline{P_1 + P_4})$-free graphs [2] and $(P_1 + P_4, \overline{P_5})$-free graphs [2]. Here, the graph $K_{1,r}$ denotes the graph with vertices $u, v_1, \ldots, v_r$ and edges $uv_1, \ldots, uv_r$. Other results on Colouring for (H_1, H_2)-free graphs can be found in [4,5,9,17,18,20,21,22,25], all of which are summarized in Theorem 3 together with the above results and a weaker formulation of our new result (Statement (ii)-8). For details we refer to Golovach and Paulusma [11], who formulate the same theorem without Statement (ii)-8. In this theorem, C_3^+ denotes the *paw*, which is the graph with vertices a, b, c, d and edges ab, ac, ad, bc.

Theorem 3. *Let H_1 and H_2 be two fixed graphs. Then the following holds:*

(i) Colouring *is* NP*-complete for (H_1, H_2)-free graphs if*

1. *$H_1 \supseteq_i C_r$ for some $r \geq 3$ and $H_2 \supseteq_i C_s$ for some $s \geq 3$*
2. *$H_1 \supseteq_i K_{1,3}$ and $H_2 \supseteq_i K_{1,3}$*
3. *H_1 and H_2 contain a spanning subgraph of $2P_2$ as an induced subgraph*
4. *$H_1 \supseteq_i C_3$ and $H_2 \supseteq_i K_{1,r}$ for some $r \geq 5$*
5. *$H_1 \supseteq_i C_r$ for $r \geq 4$ and $H_2 \supseteq_i K_{1,3}$*
6. *$H_1 \supseteq_i C_3$ and $H_2 \supseteq_i P_{164}$*
7. *$H_1 \supseteq_i C_r$ for $r \geq 5$ and H_2 contains a spanning subgraph of $2P_2$ as an induced subgraph*
8. *$H_1 \supseteq_i C_r + P_1$ for $3 \leq r \leq 4$ or $H_1 \supseteq_i \overline{C_r}$ for $r \geq 6$, and H_2 contains a spanning subgraph of $2P_2$ as an induced subgraph*
9. *$H_1 \supseteq_i K_4$ or $H_1 \supseteq_i K_4 - e$, and $H_2 \supseteq_i K_{1,3}$.*

(ii) Colouring *is polynomial-time solvable for (H_1, H_2)-free graphs if*

1. *H_1 or H_2 is an induced subgraph of $P_1 + P_3$ or of P_4*
2. *$H_1 \subseteq_i K_{1,3}$ and $H_2 \subseteq_i C_3 + P_1$*
3. *$H_1 \neq K_{1,5}$ is a forest on at most six vertices and $H_2 \subseteq_i C_3^+$*
4. *$H_1 \subseteq_i sP_2$ or $H_1 \subseteq_i sP_1 + P_5$ for $s \geq 1$, and $H_2 \subseteq_i C_3^+$*

5. *$H_1 \subseteq_i sP_2$ or $H_1 \subseteq_i sP_1 + P_5$ for $s \geq 1$, and $H_2 = \overline{K_t}$ for $t \geq 4$*
6. *$H_1 \subseteq_i P_1 + P_4$ or $H_1 \subseteq_i P_5$, and $H_2 \subseteq_i \overline{P_1 + P_4}$*
7. *$H_1 \subseteq_i P_1 + P_4$ or $H_1 \subseteq_i 2P_2$, and $H_2 \subseteq_i \overline{P_5}$*
8. *$H_1 \subseteq_i sP_1 + P_2$ for $s \geq 0$ or $H_1 = 2P_2$, and $H_2 \subseteq_i K_t - e$ for $t \geq 2$.*

The following result is the only parameterized result known for Colouring of H-free graphs. Recall that k is the number of colours permitted.

Theorem 4 ([7]). *The* Colouring *problem on $(sP_1 + P_2)$-free graphs is fixed-parameter tractable with parameter $k + s$.*

Very few parameterized results on Independent Set are known [8]. In particular, we need the following one. Recall that k is the minimum number of independent vertices required.

Theorem 5 ([8]). *The* Independent Set *problem on $(K_t - e)$-free graphs is fixed-parameter tractable with parameter $k + t$.*

We remark that the running times of the algorithms of Theorems 4 and 5 are $f(k + s)n^{O(1)}$ and $g(k + t)n^{O(1)}$, respectively, with k in the exponent of both f and g, whereas in our setting k is part of the input.

2 The Proofs of Our Results

In this section, we show that the Colouring, Independent Set and Clique problems on $(sP_1 + P_2, K_t - e)$-free graphs are fixed-parameter tractable with parameter $s + t$. We need the following additional terminology.

Let $G = (V, E)$ be a graph. Then $N(u) = \{v \in V \mid uv \in E\}$ is the *neighbourhood* of $u \in V$. For $S \subseteq V$, we write $N(S) = \{v \in V \backslash S \mid uv \in E$ for some $u \in S\}$. A subset $M \subseteq E$ is a *matching* if no two edges in M share an end-vertex. A matching M is *A-saturating* for some subset $A \subseteq V$ if every vertex of A is an end-vertex of some edge in M; if M is V-saturating, then M is a *perfect* matching. A graph is *p-partite* if its vertex set can be partitioned into at most p independent sets, which we call *partition classes.* If $p = 2$, the graph is *bipartite.* The complement of a p-partite graph is called a *co-p-partite graph* (whose partition classes are cliques).

Before stating the proofs of our results, let us first give an outline. Because a graph is $(sP_1 + P_2, K_t - e)$-free if and only if its complement is $((t-2)P_1 + P_2, K_{s+2} - e)$-free, the results for the Clique problem follow immediately from those for the Independent Set problem. In Lemma 1 we show that every $(sP_1 + P_2, K_t - e)$-free graph is $((s+1)P_1, K_t - e)$-free or $(sP_1 + P_2, K_{s^2(t-3)+2})$-free. This enables us to do as follows. We first show our results for Colouring and Independent Set on $(sP_1 + P_2, K_t)$-free graphs in Lemmas 2 and 3, respectively, and on $(sP_1, K_t - e)$-free graphs in Lemmas 8 and 4, respectively. To prove these lemmas, we will use Theorems 2, 4 and 5. We also use Lemma 2 to prove Lemma 3 and Lemma 4, along with some structural results (Lemmas 5–7) to prove Lemma 8. We then combine our intermediate steps to prove Theorem 6, in which we state our main results.

As noted, we start with Lemma 1.

Lemma 1. *Let G be a (sP_1+P_2, K_t-e)-free graph. Then G is $((s+1)P_1, K_t-e)$-free or $(sP_1+P_2, K_{s^2(t-3)+2})$-free.*

Proof. Let $G=(V,E)$ be a (sP_1+P_2, K_t-e)-free graph. Suppose that G is not $(s+1)P_1$-free. We will show that G must then be $K_{s^2(t-3)+2}$-free.

Because G is not $(s+1)P_1$-free, G contains an independent set S on at least $s+1$ vertices. We assume that S is maximal (with respect to set inclusion). Let $u_1,\ldots,u_p$ be the vertices of S for some $p \geq s+1$. Let $X_1 = N(u_1)$, and for $i=2,\ldots,p$, let X_i denote the set of vertices in $V \setminus S$ that are adjacent to u_i but not to any vertex in $\{u_1,\ldots,u_{i-1}\}$. By maximality of S, every vertex of G is either in S or in some X_i.

We claim that $X_i = \emptyset$ for $i \geq s+1$. Indeed, if $i \geq s+1$ and $x \in X_i$ then $G[\{u_1,\ldots,u_s,x,u_i\}]$ would be isomorphic to sP_1+P_2.

Now suppose that for some i, X_i contains a clique K on at least $s(t-3)+1$ vertices. If $t-2$ vertices of K were adjacent to u_j for some $j \neq i$, then these $t-2$ vertices, together with u_i and u_j, would induce a K_t-e in G. Therefore, for each $j \neq i$, at most $t-3$ vertices of K can be adjacent to u_j. Hence, there must be a vertex $x \in K$ that is not adjacent to any vertex in $\{u_1,\ldots,u_{i-1},u_{i+1},\ldots,u_{s+1}\}$. However, then $G[\{u_1,\ldots,u_{s+1},x\}]$ is isomorphic to sP_1+P_2. This is a contradiction. Thus all sets $X_1,\ldots,X_s$ can only contain cliques of size at most $s(t-3)$. Because S is an independent set and $X_i=\emptyset$ for $i \geq s+1$, this means that the largest clique in G has size at most $s^2(t-3)+1$. We conclude that G is $K_{s^2(t-3)+2}$-free, as desired. □

We are now ready to prove Lemmas 2 and 3.

Lemma 2. *The* Colouring *problem on (sP_1+P_2, K_t)-free graphs is fixed-parameter tractable with parameter $s+t$.*

Proof. We observe that every (sP_1+P_2, K_t)-free graph is (P_{2s+2}, K_t)-free, and consequently can be coloured with at most $(2s+1)^{t-2}$ colours due to Theorem 2. Hence we can apply Theorem 4. In fact, Theorem 4 gives us an explicit optimal colouring, rather than just the chromatic number. Hence, even the problem of finding an optimal colouring in a (sP_1+P_2, K_t)-free graph is fixed-parameter tractable with parameter $s+t$. □

Lemma 3. *The* Independent Set *problem on (sP_1+P_2, K_t)-free graphs is fixed-parameter tractable with parameter $s+t$.*

Proof. Let G be a (sP_1+P_2, K_t)-free graph on n vertices. By (the proof of) Lemma 2, we can find an optimal colouring c of G in fpt time with parameter $s+t$. Because G is (P_{2s+2}, K_t)-free, c uses $k \leq (2s+1)^{t-2}$ colours due to Theorem 2. Let $C_1,\ldots,C_k$ be the colour classes of c. We may assume that C_1 is a maximal independent set (with respect to set inclusion) in G, and that for $i=2,\ldots,k$, the set C_i is a maximal independent set in $G \setminus (C_1 \cup \cdots \cup C_{i-1})$.

Indeed, if some $x \in C_i$ has no neighbours in C_j for some $j < i$, we can move x to C_j in order to obtain another optimal colouring of G.

We branch by choosing an index b to be the largest index such that C_b contains a vertex of the maximum independent set I that we are searching for. There are at most $(2s+1)^{t-2}$ ways of doing this. We then branch further by choosing a vertex x in C_b that we assume will be in I. This leads to at most n branches altogether. By maximality, x has a neighbour in C_a for every $a < b$. Let J_a be the set of vertices in C_a that are not adjacent to x. Because G is $(sP_1 + P_2)$-free, J_a contains at most $s-1$ vertices. Let $H_x = J_1 \cup \cdots \cup J_{b-1}$. By the definition of the sets J_i and the choice of x, we find that $I \setminus H_x \subseteq C_b$. We branch further by choosing an independent set $I' \subseteq H_x$. Because H_x has size at most $\beta = (b-1)(s-1) \leq ((2s+1)^{t-2}-1)(s-1)$, there are at most 2^β ways of doing this. We then extend I' by adding all vertices of C_b that do not have a neighbour in I'. The final independent set is a candidate for being a maximum independent set. After considering all independent sets obtained in this way, we choose one that has maximum size. Because we considered all possible ways of constructing a maximum independent set, the above algorithm is correct. Because our algorithm constructs at most $n2^\beta$ independent sets, it runs in fpt time when parameterized by $s+t$. □

Here is Lemma 4, the proof of which we have omitted.

Lemma 4. *The* Independent Set *problem on $(sP_1, K_t{-}e)$-free graphs is fixed-parameter tractable with parameter $s+t$.*

To prove Lemma 8 we need three structural lemmas, the first of which is well-known and can be found in [10].

Lemma 5 (Hall's Marriage Theorem). *A bipartite graph G with vertex partition $A \cup B$ has an A-saturating matching if and only if $|N(X)| \geq |X|$ for all $X \subseteq A$.*

Lemma 6 follows from Lemma 5; we omit the proof details. Note that in a bipartite graph with partition classes A and B an A-saturating matching is perfect if $|A| = |B|$.

Lemma 6. *Let G be a bipartite graph with partition classes A and B. Let p, q, n be integers such that $|A| = |B| = n \geq p + q$. If every vertex in A has degree at least $n - p$ and every vertex in B has degree at least $n - q$, then G contains a perfect matching.*

We need Lemma 6 to prove Lemma 7. Lemma 7 is a key lemma. It gives us a sufficient condition on the number of edges that we may allow between mutually vertex-disjoint cliques without increasing the chromatic number of their union.

Lemma 7. *Let k, a, b be integers such that $k \geq 2a(b-1)$. Let G be a co-b-partite graph with partition classes $X_1, \ldots, X_b$, all of size at most k. If every vertex in X_i has at most a neighbours in X_j for all $1 \leq i, j \leq b$ when $i \neq j$, then G is k-colourable.*

Proof. Without loss of generality, we may assume that every clique X_i contains exactly k vertices. We use induction on b. The case $b = 1$ is trivial. Let $b \geq 2$. Let $G' = G[X_1 \cup \cdots \cup X_{b-1}]$. Because $k \geq 2a(b-1) \geq 2a(b-2)$, we can apply our induction hypothesis to find that G' is k-colourable. Let c be a k-colouring of G', and let $X_b = \{x_1, \ldots, x_k\}$. We construct an auxiliary bipartite graph F as follows. For each colour $1 \leq i \leq k$ we create a vertex u_i. This yields the set $U = \{u_1, \ldots, u_k\}$. For each vertex $x_j \in X_b$ we introduce a copy x'_j. This yields the set $X = \{x'_1, \ldots, x'_k\}$. The partition classes of F are U and X. We add an edge from u_i to x'_j in F if and only if c does not assign colour i to any neighbour of x_j. We observe that c can be extended to a k-colouring of G if and only if F has a perfect matching. Hence, it remains to show that this is indeed the case.

Because every X_i is a clique of size k, every colour of c occurs $b-1$ times. Recall that we assume that every vertex in X_i has at most a neighbours in X_j for all $1 \leq i, j \leq b$, where $i \neq j$. By combining these two facts we find that every u_i has degree at least $k - a(b-1)$ and that every x'_j has degree at least $k - a(b-1)$. As $k \geq 2a(b-1) = a(b-1) + a(b-1)$, we may apply Lemma 6 to find that F has a perfect matching. □

We are now ready to prove Lemma 8.

Lemma 8. *The* Colouring *problem on* $(sP_1, K_t - e)$*-free graphs is fixed-parameter tractable with parameter* $s + t$.

Proof. Let G be an $(sP_1, K_t - e)$-free graph on n vertices. We may assume without loss of generality that $s \geq 2$ and $t \geq 3$, as the proof is straightforward for $s \leq 1$ or $t \leq 2$. We first find a maximum independent set S of G. According to (the proof of) Lemma 4, we can do this in fpt time with parameter $s + t$.

We may assume without loss of generality that S is of size $s-1$; otherwise G is $((s-1)P_1, K_t - e)$-free. Let $u_1, \ldots, u_{s-1}$ be the vertices of S. For $1 \leq i < j \leq s-1$, let $X_{i,j}$ be the set of vertices adjacent to both u_i and u_j. If some $X_{i,j}$ contains a clique on $t-2$ vertices, then the vertices of this clique together with u_i, u_j form an induced $K_t - e$. Hence, every $X_{i,j}$ is (sP_1, K_{t-2})-free. Recall that $R(s,t)$ is the Ramsey number for integers s and t. Because $X_{i,j}$ is (sP_1, K_{t-2})-free, $X_{i,j}$ contains at most $R(s, t-2) - 1$ vertices. Let $D = \bigcup X_{i,j}$. Then $|D| \leq \binom{s-1}{2}(R(s, t-2) - 1)$. Hence, the size of D is bounded by a function of s and t.

Let X_i consist of u_i and those vertices that are adjacent to u_i but not to any other vertex in S. Each X_i must be a clique, since if $x, y \in X_i$ were non-adjacent, then $\{x, y\} \cup S \setminus \{u_i\}$ would be an independent set larger than S, contradicting the fact that S is a maximum independent set. Note that every vertex of G is either in D or in some X_i (see Fig. 1). Hence, we find that the vertices of G can be partitioned into $s-1$ cliques $X_1, \ldots, X_{s-1}$ and a set D. However, from these s sets, we only know that D has bounded size (in terms of s and t).

Now suppose that some vertex $x \in X_i$ had $t-2$ neighbours in X_j for some $j \neq i$. Then these $t-2$ neighbours, together with x and u_j, would induce a $K_t - e$ in G. Hence, every vertex in X_i has at most $t-3$ neighbours in every

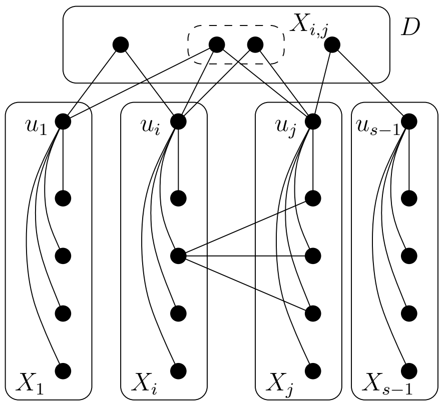


Fig. 1. Decomposition of the graph G into sets $X_1, \ldots, X_{s-1}, D$

X_j with $j \neq i$. Consequently, each vertex in X_i has at most $(t-3)(s-1)+|D|$ neighbours outside of X_i.

We now start a branching procedure by first colouring the vertices of D in every possible way using colours from the set $C = \{1, \ldots, |D|\}$. After colouring D, for each $i = 1, \ldots, s-1$, we choose a subset $C_i \subseteq C$ of size $|C_i| \leq |X_i|$, which we let consist of exactly those colours from C that will occur on X_i. We branch over all possibilities for choosing such sets C_i. After choosing the sets C_i we branch further. For all cliques X_i of size at most $\gamma = (|D|+2)(s-2)(t-3) + (t-3)|D| + |D|$ (we explain this number later) we branch by trying every possible way of colouring a subset of X_i of size $|C_i|$ with the colours from C_i. This yields a partial colouring of G. The total number of these partial colourings is at most

$$|D|^{|D|} \prod_{i=1}^{s-1} 2^{|D|} \binom{\gamma}{|C_i|} |C_i|^{|C_i|},$$

which only depends on s and t, and which may be strictly less because whenever two adjacent vertices are assigned the same colour, we naturally cut the branch.

Now let c be a partial colouring that is obtained in this way. Assume that c uses k_c colours. We let `total`(c) denote the smallest number of colours of a colouring c' of G subject to the following two conditions:

(i) c' extends c;
(ii) c' does not use any colours from $C \backslash C_i$ on vertices of X_i.

Let Z be the set of vertices that are not coloured by c. We may assume without loss of generality that $\beta = |X_1| - |C_1| = \max\{|X_i| - |C_i| \mid 1 \leq i \leq s-1\}$.

First suppose that X_1 has size at most γ, in which case Z has size at most $\sum_i |(X_i| - |C_i|) \leq (s-1)\gamma$. By definition of the sets C_i, we are required to use colours on Z that are not in C. In other words, we are to colour Z independently from the way we coloured the rest of G. Because in this case $|Z|$ only depends on s and t, we use brute force to compute the chromatic number $\chi_{G[Z]}$ of $G[Z]$. Hence we find that $\texttt{total}(c) = k_c + \chi_{G[Z]}$ in this case.

Now suppose that X_1 has size at least $\gamma + 1$. We observe the following. Let v be any vertex in D. If v is adjacent to all vertices of a clique X_i, then $c(v)$ does not appear in C_i, as otherwise we can cut the branch. If v is adjacent to a set X_i' of $t-2$ vertices of a clique X_i but not to some vertex $w \in X_i$, then $X_i' \cup \{v, w\}$ induce a $K_t - e$ in G, which is not possible. Hence, we may assume without of generality that every vertex in D is adjacent to at most $t-3$ vertices in every clique X_i. This means that the total number of vertices in X_i that have a neighbour in D is at most $(t-3)|D|$. Also recall that every vertex in X_i has at most $t-3$ neighbours in any X_j with $j \neq i$. We may assume without loss of generality that for some $h \leq s-1$, the cliques $X_1, \ldots, X_h$ are precisely those for which $|X_i| \geq \gamma + 1$. We apply the following greedy approach for assigning colours from C_i to every clique X_i with $1 \leq i \leq h$. We arbitrarily colour a set X_1^* of $|C_1|$ vertices of X_1 that are not adjacent to any vertices in D with colours from C_1. We then arbitrarily colour a set X_2^* of $|C_2|$ vertices of X_2 that are not adjacent to any vertices in $D \cup X_1^*$ with colours from C_2, and so on, until we have processed X_h. We can follow this greedy approach, because a clique X_i with $1 \leq i \leq h$ has a size that is sufficiently large, that is, at least $\gamma + 1 = (|D|+2)(s-2)(t-3) + (t-3)|D| + |D| + 1$. (Note that we could have chosen γ to be smaller here, but this value simplifies the arguments in the next paragraph.) Let c^* denote the resulting partial colouring. Note that c^* extends c.

Assume that c^* uses k_{c^*} colours. We claim that $\texttt{total}(c) = k_{c^*} + |X_1| - |C_1|$. Note that $\texttt{total}(c) < k_{c^*} + |X_1| - |C_1|$ is not possible, because of condition (ii) and the fact that X_1 is a clique. Hence, we are left to show that all uncoloured vertices of G can be coloured with at most $|X_1| - |C_1|$ colours. This follows immediately from Lemma 7 by taking $k = |X_1| - |C_1|$, $a = t-3$ and $b = s-1$. We may apply this lemma for the following two reasons. First, for $i = 1, \ldots, h$, we have $|X_i| - |C_i| \leq |X_1| - |C_1|$ by definition. Second, we have $|X_1| - |C_1| \geq \gamma + 1 - |C_1| \geq 2(s-2)(t-3)$.

As we branched in all possible ways, we find that the smallest $\texttt{total}(c)$ is the chromatic number of G. Note that our algorithm runs in fpt time with parameter $s+t$, as required, and that it also produces an optimal colouring of G. □

We are now ready to state and prove our main theorem.

Theorem 6. *The* Colouring, Independent Set *and* Clique *problems on $(sP_1 + P_2, K_t - e)$-free graphs are fixed-parameter tractable with parameter $s+t$.*

Proof. First recall the following. Because a graph is $(sP_1 + P_2, K_t - e)$-free if and only if its complement is $(P_2 + (t-2)P_1, K_{s+2} - e)$-free, we only have to consider the Independent Set and Colouring problems.

Let G be a $(sP_1 + P_2, K_t - e)$-free graph. We start by checking whether G contains an independent set on $s+1$ vertices. We can do this in fpt time with parameter $s+t$ by Theorem 5. If it does, then G is not $(s+1)P_1$-free. By Lemma 1, this means that G must be $(sP_1 + P_2, K_{s^2(t-3)+2})$-free. In this case, we can solve Colouring and Independent Set by Lemmas 2 and 3, respectively. Otherwise, that is, if G contains no independent set on $s+1$ vertices, then G is $((s+1)P_1, K_t - e)$-free. In that case, we can solve Independent Set and Colouring by Lemmas 4 and 8, respectively. $\square$

3 Final Remarks

The ultimate goal is to complete Theorem 3. This requires new proof techniques to deal with a number of nontrivial cases, such as when H_1 is the claw $K_{1,3}$ and H_2 is a long path. As regards our result it seems natural to settle, as a next step, the complexity status of Colouring for $(sP_2, K_t - e)$-free graphs. Our next result (proof omitted) shows that the case $s = 2$ is polynomial-time solvable.

Theorem 7. *The* Colouring *problem on $(2P_2, K_t - e)$-free graphs can be solved in polynomial time for all $t \geq 1$.*

Another possible generalization of our result on Colouring is to consider the following variant of graph colouring. In Precolouring Extension we assume that a (possibly empty) subset $W \subseteq V$ of a graph $G = (V, E)$ is precoloured by a *precolouring* $c_W : W \to \{1, 2, \ldots, k\}$ for some given integer k, and the question is whether we can extend c_W to a k-colouring of G. The classification of Precolouring Extension on H-free graphs is known [13]. However, the classification of Precolouring Extension on (H_1, H_2)-free graphs is still open. In this respect, we note that our results cannot be generalized to another even more general variant of graph colouring called list colouring. A *list assignment* of a graph $G = (V, E)$ is a function $\mathcal{L}$ that assigns a list $L(u)$ of so-called *admissible* colours to each $u \in V$. We say that a colouring $c : V \to \{1, 2, \ldots\}$ *respects* $\mathcal{L}$ if $c(u) \in L(u)$ for all $u \in V$. The List Colouring problem is that of testing whether a given graph has a colouring that respects some given list assignment. Golovach and Paulusma [11] completely classified the complexity of the List Colouring problem for (H_1, H_2)-free graphs by showing that this problem is polynomial-time solvable for (H_1, H_2)-free graphs in the following three cases: (i) $H_1 \subseteq_i P_3$ or $H_2 \subseteq_i P_3$, (ii) $H_1 \subseteq_i C_3$ and $H_2 \subseteq_i K_{1,3}$ and (iii) $H_1 = sP_1$ for $s \geq 3$ and $H_2 = K_t$ for $t \geq 3$, whereas it is NP-complete for all other pairs (H_1, H_2).

References

1. Brandstädt, A., Engelfriet, J., Le, H.-O., Lozin, V.V.: Clique-Width for 4-Vertex Forbidden Subgraphs. Theory Comput. Syst. 39, 561–590 (2006)
2. Brandstädt, A., Kratsch, D.: On the structure of (P_5, gem)-free graphs. Discrete Applied Mathematics 145, 155–166 (2005)
3. Brandstädt, A., Le, H.-O., Mosca, R.: Gem- and co-gem-free graphs have bounded clique-width. Internat. J. Found. Comput. Sci. 15, 163–185 (2004)
4. Broersma, H.J., Golovach, P.A., Paulusma, D., Song, J.: Determining the chromatic number of triangle-free $2P_3$-free graphs in polynomial time. Theoretical Computer Science 423, 1–10 (2012)

5. Broersma, H.J., Golovach, P.A., Paulusma, D., Song, J.: Updating the complexity status of coloring graphs without a fixed induced linear forest. Theoretical Computer Science 414, 9–19 (2012)
6. Couturier, J.F., Golovach, P.A., Kratsch, D., Paulusma, D.: List coloring in the absence of a linear forest. In: Kolman, P., Kratochvíl, J. (eds.) WG 2011. LNCS, vol. 6986, pp. 119–130. Springer, Heidelberg (2011)
7. Couturier, J.F., Golovach, P.A., Kratsch, D., Paulusma, D.: On the parameterized complexity of coloring graphs in the absence of linear forest. Journal of Discrete Algorithms 15, 56–62 (2012)
8. Dabrowski, K., Lozin, V., Müller, H., Rautenbach, D.: Parameterized complexity of the weighted independent set problem beyond graphs of bounded clique number. Journal of Discrete Algorithms 14, 207–213 (2012)
9. Dabrowski, K., Lozin, V., Raman, R., Ries, B.: Colouring vertices of triangle-free graphs without forests. Discrete Mathematics 312, 1372–1385 (2012)
10. Diestel, R.: Graph Theory, Electronic Edition. Springer (2005)
11. Golovach, P.A., Paulusma, D.: List coloring in the absence of two subgraphs. In: Spirakis, P.G., Serna, M. (eds.) CIAC 2013. LNCS, vol. 7878, pp. 288–299. Springer, Heidelberg (2013)
12. Golovach, P.A., Paulusma, D., Song, J.: 4-Coloring H-free graphs when H is small. Discrete Applied Mathematics 161, 140–150 (2013)
13. Golovach, P.A., Paulusma, D., Song, J.: Closing complexity gaps for coloring problems on H-free graphs. In: Chao, K.-M., Hsu, T.-s., Lee, D.-T. (eds.) ISAAC 2012. LNCS, vol. 7676, pp. 14–23. Springer, Heidelberg (2012)
14. Gravier, S., Hoáng, C.T., Maffray, F.: Coloring the hypergraph of maximal cliques of a graph with no long path. Discrete Mathematics 272, 285–290 (2003)
15. Gyárfás, A.: Problems from the world surrounding perfect graphs. Zastosowania Matematyki Applicationes Mathematicae XIX(3-4), 413–441 (1987)
16. Hoàng, C.T., Kamiński, M., Lozin, V., Sawada, J., Shu, X.: Deciding k-colorability of P_5-free graphs in polynomial time. Algorithmica 57, 74–81 (2010)
17. Hoàng, C.T., Maffray, F., Mechebbek, M.: A characterization of b-perfect graphs. Journal of Graph Theory 71, 95–122 (2012)
18. Král', D., Kratochvíl, J., Tuza, Z., Woeginger, G.J.: Complexity of coloring graphs without forbidden induced subgraphs. In: Brandstädt, A., Le, V.B. (eds.) WG 2001. LNCS, vol. 2204, pp. 254–262. Springer, Heidelberg (2001)
19. Kobler, D., Rotics, U.: Edge dominating set and colorings on graphs with fixed clique-width. Discrete Applied Mathematics 126, 197–221 (2003)
20. Maffray, F., Preissmann, M.: On the NP-completeness of the k-colorability problem for triangle-free graphs. Discrete Mathematics 162, 313–317 (1996)
21. Randerath, B.: 3-colorability and forbidden subgraphs. I., Characterizing pairs. Discrete Mathematics 276, 313–325 (2004)
22. Randerath, B., Schiermeyer, I.: A note on Brooks' theorem for triangle-free graphs, Australas. J. Combin. 26, 3–9 (2002)
23. Randerath, B., Schiermeyer, I.: Vertex colouring and forbidden subgraphs - a survey. Graphs and Combinatorics 20, 1–40 (2004)
24. Randerath, B., Schiermeyer, I.: On maximum independent sets in P_5-free graphs. Discrete Applied Mathematics 158(9), 1041–1044 (2010)
25. Schindl, D.: Some new hereditary classes where graph coloring remains NP-hard. Discrete Mathematics 295, 197–202 (2005)
26. Tuza, Z.: Graph colorings with local restrictions - a survey, Discuss. Math. Graph Theory 17, 161–228 (1997)

Lower and Upper Bounds for Long Induced Paths in 3-Connected Planar Graphs

Emilio Di Giacomo[1], Giuseppe Liotta[1], and Tamara Mchedlidze[2]

[1] Università degli Studi di Perugia, Italy
[2] Karlsruhe Institute of Technology (KIT), Germany

Abstract. Let G be a 3-connected planar graph with n vertices and let $p(G)$ be the maximum number of vertices of an induced subgraph of G that is a path. We prove that $p(G) \geq \frac{\log n}{12 \log \log n}$. To demonstrate the tightness of this bound, we notice that the above inequality implies $p(G) \in \Omega((\log_2 n)^{1-\varepsilon})$, where ε is any positive constant smaller than 1, and describe an infinite family of 3-connected planar graphs for which $p(G) \in O(\log n)$. As a byproduct of our research, we prove a result of independent interest: Every 3-connected planar graph with n vertices contains an induced subgraph that is outerplanar and connected and that contains at least $\sqrt[3]{n}$ vertices. The proofs in the paper are constructive and give rise to $O(n)$-time algorithms.

1 Introduction

Determining whether a graph has a large subset of vertices that induce a path is a well studied problem motivated by various applications including, for example, the performance analysis of large communication and neural networks (see, e.g., [8]). Garey and Johnson [7] prove the NP-completeness of computing the longest induced path for general graphs and Lund and Yannakakis [14] show that this maximization problem is not approximable within $O(n^{1-\varepsilon})$ for any positive constant ε. Exact algorithms that have polynomial time complexity are also known for specific graph families (see, e.g. [9,8,11,12]), while several papers study the complexity of various network optimization problems under the assumption that the length of the longest induced path is bounded (see, e.g. [2,13,17]).

From a graph theoretical point of view, Erdös, Saks, and Sós [5] prove a lower bound on the length of the longest induced path in a connected graph G in terms of the radius of G. They show that $p(G) \geq 2r(G) - 1$, where $p(G)$ denotes the maximum number of vertices in an induced path of a connected graph G and $r(G)$ is the radius of G. Since, in general, the radius is not bounded from below by a function of n, the result by Erdös, Saks, and Sós naturally raises the question about whether the length of the longest path tends to infinity as the size of the graph tends to infinity. Clearly, this question becomes interesting for graphs that are not too dense; namely, the longest induced path in a complete graph consists of a single edge.

For 3-connected planar graphs, a positive answer to the above question is a consequence of a Ramsey-type result by Böhme et al. [3]. Namely, in [3] it is proved that for every positive integers k, r, s, there exists an integer $n = n(k, r, s)$ such that any

k-connected graph with at least n vertices has either an induced path of length s or a subdivision of $K_{k,r}$. Hence, by setting $k = r = 3$, we obtain that for every positive integer s there exists a sufficiently large 3-connected planar graph having an induced path of length s. However, the construction by Böhme et al. does not yield an explicit function that defines a lower bound on the length of the longest induced path, which can be found in a paper by Arocha and Valencia [1]. Let G be a 3-connected planar graph with n vertices and having maximum degree Δ; Arocha and Valencia observe that if Δ is bounded by a constant, then G has a diameter (and hence an induced path) consisting of at least $\log_{\Delta} n$ vertices. If otherwise Δ is not a constant with n, then G has an induced outerplanar graph with at least Δ vertices from which an induced path with at least $\sqrt{\log_3 \Delta}$ vertices can be extracted.

The main contribution of this paper is to find bounds for $p(G)$ that improve those in [1,3]. Similar to Arocha and Valencia, we first construct an induced outerplanar graph H and then compute an induced path in H; however, our approach finds significantly larger outerplanar graphs and significantly larger paths than those described in [1]. More precisely, the results in this paper can be listed as follows.

- A 3-connected planar graph G with n vertices has an induced subgraph with at least $\sqrt[3]{n}$ vertices that is outerplanar and connected. In the case that the external face of G is a 3-cycle, the induced outerplanar graph is 2-connected. This result is of independent interest because it improves a previous bound by Goaoc et al. [10] who show that a maximal planar graph with n vertices has a 2-connected induced outerplanar subgraph with $\Omega(\frac{\log n}{\log \log n})$ vertices.
- Every 3-connected planar graph G with n vertices has an induced path with at least $\frac{\log n}{12 \log \log n}$ vertices. We also prove that for every n there exists a 3-connected planar graph G with n vertices such that the longest induced path of G has at most $2 \log_3(2n-5)+3$ vertices.
- Asymptotically, the gap between the upper and lower bounds described in the item above is arbitrarily small. Namely, a consequence of the above result is that for any given positive constant ε smaller than 1 and for n that tends to infinity, every 3-connected planar graph has an induced path with at least $(2 \log_3(2n-5)+3)^{1-\varepsilon}$ vertices.

Our arguments combine various techniques. We use the extension of Schnyder woods to 3-connected planar graphs (see, e.g. [4,6]) to prove that there exist three partial orders by which any two vertices of 3-connected planar graph can be compared. We then exploit Mirsky's theorem [15] to extract a large induced outerplanar graph H; finally, we analyze the structure of the extended dual of H to compute a long induced path. The proofs for the lower bound are constructive and give rise to a linear-time algorithm to compute a long induced path in a 3-connected planar graph.

The rest of the paper is organized as follows. Preliminaries are in Section 2. The computation of a large induced outerplanar graph is described in Section 3. Lower and upper bounds on the length of the longest induced path are given in Sections 4 and 5. Finally, conclusions and open problems can be found in Section 6. For reasons of space, some proofs are omitted.

2 Preliminaries

A subgraph H of a graph G is said to be *induced* if, for any pair of vertices u and v of H, (u, v) is an edge of H if and only if (u, v) is an edge of G. We denote by $p(G)$ the number of vertices of the longest induced path of G.

The *connectivity* of a graph is the minimum number of vertices whose removal results in a disconnected graph or a single vertex graph. A graph is k*-connected* if its connectivity is at least k ($k \geq 1$). Notice that a k-connected graph has at least $k+1$ vertices. A *cutvertex* is a vertex whose removal disconnect the graph. Let G be a 1-connected graph (also called a connected graph); the maximal subgraphs of G not containing a cutvertex is a 2*-connected component* of G. Notice that a 2-connected component of a connected graph is either a 2-connected subgraph or a single edge.

A graph G is planar if it can be drawn in the plane without edge crossings. A planar drawing Γ partitions the plane into topologically connected regions called *faces*; the unbounded region is the *external* face. A planar drawing of a planar graph determines a circular ordering of the edges around each vertex. The cyclic ordering of the edges around each vertex of Γ together with a choice of the external face is a *planar embedding* of G. A *plane graph* is a graph with a fixed planar embedding. The boundary of the external face of a plane graph G will be also called the *external boundary* of G.

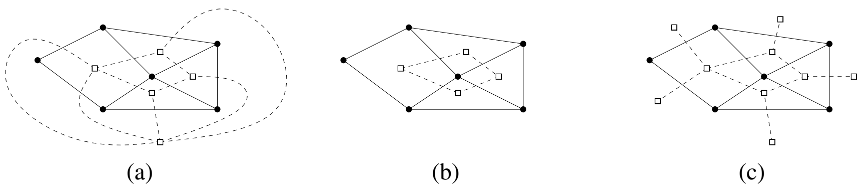


Fig. 1. (a) A graph G and its dual graph G^*: the black dots are the vertices of G, while the white squares are those of G^*; the solid edges are the edges of G, while the dashed edges are those of G^*. (b) The weak dual of G. (c) The extended dual of G.

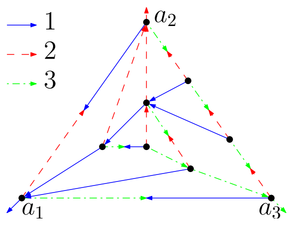
The *dual graph* G^d of a given plane graph G is a plane multigraph that has a vertex corresponding to each face of G, and an edge joining two vertices corresponding to neighboring faces of G (see for example Figure 1(a)). The *weak dual* G^w of G is the graph obtained from G^d after removing the vertex that corresponds to the external face of G (see for example Figure 1(b)). The *extended dual* of G is the graph G^* obtained from G^d by splitting the vertex f_{ext} of G^d corresponding to the external face of G into $\deg(f_{ext})$ vertices each incident to one of the edges incident to f_{ext} (see for example Figure 1(c)). Notice that the dual G^d of a plane graph G has a vertex for each face of G and an edge for each edge of G; the weak dual G^w of G has a vertex for each internal face of G and an edge for each internal edge of G; the extended dual G^* of G has a vertex for each internal face, k vertices corresponding to the external face (where k is the number of edges on the external boundary), and an edge for each edge of G. A graph is *outerplanar* if it admits a planar embedding where all vertices are on the external boundary. It is easy to see that the weak dual of an outerplanar graph is always a tree and the same is true for the extended dual. An *outerpath* is an outerplanar graph whose weak dual is a path, hence an outerpath has at least one internal face. A *bracelet* is a connected outerplanar graph such that each cutvertex is shared by two 2-connected components and each 2-connected component has at most two cutvertices. Notice that a 2-connected component of a connected graph can be a single edge.

3 Large Induced Outerplanar Subgraphs

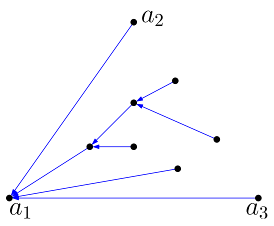
We start this section by introducing the Schnyder woods [16], which are the basic tool to prove that every 3-connected planar graph contains an outerplanar subgraph with at least $\sqrt[3]{n}$ vertices. Schnyder woods were first introduced for maximal planar graphs [16], and have been then generalized to 3-connected planar graphs (see, e.g., [4,6]).

Schnyder woods. The following definition of Schnyder woods adopts the same notation and terminology used in [6]. Let G be a plane graph and let a_1, a_2, a_3 be three vertices occurring in clockwise order on the outer face of G. A *suspension* G^σ of G is obtained by attaching a half-edge that reaches into the outer face to each of these special vertices. Let G^σ be a suspended 3-connected plane graph. A *Schnyder wood* rooted at a_1, a_2, a_3 is an orientation and coloring of the edges of G^σ with the colors 1, 2, 3 satisfying the following properties[1] (see Figure 2(a) for an illustration).

(W1) Every edge e is oriented in one direction or in two opposite directions. The directions of edges are colored such that if e is bidirected the two directions have distinct colors.
(W2) The half-edge at a_i is directed outwards and colored i.
(W3) Every vertex v has out-degree one in each color. The edges e_1, e_2, e_3 leaving v in colors 1, 2, 3 occur in clockwise order. Each edge entering v in color i enters v in the clockwise sector from e_{i+1} to e_{i-1}.
(W4) There is no interior face the boundary of which is a directed monochromatic cycle.



(a) $\mathcal{S}$



(b) T_1

Fig. 2. (a) A Schnyder wood $\mathcal{S}$ of a 3-connected plane graph. (b) the tree T_1.

Every suspension of a 3-connected plane graph has a Schnyder wood that can be computed in linear time [4]. Notice that, in the definition of Schnyder woods the addition of the half edges to a_1, a_2, and a_3 is used to make property (W3) hold for every

[1] We assume a cyclic structure on the labels so that $i + 1$ and $i - 1$ are always defined.

vertex of G (otherwise it will not hold for a_1, a_2, and a_3). In what follows we will often refer to the Schnyder wood of a 3-connected plane graph without choosing a suspension explicitly. Also, from now on we will ignore the half-edges at a_1, a_2, and a_3. We denote by T_1, T_2, and T_3 the subgraph induced by the edges of color 1, 2, and 3, respectively and by $\mathcal{S}$ the Schnyder wood $\{T_1, T_2, T_3\}$ of G. Let T_i^{-1} be the subgraph obtained from T_i by reversing all its edges. The following two properties of $\mathcal{S}$ are known (see [6]).

Property 1. Let $\mathcal{S}$ be a Schnyder wood of a 3-connected planar graph G. The digraph $D_i = T_i \cup T_{i-1}^{-1} \cup T_{i+1}^{-1}$ is acyclic ($i = 1, 2, 3$).

Property 2. Let $\mathcal{S}$ be a Schnyder wood of a 3-connected planar graph G. T_i is a tree rooted at a_i ($i = 1, 2, 3$).

The tree T_1 defined by the Schnyder wood of Figure 2(a) is shown in Figure 2(b). Let $\mathcal{S}$ be a Schnyder wood of a 3-connected plane graph G, let v be an internal vertex and let $P_i(v)$, $1 \leq i \leq 3$ be the oriented path of T_i from v to a_i. The path $P_i(v)$ is called the *i-path starting at v*. For $i \neq j$ ($1 \leq i, j \leq 3$), the two paths $P_i(v)$ and $P_j(v)$ only share vertex v. Thus, for each internal vertex v, the three paths $P_1(v)$, $P_2(v)$, and $P_3(v)$ divide G into three regions $R_1(v)$, $R_2(v)$, and $R_3(v)$, where $R_i(v)$ ($1 \leq i \leq 3$) denotes the vertices that are inside the cycle[2] $P_{i-1}(v) \cup P_{i+1}(v) \cup P_{i+1}(a_{i-1})$ (see Figure 3(a) for an illustration).

We now use the Schnyder wood $\mathcal{S}$ to define three different partial orders on the vertices $V(G)$ of G. For each $i = 1, 2, 3$, we define a directed graph $G_i = T_i \cup T_{i+1}^{-1}$ (see Figure 3(b)). By Property 1, G_i is a directed acyclic graph with one source a_{i+1} and one sink a_i. Thus, it defines a partial order $\prec_i$ for the vertices of G; namely, $u \prec_i v$ if there is a directed path from u to v in G_i. Given a set X with a partial order $\prec$ (also called a *partial ordered set*), a subset of X such that every two of its elements are comparable with respect to $\prec$ is called a *chain* of X. A subset of X such that no two of its elements are comparable with respect to $\prec$ is called an *antichain* of X. According to the definition of $\prec_i$, a chain χ of $V(G)$ with respect to $\prec_i$ corresponds to a oriented path π in G_i. We say that π is the oriented path of G_i *associated with* χ. We prove now some properties of the partial orders $\prec_i$.

Lemma 1. *Let $\mathcal{S}$ be a Schnyder wood of a 3-connected planar graph G. Let u and v be two vertices of G. Then u and v are comparable by at least one of the three partial orders $\prec_1$, $\prec_2$, and $\prec_3$.*

Proof. If u belongs to a i-path starting at v for some i ($1 \leq i \leq 3$), then u and v are comparable with respect to both $\prec_i$ and $\prec_{i-1}$. Namely, the concatenation of $P_i(v)$ with the reversed version of $P_{i+1}(v)$ is an oriented path of G_i containing both u and v, and therefore u and v are comparable with respect to $\prec_i$. Analogously, the concatenation of $P_{i-1}(v)$ with the reversed version of $P_i(v)$ is an oriented path of G_{i-1} containing both u and v, and therefore u and v are comparable with respect to $\prec_{i-1}$. Clearly, the same argument applies if v belongs to an i-path starting at u for some i ($1 \leq i \leq 3$).

[2] In the literature these regions are usually defined as including also $P_1(v)$, $P_2(v)$, and $P_3(v)$. For our purposes, however, it results more useful to use this different definition.

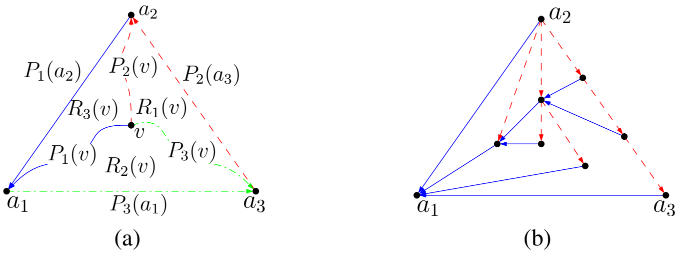


Fig. 3. The three paths $P_1(v)$, $P_2(v)$, and $P_3(v)$, and the three regions $R_1(v)$, $R_2(v)$, and $R_3(v)$. (b) The directed graph G_1 used to define the partial order $\prec_1$.

Consider the case when u does not belong to any i-path starting at v and vice versa. It follows that u must be in one region $R_i(v)$ $(1 \leq i \leq 3)$. Consider the i-path starting at u, $P_i(u)$. Since this path does not contain v, it must share a vertex w with either $P_{i-1}(v)$ or $P_{i+1}(v)$ (see Figure 4(a)). Suppose that w belongs to $P_{i-1}(v)$, the case when it belongs to $P_{i+1}(v)$ is analogous. Consider the path P consisting of the concatenation of: (i) the subpath of $P_{i-1}(v)$ from v to w; (ii) the reversed version of the subpath of $P_i(u)$ from w to v. Path P is an oriented path of G_{i-1} from v to u. Thus, u and v are comparable with respect to $\prec_{i-1}$. $\square$

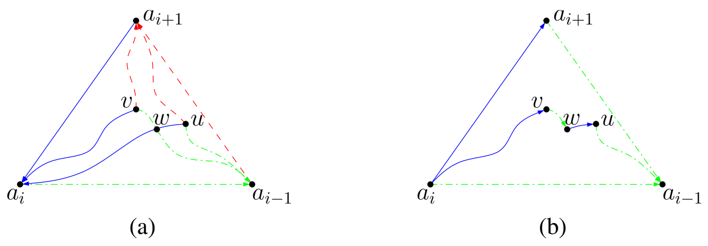


Fig. 4. An illustration for the proof of Lemma 1. (a) The path $P1(u)$ crosses the path $P_3(v)$. (b) The oriented path containing u and v in G_3.

Lemma 2. *Let $\mathcal{S}$ be a Schnyder wood of a 3-connected planar graph G. Let u be an internal vertex of G and let v be another vertex of G that is comparable with u by $\prec_i$. If $v \prec_i u$ then $v \in R_i(u) \cup P_{i+1}(u)$. If $u \prec_i v$ then $v \in R_{i+1}(u) \cup P_i(u)$.*

In the next lemma we show that any maximal chain of $V(G)$ with respect to any partial order $\prec_i$ defines an induced bracelet of G.

Lemma 3. *Let $\mathcal{S}$ be a Schnyder wood of a 3-connected planar graph G. Let χ be a maximal chain of $V(G)$ with respect to the partial order $\prec_i$, $1 \leq i \leq 3$ and let π be the oriented path associated with χ. The graph induced by the vertices of π is an induced bracelet of G.*

Proof. Since G_i has a single source a_{i+1} and single sink a_i, then every maximal chain of $V(G)$ with respect to $\prec_i$ has r_{i+1} as the first element and r_i as the last element. As a consequence π starts at r_{i+1} and ends at r_i. We first prove that the subgraph H induced by the vertices in π is outerplanar. To this aim, it is sufficient to show that all edges of H that are not in π are to right of π when walking along π from a_{i+1} to a_i. Consider an edge (u, v) of H not in π. Vertices u and v must be two non-consecutive vertices of π. Without loss of generality assume $u \prec_i v$, i.e., u is encountered first when walking along π from a_{i+1} to a_i. Since u and v are non-consecutive, there must be a vertex w such that $u \prec_i w \prec_i v$. By Lemma 2, u is in $R_i(w) \cup P_{i+1}(w)$ and v is in $R_{i+1}(w) \cup P_i(w)$. As a consequence, if edge (u, v) is not to the right of π, then it crosses path $P_{i-1}(w)$, which is not possible.

In order to prove that H is a bracelet, we need to show that: (i) each cutvertex of H is shared by two 2-connected components and (ii) each 2-connected component has at most two cutvertices. Suppose that there exists a cutvertex v shared by at least three 2-connected components. Then there exist three vertices w_1, w_2 and w_3 such that every path connecting w_i to w_j $(1 \leq i, j \leq 3, i \neq j)$ contains v. On the other hand, w_1, w_2 and w_3 belong to π and therefore at least two of them, say w_1 and w_2, either precede or follow v along π; but this means that w_1 and w_2 are connected by a path that does not contain v. Suppose now that there exists a 2-connected component B that has at least three cutvertices. Also in this case there exist three vertices w_1, w_2 and w_3 such that every path connecting w_i to w_j $(1 \leq i, j \leq 3, i \neq j)$ contains a vertex of B. The vertices of B are consecutive along π and therefore at least two between w_1, w_2 and w_3, say w_1 and w_2, either precede or follow the vertices of B along π; but then w_1 and w_2 are connected by a path that does not contain any vertex of B. □

The next lemma shows that the bracelet of Lemma 3 has at least $\sqrt[3]{n}$ vertices.

Lemma 4. *Let $\mathcal{S}$ be a Schnyder wood of a 3-connected planar graph G. $V(G)$ has a chain of size at least $\sqrt[3]{n}$ with respect to one of the three partial orders $\prec_1$, $\prec_2$, and $\prec_3$.*

Proof. Consider first the partial order $\prec_1$. If $V(G)$ has a chain of size at least $\sqrt[3]{n}$ with respect to $\prec_1$, then the statement is true. If not, by Mirsky's theorem [15] (the dual of Dilworth's theorem), $V(G)$ has a partition into at most $\sqrt[3]{n}$ antichains with respect to $\prec_1$. Hence, one of this antichains must have at least $\frac{n}{\sqrt[3]{n}} = n^{2/3}$ vertices. Let U be such a set of vertices. Since the vertices in U form an antichain of $V(G)$ with respect to $\prec_1$, then, by Lemma 1, any two vertices in U must be comparable by $\prec_2$ or by $\prec_3$. Consider $\prec_2$ restricted to the set U; if U has a chain of size at least $\sqrt[3]{n}$ with respect to $\prec_2$, then the statement is true. Otherwise, applying again Mirsky's theorem, U can be partitioned into at most $\sqrt[3]{n}$ antichains. One of this antichain must have at least $\frac{n^{2/3}}{n^{1/3}} = \sqrt[3]{n}$ vertices. Let U' be such a set of vertices. The vertices in U' are not comparable with respect to $\prec_1$, neither with respect to $\prec_2$. Thus by Lemma 1 they must be comparable by $\prec_3$, i.e., they form a chain of U' with respect to $\prec_3$. □

Theorem 1. *Let G be a 3-connected planar graph with n vertices. G contains an induced connected outerplanar graph H. The subgraph H is a bracelet with at least $\sqrt[3]{n}$ vertices and can be computed in $O(n)$ time.*

Proof. Consider a Schnyder wood $\mathcal{S}$ of G and define the three partial orders $\prec_1$, $\prec_2$, and $\prec_3$. By Lemma 4, $V(G)$ has a chain of size at least $\sqrt[3]{n}$ with respect to one of these partial orders. From Lemma 3 it follows that G contains an induced bracelet H with at least $\sqrt[3]{n}$ vertices.

The Schnyder wood $\mathcal{S}$ can be computed in $O(n)$ time [4]. Using $\mathcal{S}$, the three graphs G_1, G_2, and G_3 can be also computed in $O(n)$ time. By computing a longest path in each of these directed graphs (which can be done in $O(n)$ time) we find a longest chain of $V(G)$ with respect to one of the three partial orders $\prec_1$, $\prec_2$, and $\prec_3$. The subgraph induced by the vertices of this path is H. □

The following corollary shows that if G is a maximal plane graph, then the bracelet of Theorem 1 is in fact a 2-connected outerplanar graph.

Corollary 1. *Let G be a 3-connected plane graph with n vertices. If the external boundary of G is a 3-cycle, then G contains an induced 2-connected outerplanar graph with at least $\sqrt[3]{n}$ vertices, which can be computed in $O(n)$ time.*

The result stated in Corollary 1 holds, in particular, for maximal plane graph, in which case it improves a previous result by Goaoc et al. [10]. Namely, Theorems 4.7 and 4.11 of [10] imply that a maximal planar graph G contains an induced 2-connected outerplanar graph of size $\Omega\left(\frac{\log_2 n}{\log_2 \log_2 n}\right)$.

4 Lower Bound

In this section we prove that a 3-connected planar graph contains a long induced path. Based on Theorem 1 it is sufficient to prove that a bracelet has a long induced path. The next three lemmas are used to prove this.

Lemma 5. *Let G be an outerpath with n vertices. Then $p(G) \geq \frac{n}{2}$.*

Proof. Since G is an outerpath its weak dual is a path Π. Let f_1 and f_2 be the two faces corresponding to the endvertices of Π (f_1 coincides with f_2 if Π is a single vertex). Let v_1 and v_2 be two vertices belonging to f_1 and f_2, respectively, and not belonging to any other internal face (such two vertices always exist in an outerpath). The external circuit of G is divided into two paths Π_1 and Π_2 by v_1 and v_2. Both Π_1 and Π_2 are induced, and one of them contains at least $\frac{n}{2}$ vertices. □

Lemma 6. *Let G be a 2-connected outerplanar graph with n vertices ($n \geq 3$). Then $p(G) \geq \frac{1}{2}\frac{\log_2 n}{\log_2 \log_2 n}$.*

Proof. First of all, notice that if $n \leq 4$, then $\frac{1}{2}\frac{\log_2 n}{\log_2 \log_2 n} \leq 2$ and the statement trivially holds. So in the following we assume $n > 4$. Denote by $\alpha(n)$ the function $\frac{\log_2 n}{\log_2 \log_2 n}$ which is defined for every $n > 4$. Let G^* be the extended dual of G. Since G is outerplanar, G^* is a tree. We distinguish two cases:

There exists a node of G^* whose degree is larger than $\alpha(n)+1$. In this case there exists a face with at least $\alpha(n)+2$ vertices. The boundary of such a face is an induced cycle with $\alpha(n)+2$ vertices and edges. Removing a vertex from this cycle, we obtain an induced path with at least $\alpha(n)+1$ vertices. We have $\alpha(n)+1 = \frac{\log_2 n}{\log_2 \log_2 n} + 1 \geq \frac{1}{2}\frac{\log_2 n}{\log_2 \log_2 n}$.

Every node of G^* has degree at most $\alpha(n)+1$. Denote by $N(n)$ the number of nodes of G^*. Since every node of G^* has degree at most $\alpha(n)+1$, G^* contains a path Π with at least $h(n)$ vertices, where$h(n) \geq 2\log_{\alpha(n)}\left(\frac{(N(n)-1)(\alpha(n)-1)+\alpha(n)+1}{\alpha(n)+1}\right)+1$ (notice that this function is defined and it is not equal to one for every $n>4$). The subgraph G' of G induced by the faces corresponding to the nodes of Π is an induced outerpath of G. By Lemma 5 G', and therefore G, has an induced path with at least $\frac{h(n)}{2}$ vertices. We now write the function $h(n)$ in explicit form. The number of nodes of G^* is $N(n) = f_{int}(n) + n$, where $f_{int}(n)$ is the number of internal faces of G. Namely, the extended dual contains a vertex for each internal face of G^* and a vertex for each edge of the external face (the external boundary has n edges because G is 2-connected). Furthermore, since the degree of G^* is at most $\alpha(n)+1$, each internal face has at most $\alpha(n)+1$ edges. Thus, $\frac{f_{int}(n)\cdot(\alpha(n)+1)+n}{2} \geq m$. By Euler's formula, $n + f_{int}(n) + 1 \leq \frac{f_{int}(n)\cdot(\alpha(n)+1)+n}{2} + 2$, from which we obtain $f_{int}(n) \geq \frac{n-2}{\alpha(n)-1}$ and $h(n) \geq \log_{\alpha(n)} \frac{n\cdot\alpha(n)}{\alpha(n)+1} + 1$. We have $\log_{\alpha(n)} \frac{n\cdot\alpha(n)}{\alpha(n)+1} + 1 = \log_{\alpha(n)} n + \log_{\alpha(n)} \alpha(n) - \log_{\alpha(n)}(\alpha(n)+1) + 1 = \log_{\alpha(n)} n + 2 - \log_{\alpha(n)}(\alpha(n)+1)$. Since $\log_{\alpha(n)}(\alpha(n)+1) \leq 2$ for every $n>4$, we have $\log_{\alpha(n)} n + 2 - \log_{\alpha(n)}(\alpha(n)+1) \geq \log_{\alpha(n)} n + 2 - 2 = \log_{\alpha(n)} n$. Thus we obtain $h(n) \geq \log_{\alpha(n)} n$. Now, $\log_{\frac{\log_2 n}{\log_2\log_2 n}} n = \frac{\log_2 n}{\log_2 \frac{\log_2 n}{\log_2 \log_2 n}} = \frac{\log_2 n}{\log_2\log_2 n - \log_2\log_2\log_2 n}$.
Since $\log_2\log_2\log_2 n \geq 0$ for $n \geq 4$, we obtain $h(n) \geq \frac{\log_2 n}{\log_2 \log_2 n}$. Thus G has an induced path with at least $\frac{1}{2}\frac{\log_2 n}{\log_2\log_2 n}$ vertices. □

Lemma 7. *Let G be a bracelet with n vertices ($n \geq 3$). Then $p(G) \geq \frac{1}{4}\frac{\log_2 n}{\log_2\log_2 n}$.*

Proof. If $n < 9$, then $\frac{1}{4}\frac{\log_2 n}{\log_2\log_2 n} < 2$ and the statement is trivially true. So assume $n \geq 9$. Let k be the number of cut-vertices of G. We distinguish two cases:

Case 1: $k \leq \sqrt{n}-1$. In this case, there are at most $\sqrt{n}$ 2-connected components. It follows that there exists a 2-connected component G' that contains n' vertices with $n' \geq \sqrt{n}$. Since $n > 9$, we have $n' \geq 3$ and by Lemma 6, G' has an induced path with at least $\frac{1}{2}\frac{\log_2 n'}{\log_2\log_2 n'} = \frac{1}{2}\frac{\log_2 n^{\frac{1}{2}}}{\log_2\log_2 n^{\frac{1}{2}}} = \frac{1}{4}\frac{\log_2 n}{\log_2 \frac{1}{2} + \log_2\log_2 n} = \frac{1}{4}\frac{\log_2 n}{\log_2\log_2 n - 1} \geq \frac{1}{4}\frac{\log_2 n}{\log_2\log_2 n}$.

Case 2: $k > \sqrt{n}-1$. In this case, there are at least $\sqrt{n}$ 2-connected components. Since each cutvertex is shared by two 2-connected components and each 2-connected component has at most two cutvertices, the $\sqrt{n}+2$ 2-connected components form a sequence $G_1, G_2, \ldots, G_{k+1}$, where G_i and G_{i+1} share a cutvertex c_i. Since in each 2-connected component there exists an induced path (possibly consisting of a single edge) connecting c_{i-1} to c_i, we have an induced path containing all cutvertices, i.e., containing at least $\sqrt{n}-1 \geq \frac{1}{4}\frac{\log_2 n}{\log_2\log_2 n}$ vertices. □

Lemma 7 and Theorem 1 imply the following.

Lemma 8. *Let G be a 3-connected planar graph with n vertices. Then $p(G) \geq \frac{1}{12} \frac{\log_2 n}{\log_2 \log_2 n}$.*

5 Upper Bound and Main Theorem

Lemma 8 gives a lower bound to the length of an induced path in a 3-connected planar graph. Thus, it is natural to ask for an upper bound. In the following we prove that there exists an n-vertex 3-connected planar graph G such that every induced path of G has $O(\log n)$ vertices. It is worth remarking that an analogous upper bound is provided by Arocha and Valencia [1] for outerplanar graphs.

A *complete planar* 3*-tree* is a 3-connected planar graph recursively defined as follows. The graph G_0 consisting of a 3-cycle is a complete planar 3-tree. The graph G_i obtained by inserting a vertex v inside each internal face u_1, u_2, u_3 of G_{i-1} and connecting v to each u_i ($i = 1, 2, 3$) is a complete planar 3-tree. The face u_1, u_2, u_3 is called the *attaching face* of v. Let F be a subset of the internal faces of a complete planar 3-tree G_{i-1}; the graph $\overline{G}_i$ obtained by inserting a vertex v inside each face u_1, u_2, u_3 of F and connecting v to each u_i ($i = 1, 2, 3$) is an *almost complete planar* 3*-tree*. Notice that a complete planar 3-tree is also an almost complete planar 3-tree.

Given an almost complete planar 3-tree $\overline{G}_i$, we assign a value, called *level of* v and denoted as $lev(v)$, to each vertex v of $\overline{G}_i$ as follows. If v is a vertex of G_0, then $lev(v) = 0$. If v is a vertex of $\overline{G}_i$ but it is not a vertex of G_{i-1}, then $lev(v) = i$.

Lemma 9. *Let $\overline{G}_i$ be an almost complete planar 3-tree and let Π be an induced path of $\overline{G}_i$. There does not exist two consecutive edges (w_1, v) and (v, w_2) in Π such that $lev(w_1) < lev(v)$ and $lev(w_2) < lev(v)$.*

Proof. Let $lev(v) = j$. The only vertices of $\overline{G}_i$ adjacent to v and having level less than j are the three vertices of the attaching face of v (which form a face of G_{j-1}). Thus, if $lev(w_1) < lev(v)$ and $lev(w_2) < lev(v)$, then w_1 and w_2 would be adjacent and Π would not be an induced path. □

Lemma 10. *For every $n \geq 3$ there exists a 3-connected planar graph G such that $p(G) \leq 2\log_3(2n-5)+3$.*

Proof. A complete planar 3-tree G_i has $\frac{3^i+5}{2}$ vertices. Given a value of n, let i be the integer such that $\frac{3^{i-1}+5}{2} \leq n < \frac{3^i+5}{2}$. Let G be an almost complete planar 3-tree $\overline{G}_i$ with n vertices (i.e., an almost complete planar 3-tree obtained from G_{i-1} by adding $n - \frac{3^{i-1}+5}{2}$ vertices). Let Π be an induced path of $\overline{G}_i$; by Lemma 9, Π is such that either the levels of the vertices monotonically increase along Π, or they monotonically decrease, or they form a first sequence monotonically decreasing followed by a second sequence monotonically increasing. Since the range of the values for the level of the vertices is $[0, i]$, each monotone sequence has at most $i+1$ vertices, and Π has at most $2i+1$ vertices. It follows that $p(G) \leq 2i+1 \leq 2\log_3(2n-5)+3$. □

We are now ready to prove the main result of this paper, where we summarize the results given above and also show that the gap between the lower bound of Lemma 8 and the upper bound of Lemma 10 is arbitrarily small in asymptotic terms.

Theorem 2. *Let $\mathcal{G}_n$ be the set of the 3-connected planar graphs having n vertices. Let $p_n = \min_{G \in \mathcal{G}_n}\{p(G)\}$. Then:*

(i) $\frac{\log n}{12 \log\log n} \leq p_n \leq 2\log_3(2n-5)+3$, *for any value of n.*

(ii) $(2\log_3(2n-5)+3)^{1-\varepsilon} \leq p_n \leq 2\log_3(2n-5)+3$, *for every ε such that $0 < \varepsilon < 1$ and n that tends to infinity.*

Furthermore, there exists an $O(n)$-time algorithm that, for any $G \in \mathcal{G}_n$, computes an induced path of having at least $\frac{\log n}{12 \log\log n}$ vertices.

Proof. The upper and lower bound expressed by (i) follow from Lemmas 8 and 10. The lower bound of (ii) immediately follows from:

$$\lim_{n \to +\infty} \frac{(2\log_3(2n-5)+3)^{1-\varepsilon}}{\frac{\log n}{12 \log\log n}} = 0.$$

As for the time complexity, a bracelet H of size at least $\sqrt[3]{n}$ can be computed in $O(n)$ time by Theorem 1; in $O(n)$ time we can compute the cutvertices of H and decide which case of Lemma 7 applies. If we are in Case 2, we can easily compute the desired path in $O(n)$ time by performing a BFS visit of each 2-connected component; if we are in Case 1, we need to compute the extended dual H^* of H and decide which case of Lemma 6 applies; both things can be done in $O(n)$ time. If we are in Case 1, we compute in $O(n)$ time the desired path (it is obtained by removing a vertex from the boundary of the face corresponding the high degree vertex of H^*); if we are in Case 2 we compute the longest path in H^* in $O(n)$ time and then we use it to compute the corresponding outerpath and from this the desired path; both things can be done in $O(n)$ time. □

6 Conclusions and Open Problems

In this paper we proved that every 3-connected planar graph contains an induced path with at least $\frac{1}{12}\frac{\log_2 n}{\log_2 \log_2 n}$ vertices, which can be computed in $O(n)$ time; we also showed that for every $n \geq 3$ there exists a graph G such that $p(G) \in O(\log n)$. We also prove that, asymptotically speaking, the previous lower bound on the length of an induced path can be improved to $\Omega((\log_2 n)^{1-\varepsilon})$. In order to prove the above mentioned results, we proved that every 3-connected planar graph contains an induced connected outerplanar graph with at least $\sqrt[3]{n}$ vertices, which is a result of independent interest.

A list of open problems suggested by the results of this paper includes:

- Close the gaps between upper and lower bounds described by Theorem 2.
- Improve the bound of Theorem 1 or show that it is tight.
- Study the length of induced paths for general planar graphs.

References

1. Arocha, J.L., Valencia, P.: Long induced paths in 3-connected planar graphs. Discuss. Math. Graph Theory 20(1), 105–107 (2000)
2. Atminas, A., Lozin, V.V., Razgon, I.: Linear time algorithm for computing a small biclique in graphs without long induced paths. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 142–152. Springer, Heidelberg (2012)
3. Böhme, T., Mohar, B., Skrekovski, R., Stiebitz, M.: Subdivisions of large complete bipartite graphs and long induced paths in k-connected graphs. Journal of Graph Theory 45(4), 270–274 (2004)
4. Di Battista, G., Tamassia, R., Vismara, L.: Output-sensitive reporting of disjoint paths. Algorithmica 23, 302–340 (1999)
5. Erdös, P., Saks, M., Sós, V.T.: Maximum induced trees in graphs. Journal of Combinatorial Theory, Series B 41(1), 61–79 (1986)
6. Felsner, S., Zickfeld, F.: Schnyder woods and orthogonal surfaces. Discrete & Computational Geometry 40(1), 103–126 (2008)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
8. Gavril, F.: Algorithms for maximum weight induced paths. Inf. Process. Lett. 81(4), 203–208 (2002)
9. Gavril, F., Laredo, V.T., de Werra, D.: Chordless paths, odd holes, and kernels in graphs without m-obstructions. J. Algorithms 17(2), 207–221 (1994)
10. Goaoc, X., Kratochvíl, J., Okamoto, Y., Shin, C.-S., Spillner, A., Wolff, A.: Untangling a planar graph. Discrete & Computational Geometry 42(4), 542–569 (2009)
11. Ishizeki, T., Otachi, Y., Yamazaki, K.: An improved algorithm for the longest induced path problem on k-chordal graphs. Discrete Applied Mathematics 156(15), 3057–3059 (2008)
12. Kratsch, D., Müller, H., Todinca, I.: Feedback vertex set and longest induced path on AT-free graphs. In: Bodlaender, H.L. (ed.) WG 2003. LNCS, vol. 2880, pp. 309–321. Springer, Heidelberg (2003)
13. Lozin, V.V., Rautenbach, D.: Some results on graphs without long induced paths. Inf. Process. Lett. 88(4), 167–171 (2003)
14. Lund, C., Yannakakis, M.: The approximation of maximum subgraph problems. In: Lingas, A., Carlsson, S., Karlsson, R. (eds.) ICALP 1993. LNCS, vol. 700, pp. 40–51. Springer, Heidelberg (1993)
15. Mirsky, L.: A dual of Dilworth's decomposition theorem. The American Mathematical Monthly 78(8), 876–877 (1971)
16. Schnyder, W.: Embedding planar graphs on the grid. In: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1990, pp. 138–148 (1990)
17. Woeginger, G.J., Sgall, J.: The complexity of coloring graphs without long induced paths. Acta Cybern. 15(1), 107 (2001)

Computing Minimum Cycle Bases in Weighted Partial 2-Trees in Linear Time

Carola Doerr[1,2,⋆], G. Ramakrishna[3,⋆⋆], and Jens M. Schmidt[2]

[1] Université Paris Diderot - Paris 7, LIAFA, Paris, France
[2] Max Planck Institute for Informatics, Saarbrücken, Germany
[3] Indian Institute of Technology Madras, India

Abstract. We present a linear time algorithm for computing an implicit linear space representation of a minimum cycle basis (MCB) in weighted partial 2-trees, i.e., graphs of treewidth two. The implicit representation can be made explicit in a running time that is proportional to the size of the MCB.

For planar graphs, Borradaile, Sankowski, and Wulff-Nilsen [Min *st*-cut Oracle for Planar Graphs with Near-Linear Preprocessing Time, FOCS 2010] showed how to compute an implicit $O(n \log n)$ space representation of an MCB in $O(n \log^5 n)$ time. For the special case of partial 2-trees, our algorithm improves this result to linear time and space. Such an improvement was achieved previously only for outerplanar graphs [Liu and Lu: Minimum Cycle Bases of Weighted Outerplanar Graphs, IPL 110:970–974, 2010].

1 Introduction

A *cycle basis* of a graph G is a minimum-cardinality set $\mathcal{C}$ of cycles in G such that every cycle $C \in G$ can be written as the exclusive-or sum of a subset of cycles in $\mathcal{C}$. A *minimum cycle basis* (MCB) of G is a cycle basis that minimizes the total weight of cycles in the basis. Minimum cycle bases have numerous applications in the analysis of electrical networks, biochemistry, periodic timetabling, surface reconstruction, and public transportation, and have been intensively studied in the computer science literature, cf. [12] for an exhaustive survey. It is therefore—both from a practical and a theoretical viewpoint—an interesting task to compute them efficiently.

All graphs considered in this work are simple graphs $G = (V, E)$ with a non-negative edge-weight function $w : E \to \mathbb{R}_{\geq 0}$. (Computing MCBs for graphs with cycles of negative weight is an NP-hard problem [12]. In all previous work that we are aware of it is therefore assumed that the edge-weights are non-negative.)

[⋆] Supported by a Feodor Lynen postdoctoral research fellowship of the Alexander von Humboldt Foundation and by the Agence Nationale de la Recherche under the project ANR-09-JCJC-0067-01.

[⋆⋆] Work initiated while visiting the Max Planck Institute for Informatics during an internship.

1.1 Previous Work

The first polynomial time algorithm for computing MCBs was presented by Horton [11] in 1987. His algorithm has running time $O(m^3n)$. This was improved subsequently in a series of papers by different authors, cf. [12] or [14] for surveys of the history. The currently fastest algorithms for general graphs are a deterministic $O(m^2n/\log n)$ algorithm of Amaldi, Iuliano, and Rizzi [2] and a Monte Carlo based algorithm by Amaldi, Iuliano, Jurkiewicz, Mehlhorn, and Rizzi [1] of running time $O(m^\omega)$, where ω is the matrix multiplication constant.

The algorithm from [1] is deterministic on planar graphs, and has a running time of $O(n^2)$. This improved the previously best known bound by Hartvigsen and Mardon [10], which is of order $n^2 \log n$. The currently best known algorithm on planar graphs is due to Borradaile, Sankowski, and Wulff-Nilsen [6]. It constructs an $O(n \log n)$ space implicit representation of an MCB in planar graphs in time $O(n \log^5 n)$.

Faster algorithms for planar graphs are known only for the special case of outerplanar graphs. For unweighted outerplanar graphs, Leydold, and Stadler [13] presented a linear time algorithm. More recently, Liu and Lu [14] presented a linear time, linear space algorithm to compute an MCB of a weighted outerplanar graph (using an implicit representation). This is optimal in terms of both time and space.

1.2 Our Result

In this contribution, we present a linear time algorithm for computing an implicit $O(n)$-space representation of a minimum cycle basis in partial 2-trees (graphs of treewidth two). The explicit representation can be obtained in additional time that is proportional to the size of the MCB. Since partial 2-trees are planar graphs, the previously best known algorithm was the one by Borradaile, Sankowski, and Wulff-Nilsen. That is, for the special case of partial 2-trees we are able to improve their running time by a factor of $\Theta(\log^5 n)$.

The class of partial 2-trees subsumes, in particular, the class of outerplanar graphs. Our result is achieved by an iterative decomposition of the partial 2-tree into outerplanar graphs, to which the recent result of Liu and Lu [14] can be applied. We state our main theorem.

Theorem 1. *Given a partial* 2*-tree* G *on* n *vertices and a non-negative weight function* $w : E \to \mathbb{R}_{\geq 0}$*, a minimum cycle basis* B *of* G *(implicitly encoded in* $O(n)$ *space) can be obtained in* $O(n)$ *time.*

Moreover, B *can be reported explicitly in time* $O(\text{size}(B))$*, where* $\text{size}(B)$ *is the number of edges in* B *counted according to their multiplicity.*

Note in Theorem 1 that, although B has an *implicit* representation of linear size, the *explicit size* of B may be quadratic. This is true already for outerplanar graphs, cf. [14] for a simple grid graph G in which the unique MCB of G contains $\Theta(n^2)$ edges.

For the proof of Theorem 1, it will be crucial that the set of *lex short cycles* (cf. Section 2.3) in any weighted partial 2-tree forms a minimum cycle basis [15]. As lex short cycles are inherently defined by shortest paths, we will need a data structure that reports the distance between two vertices in constant time (e.g., for checking whether an edge is the shortest path between its endpoints). In outerplanar graphs, such a data structure exists due to Frederickson and Jannardan [9]. For our more general case, we will instead extend a result of Chaudhuri and Zaroliagis [7, Lemma 3.2] on weighted partial k-trees, which is able to give the distance of every two vertices in constant time, as long as both are contained in one bag of a tree decomposition. Using this extension, we can report the shortest path P between any two such vertices in time $O(|E(P)|)$.

2 Graph Preliminaries, Partial 2-Trees and Lex Shortest Paths

We consider weighted undirected graphs $G = (V, E)$ where V denotes the set of vertices, E the set of edges and $w : E \to \mathbb{R}_{\geq 0}$ a non-negative weight function. Throughout this work, we set $n = |V|$ and $m = |E|$. All graph classes considered in this paper are sparse, i.e., we have a linear dependence $m = O(n)$.

We use standard graph terminology from [8]. The weight $w(P)$ of a path P in G is the sum of weights of edges in P; i.e., $w(P) := \sum_{e \in P} w(e)$. For a graph H, an H-subdivision is a graph obtained from H by replacing its edges with non-empty and pairwise vertex-disjoint paths. In this work, we will be mainly concerned with $K_{2,3}$-subdivisions. In such a $K_{2,3}$-subdivision, we call the vertices of degree greater than two the *branch vertices* of the subdivision.

2.1 Minimum Cycle Bases

A cycle C in G is a connected subgraph of G in which every vertex has degree two. Let $C_1, \ldots, C_k$ be cycles in G and let $\oplus$ denote the symmetric difference function. Then the sum $\mathcal{S} := C_1 \oplus \ldots \oplus C_k$ is the set of edges appearing an odd number of times in the multi-set $\{C_1, \ldots, C_k\}$. It is well known that $\mathcal{S}$ is a union of cycles in G.

Let a set $\mathcal{C} = \{C_1, \ldots, C_k\}$ of cycles of G *span the cycle space of* G if every cycle C of G can be written as a sum $C_{i_1} \oplus \ldots \oplus C_{i_\ell}$ of elements of $\mathcal{C}$. In this case, we say that $C_{i_1}, \ldots, C_{i_\ell}$ *generate* C. The size size($\mathcal{C}$) of $\mathcal{C}$ is the number of edges in $\mathcal{C}$ counted according to their multiplicity.

A *cycle basis* of G is a minimum cardinality set of cycles that spans the cycle space of G. Put differently, a cycle basis is a maximal set of independent cycles, where we consider a set of cycles to be independent if their incidence vectors in $\{0, 1\}^m$ are independent over the field GF(2). The cardinality of a cycle basis is sometimes referred to as the *dimension* of the cycle space of G. For any simple weighted graph the dimension equals $m - n + 1$ [5].

We are interested in identifying a *minimum cycle basis* (MCB) of G; i.e., a cycle basis $\mathcal{C}$ of minimum total weight $\sum_{C \in \mathcal{C}} w(C)$.

A minimum cycle basis of a graph G is equal to the disjoint union of the minimum cycle basis of the 2-connected components of G. Therefore, throughout this paper, we assume without loss of generality that G is 2-connected.

2.2 Tree Decompositions and Partial 2-Trees

A *tree decomposition* of a graph G is a pair $(\{X_1, \dots, X_r\}, T)$ of a set of *bags* $X_1, \dots, X_r$ and a tree T with vertex set $V(T) = \{X_1, \dots, X_r\}$ that satisfies the following three properties:

1. $X_1 \cup \dots \cup X_r = V$,
2. For each edge $\{u, v\} \in E$, there is an index $1 \leq i \leq r$ such that $\{u, v\} \subseteq X_i$, and
3. For each vertex $v \in V$, the bags in T containing v form a subtree of T (*subtree property*).

The *treewidth* of $(\{X_1, \dots X_r\}, T)$ is $\max\{|X_1|, \dots |X_r|\} - 1$. The *treewidth* of G is the minimum treewidth over all possible tree decompositions of G. We call a tree decomposition $(\{X_1, \dots X_r\}, T)$ *optimal* if the treewidth of T is equal to the treewidth of G. To distinguish between the edges of G and T, we refer to the edges of T as *links*.

A *k-tree* is a graph of treewidth k for which the addition of any edge between non-adjacent vertices would increase the treewidth.

Any subgraph of a k-tree is called a *partial k-tree*. Partial 2-trees are also known as graphs in which every biconnected component is a *series-parallel* graph [4]. The partial 2-trees form a strict superclass of outerplanar graphs, as outerplanar graphs are characterized by the forbidden minor set $\{K_4, K_{2,3}\}$, while partial 2-trees have the forbidden minor set $\{K_4\}$. Equivalently, a partial 2-tree is outerplanar if and only if it does not contain a $K_{2,3}$-subdivision (as a subgraph). We will need the following somewhat stronger statement.

Lemma 2 (Lemma 2.4 in [15]). *Let G be a weighted partial 2-tree. G is not outerplanar if and only G contains a $K_{2,3}$-subdivision with branch vertices u and v such that $G - \{u, v\}$ has at least 3 connected components.*

2.3 Lex Shortest Paths and Lex Short Cycles

It is known (Proposition 4.5 in [10]) that for any edge-weighted simple graph G the set of so-called *lex short cycles* contains a minimum cycle basis. For outerplanar graphs [14] and partial 2-trees [15], the whole set of lex short cycles forms a minimum cycle basis.

Definition 3 (Lex Shortest Paths). *Let $G = (V, E)$ be a graph with weight function $w : E \to \mathbb{R}_{\geq 0}$. Let $\sigma : V \to \{1, 2, \dots, n\}$ be an arbitrary ordering of the vertices.*

A path P between two distinct vertices $u, v \in V$ is called a lex shortest path *if for any other path P' between u and v either $w(P') > w(P)$ or $(w(P') = w(P)$ and $|E(P')| > |E(P)|)$ or $(w(P') = w(P)$, $|E(P')| = |E(P)|$ and $\min_{y \in V(P') \setminus V(P)} \sigma(y) > \min_{y \in V(P) \setminus V(P')} \sigma(y))$ holds.*

It is easily verified (cf. Proposition 4.1 in [10]) that between any two vertices u, v in G there is exactly one lex shortest path. We refer to this path as $\mathrm{lsp}(u, v)$. If the dependence of the graph is not clear from the context, we write $\mathrm{lsp}_G(u, v)$. Note that every subpath of a lex shortest path is a lex shortest path.

Definition 4 (Lex Short Cycles). *A* lex short cycle *C is a cycle that contains for any two vertices $u, v \in C$ the lex shortest path* $\mathrm{lsp}(u, v)$*. For an edge-weighted graph G, we denote by* $\mathrm{LSC}(G)$ *the set of all lex short cycles in G.*

Lemma 5 ([10,15]). *For any edge-weighted simple graph G, there is a set $B \subseteq \mathrm{LSC}(G)$ such that B is a minimum cycle basis for G. Additionally, the set of lex short cycles $\mathrm{LSC}(G)$ forms a minimum cycle basis if G is a weighted partial 2-tree.*

Abusing notation (since $\mathrm{MCB}(G)$ may not be unique) we write $\mathrm{MCB}(G) \subseteq \mathrm{LSC}(G)$ and $\mathrm{MCB}(G) = \mathrm{LSC}(G)$, respectively, for the two statements in Lemma 5. This lemma allows us to restrict ourselves to compute the set of lex short cycles. For outerplanar graphs, Liu and Lu showed that an implicit representation of $\mathrm{LSC}(G)$ can be computed in linear time.

Theorem 6 ([14]). *For any weighted outerplanar graph G an $O(n)$-space representation of the set $\mathrm{LSC}(G)$ can be computed in $O(n)$ time. From this representation, we can compute a cycle $C \in \mathrm{LSC}(G)$ explicitly in time $O(\mathrm{size}(C))$.*

3 High-Level Overview of Our Algorithm and Technical Details

We first describe the high-level idea of our algorithm; the proofs and all details are presented in the subsequent sections. From now on we assume that G is a weighted partial 2-tree.

3.1 Preprocessing Steps

We introduce a few preprocessing steps besides 2-connectivity that will simplify the description of our algorithm. As a first step, we construct an alternative to the weight function w for which every lex shortest path is the unique shortest path. That this can be done in linear time has been shown in [10].

Lemma 7 (Proposition 4.3 in [10]). *If G is a simple graph with edge weight function w, then there exists a perturbation $\hat{w}$ of w such that every lex shortest path $\mathrm{lsp}(u, v)$ under w is the unique shortest $u - v$ path under $\hat{w}$.*

We call an edge $\{u, v\}$ *tight* if it is the unique shortest path between u and v, and we call it *long* otherwise. The long edges in G will be treated separately. This is based on the following lemma.

Lemma 8. *Let $G = (V, E)$ be an edge-weighted graph. Let L be the set of long edges in G. Then $\mathrm{MCB}(G) = \mathrm{MCB}(G \setminus L) \cup \{\{e\} \cup \mathrm{lsp}(u, v) \mid e = \{u, v\} \in L\}$.*

Proof. For every $e = \{u, v\} \in L$ the cycle $\{e\} \cup \mathrm{lsp}(u, v)$ is a lex short cycle in G. By definition, it is the only lex short cycle that contains e. Therefore, $\mathrm{LSC}(G) = \mathrm{LSC}(G \setminus L) \cup \{e' \cup \mathrm{lsp}(u', v') \mid e' = \{u', v'\} \in L\}$. By Lemma 5 we know that $\mathrm{MCB}(G) \subseteq \mathrm{LSC}(G)$. The independence of $\mathcal{M} := \mathrm{MCB}(G \setminus L) \cup \{e' \cup \mathrm{lsp}(u', v') \mid e' = \{u', v'\} \in L\}$ follows again from the fact that e is contained in $\mathcal{M}$ only in this one cycle $\{e\} \cup \mathrm{lsp}(u, v)$. $\square$

As we shall see in Lemma 15 below, the set L in Lemma 8 can be identified and removed from G in $O(n)$ time using a suitable data structure. Using these two lemmata, computing the contribution of the removed long edges to the MCB of G takes at most linear time.

3.2 High-Level Overview of the Main Algorithm

By Lemma 5 it suffices to compute the set of lex short cycles in G. This set forms a minimum cycle basis. For simplicity, we assume that the simplifications described above have been performed in a preprocessing step. In particular, all shortest paths in G are unique and G contains only tight edges. As shown above, the preprocessing steps can be done in time $O(n)$.

The key approach for our algorithm is to iteratively decompose the graph G into outerplanar subgraphs $G_1, \ldots, G_r$. To these subgraphs we apply the linear time algorithm of Liu and Lu (Theorem 6). Intuitively, the decomposition is done as follows.

When G is not outerplanar, then there exists a $K_{2,3}$-subdivision in G with branch vertices u and v such that (i) $\{u, v\}$ is a minimum vertex separator of G and (ii) the removal of $\{u, v\}$ disconnects G into at least three connected components $H_1, \ldots, H_k$ (cf. Lemma 2). We distinguish two cases. If $\{u, v\} \in E$, we set $G_i := G[V(H_i) \cup \{u\} \cup \{v\}]$. Otherwise, let $j \in \{1, 2, \ldots, k\}$ such that $\mathrm{lsp}(u, v) \in G[V(H_j) \cup \{u\} \cup \{v\}]$. We set $G_j := G[V(H_j) \cup \{u\} \cup \{v\}]$, and for all $1 \leq i \neq j \leq k$ we set $G_i := G[V(H_i) \cup \{u\} \cup \{v\}] \cup \mathrm{blue}(\{u, v\})$, where $\mathrm{blue}(\{u, v\})$ denotes a new "colored" (i.e., marked) edge $\{u, v\}$. This *blue* edge serves as a placeholder for the lex shortest path between u and v, as this is not contained in G_i. The weight $w(\mathrm{blue}(\{u, v\}))$ assigned to this new edge is therefore $w(\mathrm{lsp}(u, v))$. Let the operation $\mathrm{decomp}(G, u, v)$ decompose G into $G_1, \ldots, G_k$ with respect to the vertices u, v.

We now iteratively decompose the graphs $G_1, \ldots, G_k$ as described above until we are left with graphs $G_1, \ldots, G_r$ that do not contain any $K_{2,3}$-subdivision, i.e., with outerplanar graphs according to Lemma 2. Since all the edges in G_i are tight, the set of lex short cycles in G_i equals the boundaries of its internal faces. Extracting the internal faces of G_i can be done in linear time, cf. [14]. We will show in Lemma 17 that $\mathrm{LSC}(G)$ equals the disjoint union of $\mathrm{expand}(\mathrm{LSC}(G_1)), \ldots, \mathrm{expand}(\mathrm{LSC}(G_r))$, where $\mathrm{expand}(\mathrm{LSC}(G_i))$ replaces the blue edges $\mathrm{blue}(\{u, v\})$ in every cycle by the lex shortest path $\mathrm{lsp}(u, v)$.

The challenging part of the algorithm is to find a data structure that allows to identify all the $K_{2,3}$-subdivisions and to do the respective decompositions in linear time. To this end, we define *suitable tree decompositions.*

3.3 Suitable Tree Decompositions

We define suitable tree decompositions and we show how they help in efficiently computing our decomposition. Let the *label* of a link $\ell = (X, Y)$ in a tree decomposition be $X \cap Y$.

Definition 9 (Suitable Tree Decomposition). *An optimal rooted tree decomposition of G is* suitable *if it satisfies the following properties:*

1. *The size of every bag X_i is 3 and every two adjacent bags X_i, X_j in T differ by exactly one vertex; i.e., $|X_i \cap X_j| = 2$ (this property is called* smooth *in [3]).*
2. *Any two links with the same label have a common parent in T.*

Observe that for any internal bag in T, the number of children could be arbitrary, but there are at most three different labels associated with the links to its children.

Our algorithm will perform all computations in a suitable tree decomposition of the tight induced subgraph of G. It is therefore important that such a tree decomposition can be computed in linear time.

Lemma 10. *Given a weighted partial 2-tree G, a suitable tree decomposition can be computed in linear time and has linear space.*

One of the underlying key observations of our algorithm is the fact that for all $K_{2,3}$-subdivisions the two branch vertices must be contained in a common bag of a suitable tree decomposition. This is shown using the following result.

Lemma 11 (Lemma 12.3.4 in [8]). *Let $W \subseteq V(G)$ and T be a tree decomposition of G. Then T contains either a bag that contains W or a link $\{X_1, X_2\}$ such that two vertices of W are separated by $X_1 \cap X_2$ in G.*

Lemma 12 ($K_{2,3}$-subdivisions in partial 2-trees). *For every $K_{2,3}$-subdivision H in a partial 2-tree G and every smooth tree decomposition T of G, the two branch vertices u and v of H are contained in a common bag of T.*

Proof. We apply Lemma 11 with $W = \{u, v\}$. If u and v are not contained in a common bag, there must be a link $\{X_1, X_2\}$ such that u and v are separated by $X_1 \cap X_2$ in G. Since H is a $K_{2,3}$-subdivision, $|X_1 \cap X_2| \geq 3$, contradicting the smoothness of T. □

3.4 Suitable Data Structures for Finding the Lex Shortest Paths

Another useful tool in our algorithm will be the following data structure. It supports the query for an intermediate vertex that lies on the lex shortest path between two nodes. The following lemma can be proven along the lines of [9].

Lemma 13. *Let T be a rooted tree decomposition of G. There is a linear space data-structure with $O(n)$ preprocessing time that supports the following query: Given a bag $A \in T$ and a vertex $v \in G$ that is not in A, find the link incident to A that leads to the subtree of T containing a bag with vertex v. The query time is $O(\log d)$, where d is the degree of A in T.*

We are finally ready to show that for any long edge $\{u, v\}$ the lex shortest path between u and v can be computed in time $O(|E(\text{lsp}(u, v))|)$. The following lemmata will be useful also to identify the subtree of the tree decomposition that contains $\text{lsp}(u, v)$ for two branch vertices u and v with $\{u, v\} \notin E$.

Lemma 14 (Lemma 3.2 in [7]). *Given a partial k-tree G and an optimal tree decomposition T of G, there is an algorithm with running time $O(k^3 n)$ that outputs the distances of all vertex pairs that are contained in common bags and that, for each such vertex pair, outputs some intermediate vertex of the shortest path between the vertices.*

Lemma 14 is originally stated for directed graphs in [7]. However, representing each undirected edge with two edges oriented in opposite directions gives the above undirected variant.

We extend Lemma 14 by giving the following data structure.

Lemma 15. *Given a connected partial 2-tree G and a suitable tree decomposition T of G, there is an $O(n)$-space data structure requiring $O(n)$ preprocessing time that supports the following queries, given two vertices u and v and a bag $X \in T$ that contains u and v:*

- *Compute in time $O(1)$ the distance of the shortest path between u and v (*distance query*).*
- *Compute in time $O(1)$ an intermediate vertex w of the shortest path between u or v (if exists) and a bag $Y \in T$ such that $Y = \{u, v, w\}$ (*intermediate vertex query*)*
- *Compute in time $O(|E(P)|)$ the shortest path P between u and v (*shortest path extraction*)*

Proof. We perform the algorithm of Lemma 14 and store the distance of every vertex pair $\{u, v\}$ that is contained in a common bag, say in X, in a table linked to X. Since T contains only linearly many bags, this takes $O(n)$ space. The table supports distance queries in constant time, as there are only constantly many vertex pairs in each bag.

Assume for the moment that we know how to support the intermediate vertex query. Then we can easily support the shortest path extraction by first applying an intermediate vertex query, which gives Y, and subsequently recursing on the two intermediate vertex queries $\{u, w\}$ and $\{w, v\}$, both in Y, until each shortest path is just an edge. This allows to extract the shortest path u and v in time proportional to its length.

It remains to show how to support intermediate vertex queries. We initialize the data structure D of Lemma 13 for the tree decomposition T in time $O(n)$

and apply the algorithm of Lemma 14 in time $O(n)$. Let X be a bag containing u and v. By Lemma 14, we have already found an intermediate vertex z between u and v, but want to find an intermediate vertex w that is in a common bag Y with u and v. If z does not exist, the shortest path is just an edge, in which case we just set w to be non-existent as well. If $z \in X$, we set $w = z$ and $Y = X$ and are done.

Otherwise, we query D with (X, z) and get a link (X, A) such that z is contained in the subtree of T that is separated by (X, A) and does not contain X (note that A may be the parent of X in T). According to Lemma 13, this query takes time proportional to at most the degree of X in T.

We now distinguish two cases. In the case that A contains u and v, we iterate this procedure further on A instead on X. In this iteration, this case cannot happen more than a constant number of times, as T is suitable, so any path in the subtree of T consisting of bags containing $\{u, v\}$ has length at most 2.

Otherwise, A contains exactly one vertex of $\{u, v\}$, say u. Consider $X = \{u, v, r\}$ and the subtree T_1 of T that is separated by the link (X, A) and contains A. By the subtree property, T_1 cannot contain a bag with v, as then v would also be contained in A. Since T_1 contains a part of the shortest path between u and v, but has only u and r in common with X, r must be an intermediate vertex. Since X contains u, v, and r, we set $w = r$ and $Y = X$.

We investigate the preprocessing time of the data structure, i.e., the time spent computing for all vertex pairs (u, v) the intermediate vertex w and the bag containing all three vertices $\{u, v, w\}$. In every bag X, there are only constantly many vertex pairs. For each such vertex pair, we could find w in time $O(\deg(X))$, where $\deg(X)$ is the degree of X in T. Hence, the preprocessing time sums up to a linear total. $\square$

Since a tree decomposition maintains for every edge the bag that contains it, the query of Lemma 15 can also be performed when no bag is given but an edge $\{u, v\}$.

3.5 Obtaining LSC(G) from LSC(G_1), . . . , LSC(G_r)

As a last technicality, we show that—as claimed in the high-level overview of our algorithm—the set of lex short cycles in G equals the disjoint union $\text{expand}(\text{LSC}(G_1)) \uplus \ldots \uplus \text{expand}(\text{LSC}(G_r))$. This follows from iteratively applying Lemma 17 below.

Definition 16. *For any cycle C of G, let* $\text{expand}(C)$ *be the cycle obtained from C by replacing the blue edges* $\text{blue}(\{u, v\})$ *in C (if exist) by the lex shortest path* $\text{lsp}(u, v)$. *For a set of cycles $\mathcal{C}$, let* $\text{expand}(\mathcal{C}) := \{\text{expand}(C) \mid C \in \mathcal{C}\}$.

As mentioned above, we obtain a minimum cycle basis of G by expanding the cycles in the MCBs of the subgraphs.

Lemma 17. *Let G be a weighted partial 2-tree in which every edge is tight. Let u and v be the two branch vertices of a $K_{2,3}$-subdivision in G. Let $k \geq 3$ and let*

Algorithm 1. A linear time algorithm to compute the implicit representation of an MCB of a 2-connected weighted partial 2-tree G

```
1  Do the graph preprocessing steps described in Section 3.1;
2  Compute a suitable tree decomposition of $G$;
3  for each internal bag $Y_1 \in V(T)$ and every $u, v \in Y_1$ do
4      Let $Y_2, \dots Y_k$ be the children of $Y_1$ such that for
       $2 \leq i \leq k, Y_1 \cap Y_i = \{u, v\}$;
5      if $k \geq 3$ then
6          for $2 \leq i \leq k$ do remove the link $\{Y_1, Y_i\}$;
7          if $\{u, v\} \notin E$ then
8              Find an intermediate vertex $y$ in $\mathrm{lsp}(u, v)$ and compute
               $w(\mathrm{lsp}(u, v))$ ;
9              if there exists a $j \in \{2, \dots, k\}$ such that the subtree rooted at
               $Y_j$ has a bag that contains the vertex $y$ then let $j$ be that
               index; else $j = 1$;
10             for $1 \leq i \neq j \leq k$ do
11                 Add the new blue edge $\mathrm{blue}(\{u, v\})$ to $Y_i$ and assign to it
                   weight $w(\mathrm{lsp}(u, v))$;
12 Let $T_1, \dots, T_r$ be the connected components of $T$;
13 Obtain the graphs $G_1, \dots, G_r$ that correspond to the tree decompositions
   $T_1, \dots, T_r$, respectively;
14 Compute $\mathrm{LSC}(G_1), \dots, \mathrm{LSC}(G_r)$ using [14] ; // $\mathrm{LSC}(G_i)$ equals the
   internal faces of $G_i$
15 Store $(L, \mathrm{LSC}(G_1), \dots, \mathrm{LSC}(G_r))$ ; // $L$ is the set of long edges
```

$G_1, \dots, G_k$ be the subgraphs resulting from the decomposition $\mathrm{decomp}(G, u, v)$. *Then* $\mathrm{LSC}(G) = \mathrm{expand}(\mathrm{LSC}(G_1)) \uplus \dots \uplus \mathrm{expand}(\mathrm{LSC}(G_k))$.

Lemma 17 can be proven using the following observation.

Lemma 18 (Lemma 2.5 and Corollary 2.8 in [15]). *Let G be a weighted graph and let G' be a subgraph of G. Let P be a path in G'. If P is lex shortest in G, it is lex shortest in G'.*

Furthermore, for G, k, and $G_1, \dots, G_k$ as in Lemma 17, we have $\mathrm{expand}(\mathrm{LSC}(G_i)) \subseteq \mathrm{LSC}(G)$, $1 \leq i \leq k$.

4 Computing an MCB in Weighted Partial 2-Trees

Our algorithm, Algorithm 1, can now be described as follows.

Preprocessing. Given a 2-connected weighted partial 2-tree $G = (V, E)$ with edge-weight function $\hat{w}$, we first compute the perturbation w of $\hat{w}$ such that every lex shortest path is the unique shortest path (cf. Lemma 7). We then identify the set L of long edges; i.e., the set of edges that are not the shortest paths between their two endpoints (cf. Lemma 8), and, for the moment, we remove these edges from G.

For the graph obtained from this preprocessing step, we compute a suitable tree decomposition.

Main Procedure. In the main loop of Algorithm 1, we check for every vertex pair $\{u, v\}$ in every bag if $\{u, v\}$ is a vertex separator that decomposes G into at least three different components. By the Lemmata 2 and 12 and the fact that G is assumed to be 2-connected, this identifies a $K_{2,3}$-subdivision if it exists. If so, we decompose along u and v by deleting the appropriate links in T in line 6. Lines 8 and 9 are needed to identify the subtree of T that contains $\mathrm{lsp}(u, v)$. In all other subtrees, the edge $\{u, v\}$ is marked in blue, and the weight associated to this edge is $w(\mathrm{lsp}(u, v))$. By maintaining the data structure of Lemma 15, the intermediate vertex can be found efficiently.

When Algorithm 1 stops, we compute the connected components $T_1, \dots, T_r$ of T in linear time by performing a standard graph traversal routine. We can compute the graphs $G_1, \dots, G_r$ that are represented by these tree decompositions in linear total time by just collecting the vertices and edges in all bags. Note that the total number of edges in $G_1, \dots, G_r$ is linear, as we add at most $\deg(V_1)$ new blue edges for each bag V_1, where $\deg(V_1)$ is the degree of bag V_1 in T. Every G_i is outerplanar and the LSCs of all these outerplanar graphs can be computed in linear total time. (As mentioned in the high-level overview, one can simply apply the result of Liu and Lu (Theorem 6) as a black-box. Or one observes that since all edges in G_i are tight, the set of lex short cycles in G_i equals the boundaries of its internal faces. Extracting the internal faces of G_i can be done in linear time.)

The set L of long edges together with (the implicit representations) of $\mathrm{LSC}(G_1), \dots, \mathrm{LSC}(G_r)$ forms the implicit representation of our minimum cycle basis.

This concludes the presentation of our main result, the first part of Theorem 1. In order to get the explicit representation of the minimum cycle basis, we apply Lemma 8. It thus suffices to augment every edge (u, v) in the set L of long edges of the implicit representation by $lsp(u, v)$, which can be done in time $O(|C|)$ for each constructed cycle C, according to Lemma 15.

5 Discussion

We have shown that an implicit representation of a minimum cycle basis of a weighted partial 2-tree can be computed in linear time. It remains a challenging question if our result can be extended to partial k-trees for $k > 2$. We remark that it was noted in [15] that already for partial 3-trees the set of lex short cycles do not necessarily form a minimum cycle basis. Extending our result to partial 3-trees may therefore require substantially new ideas.

Acknowledgments. We would like to thank Geevarghese Philip for pointing us to Lemma 11, which greatly simplified our proof of the runtime bound.

G. Ramakrishna would like to thank his adviser N.S. Narayanaswamy for fruitful discussions during the early stages of this work. He would also like to thank Kurt Mehlhorn for providing him the opportunity to do an internship at the Max Planck Institute for Informatics.

References

1. Amaldi, E., Iuliano, C., Jurkiewicz, T., Mehlhorn, K., Rizzi, R.: Breaking the $o(m^2\ n)$ barrier for minimum cycle bases. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 301–312. Springer, Heidelberg (2009)
2. Amaldi, E., Iuliano, C., Rizzi, R.: Efficient deterministic algorithms for finding a minimum cycle basis in undirected graphs. In: Eisenbrand, F., Shepherd, F.B. (eds.) IPCO 2010. LNCS, vol. 6080, pp. 397–410. Springer, Heidelberg (2010)
3. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM Journal of Computing 25(6), 1305–1317 (1996)
4. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theoretical Computer Science 209(1-2), 1–45 (1998)
5. Bondy, J.A., Murty, U.S.R.: Graph Theory. Springer (2008)
6. Borradaile, G., Sankowski, P., Wulff-Nilsen, C.: Min st-cut oracle for planar graphs with near-linear preprocessing time. In: Proc. of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010), pp. 601–610. IEEE Computer Society (2010)
7. Chaudhuri, S., Zaroliagis, C.D.: Shortest paths in digraphs of small treewidth. Part I: Sequential algorithms. Algorithmica 27(3), 212–226 (2000)
8. Diestel, R.: Graph Theory, 4th edn. Springer (2010)
9. Frederickson, G.N., Janardan, R.: Designing networks with compact routing tables. Algorithmica 3, 171–190 (1988)
10. Hartvigsen, D., Mardon, R.: The all-pairs min cut problem and the minimum cycle basis problem on planar graphs. SIAM Journal on Discrete Mathematics 7, 403–418 (1994)
11. Horton, J.D.: A polynomial-time algorithm to find the shortest cycle basis of a graph. SIAM Journal of Computing 16, 358–366 (1987)
12. Kavitha, T., Liebchen, C., Mehlhorn, K., Michail, D., Rizzi, R., Ueckerdt, T., Zweig, K.A.: Cycle bases in graphs characterization, algorithms, complexity, and applications. Computer Science Review 3(4), 199–243 (2009)
13. Leydold, J., Stadler, P.F.: Minimal cycle bases of outerplanar graphs. Electronic Journal of Combinatorics 5 (1998)
14. Liu, T., Lu, H.: Minimum cycle bases of weighted outerplanar graphs. Information Processing Letters 110, 970–974 (2010); A preliminary report appeared in Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 564–572. Springer, Heidelberg (2009)
15. Narayanaswamy, N.S., Ramakrishna, G.: Characterization of minimum cycle bases in weighted partial 2-trees. In: Proc. of the 11th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW 2012), pp. 193–196 (2012), `http://arxiv.org/abs/1302.5889` All references in this paper refer to this arXiv version

Thickness and Colorability of Geometric Graphs

Stephane Durocher[1,⋆], Ellen Gethner[2], and Debajyoti Mondal[1,⋆⋆]

[1] Department of Computer Science, University of Manitoba
[2] Department of Computer Science, University of Colorado Denver
{durocher,jyoti}@cs.umanitoba.ca, ellen.gethner@ucdenver.edu

Abstract. The geometric thickness $\overline{\theta}(G)$ of a graph G is the smallest integer t such that there exist a straight-line drawing Γ of G and a partition of its straight-line edges into t subsets, where each subset induces a planar drawing in Γ. Over a decade ago, Hutchinson, Shermer, and Vince proved that any n-vertex graph with geometric thickness two can have at most $6n-18$ edges, and for every $n \geq 8$ they constructed a geometric thickness two graph with $6n-20$ edges. In this paper, we construct geometric thickness two graphs with $6n-19$ edges for every $n \geq 9$, which improves the previously known $6n-20$ lower bound. We then construct a thickness two graph with 10 vertices that has geometric thickness three, and prove that the problem of recognizing geometric thickness two graphs is NP-hard, answering two questions posed by Dillencourt, Eppstein and Hirschberg. Finally, we prove the NP-hardness of coloring graphs of geometric thickness t with $4t-1$ colors, which strengthens a result of McGrae and Zito, when $t=2$.

1 Introduction

The *thickness* $\theta(G)$ of a graph G is the smallest integer t such that the edges of G can be partitioned into t subsets, where each subset induces a planar graph. Since 1963, when Tutte [21] first formally introduced the notion of graph thickness, this property of graphs has been extensively studied for its interest from both the theoretical [2,5,7] and practical point of view [17,19]. A wide range of applications, e.g., circuit layout design, simultaneous geometric embedding, and network visualization, have motivated the examination of thickness in the geometric setting [7,11,12,14]. The *geometric thickness* $\overline{\theta}(G)$ of a graph G is the smallest integer t such that there exist a *straight-line drawing* (i.e., a drawing on the Euclidean plane, where every vertex is drawn as a point and every edge is drawn as a straight line segment) Γ of G and a partition of its straight-line edges into t subsets, where each subset induces a planar drawing in Γ. If $t=2$, then G is called a *geometric thickness two graph* (or, a doubly-linear graph [14]), and Γ is called a *geometric thickness two representation* of G. While graph theoretical

[⋆] Work of the author is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

[⋆⋆] Work of the author is supported in part by a University of Manitoba Graduate Fellowship.

thickness does not impose any restriction on the placement of the vertices in each planar layer, the geometric thickness forces the same vertices in different planar layers to share a fixed point in the plane. Eppstein [11] clearly established this difference by constructing thickness three graphs that have arbitrarily large geometric thickness.

Structural Properties. Geometric thickness has been broadly examined on several classes of graphs, e.g., complete graphs [7], bounded-degree graphs [4,10,11], and graphs with bounded treewidth [8,9]. Hutchinson, Shermer, and Vince [14] examined properties of graphs with geometric thickness two. They proved that these graphs at most $6n-18$ edges, and for every $n \geq 8$ they constructed a geometric thickness two graph with $6n-20$ edges. Even after several attempts [7,10] to understand the structural properties of geometric thickness two graphs, the question whether there exists a geometric thickness two graph with $6n-18$ edges remained open for over a decade. Answering this question is quite challenging since although one can generate many thickness two graphs with $6n-18$ or $6n-19$ edges, no efficient algorithm is known that can determine the geometric thickness of such a graph. However, by examining the point configurations that are likely to support geometric thickness two graphs with large numbers of edges, we have been able to construct geometric thickness two graphs with $6n-19$ edges (see Section 2).

Recognition. Although graph theoretical thickness is known for all complete graphs [2] and complete bipartite graphs [5], geometric thickness for these graph classes is not completely characterized. Dillencourt, Eppstein and Hirschberg [7] proved an $\lceil n/4 \rceil$ upper bound on the geometric thickness of K_n, giving a nice construction for representations with geometric thickness $t = \lceil n/4 \rceil$. They also gave a lower bound on the geometric thickness of complete graphs that matches the upper bound for several smaller values of n. Their bounds show that the geometric thickness of K_{15} is greater than its graph theoretical thickness, i.e., $\overline{\theta}(K_{15}) = 4 > \theta(K_{15}) = 3$, which settles the conjecture of [16] on the relation between geometric and graph theoretical thickness. Since the exact values of $\overline{\theta}(K_{13})$ and $\overline{\theta}(K_{14})$ are still unknown, Dillencourt et al. [7] hoped that the relation $\overline{\theta}(G) > \theta(G)$ could be established with a graph of smaller cardinality. In Section 3 we prove that the smallest such graph contains 10 vertices.

Since determining the thickness of an arbitrary graph is NP-hard [17], Dillencourt et al. [7] suspected that determining geometric thickness might be also NP-hard, and mentioned it as an open problem. The hardness proof of Mansfield [17] relies heavily on the fact that $\theta(K_{6,8}) = 2$. Dillencourt et al. [7] mentioned that this proof cannot be immediately adapted to prove the hardness of the problem of recognizing geometric thickness two graphs by showing that $\overline{\theta}(K_{6,8}) = 3$. This complexity question has been repeated several times in the literature [8,11] since 2000, and also appeared as one of the selected open questions in the 11th International Symposium on Graph Drawing (GD 2003) [6]. In Section 4 we settle the question by proving the problem to be NP-hard.

Colorability. As a natural generalization of the well-known Four Color Theorem for planar graphs [3], a long-standing open problem in graph theory is to determine the relation between thickness and colorability [15,20]. For every $t \geq 3$, the best known lower bound on the chromatic number of the graphs with thickness t is $6t - 2$, which can be achieved by the largest complete graph of thickness t. On the other hand, every graph with thickness t is $6t$ colorable [15]. Recently, McGrae and Zito [18] examined a variation of this problem that given a planar graph and a partition of its vertices into subsets of at most r vertices, asks to assign a color (from a set of s colors) to each subset such that two adjacent vertices in different subsets receive different colors. They proved that the problem is NP-complete when $r = 2$ (respectively, $r > 2$) and $s \leq 6$ (respectively, $s \leq 6r - 4$) colors. In Section 5 we prove the NP-hardness of coloring geometric thickness t graphs with $4t - 1$ colors. As a corollary, we strengthen the result of McGrae and Zito [18] that coloring thickness $t = r = 2$ graphs with 6 colors is NP-hard. Our hardness result is particularly interesting since no geometric thickness t graph with chromatic number more than $4t$ is known.

2 Geometric Thickness Two Graphs with $6n - 19$ Edges

Let K_9' be the graph obtained by deleting an edge from K_9. In this section we first construct a geometric thickness two representation Γ of K_9' that has $6n-19$ edges. We then show how to add vertices in Γ such that for any $n \geq 9$ one can construct a geometric thickness two graph with $6n - 19$ edges.

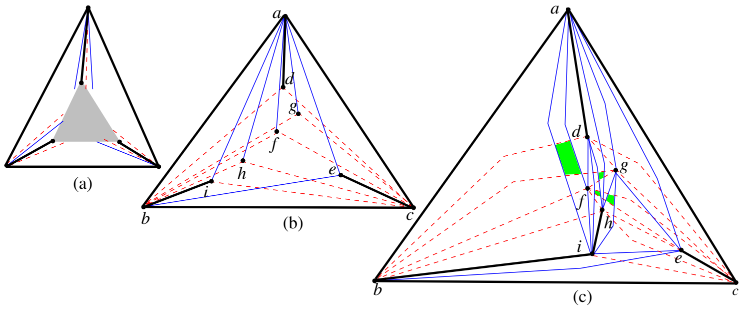


Fig. 1. (a) Illustration for the shared edges (bold). (b) Initial point set. (c) A geometric thickness two representation Γ of K_9', where the planar layers are shown in red (dashed) and blue (thin). Black edges can belong to either red or blue layer. Free quadrangles are shown in green (shaded). Some edges are drawn with bends for clarity.

Hutchinson et al. [14, Theorem 6] proved that if any geometric thickness two graph with $6n - 18$ edges exists, then the convex hull of its geometric thickness two representation must be a triangle. This representation is equivalent to the union of two plane triangulations that share at least six common edges, i.e., the three outer edges and the other three edges are adjacent to the three outervertices, as shown in black in Figure 1(a). Since each triangulation contains

$3n-6$ edges, the upper bound of $2(3n-6)-6=6n-18$ follows. These properties of an edge maximal geometric thickness two representation motivated us to examine pairs of triangulations that create many edge crossings while drawn simultaneously. In particular, we first created a set of points interior to the convex hull such that addition of straight line segments from each interior point to the three points on the convex hull creates two plane drawings that, while drawn simultaneously, contain a crossing in all but the six common edges. Figure 1(b) illustrates such a scenario. We then tried to extend each of these two planar drawings to a triangulation by adding new edges such that every new edge crosses at least one initial edge. We found multiple distinct point sets for which all but one newly added edge cross at least one initial edge, resulting in multiple distinct geometric thickness two representations with $2(3n-6)-7=6n-19$ edges. For example, see Figure 1(c), where the underlying graph is K_9'.

Let Γ be a geometric thickness two representation. A triangle in Γ is *empty* if it contains exactly three vertices on its boundary, but does not contain any vertex in its proper interior, e.g., the triangle Δghi in Figure 1(d). A *quadrangle in Γ is free* if it is created by the intersection of two empty triangles but does not contain any vertices of Γ, as shown in Figure 1(d) in green.

Theorem 1. *For each $n \geq 9$, there exists a geometric thickness two graph with n vertices and $6n-19$ edges that contains K_9 minus an edge as a subgraph. For each $n \geq 11$, there exists a geometric thickness two graph with $6n-19$ edges that does not contain K_8.*

Proof. Since K_9' has a geometric thickness two representation, as shown in Figure 1(c), the claim holds for $n=9$. We now claim that given an n-vertex geometric thickness two representation with $6n-19$ edges that contains a free quadrangle, one can construct a geometric thickness two representation with $n+1$ vertices and $6(n+1)-19$ edges. One can verify this claim as follows. Place a new vertex p interior to the free quadrangle. Since a free quadrangle is the intersection of two empty triangles, one can add three straight line edges from p to the three vertices of each empty triangle such that the new drawing in each layer remains planar, as shown in Figure 2(a). Since the number of vertices increases by one, and the number of edges increases by six, the resulting geometric thickness two representation must have $6n-19+6=6(n+1)-19$ edges.

Observe that there are at least four free quadrangles in the geometric thickness two representation of K_9', as shown in Figure 1(d). Therefore, for each $i, 9 \leq i \leq 12$, we can construct a geometric thickness two representation Γ_i with i vertices and $6i-19$ edges that contain at least one free quadrangle. We use these four geometric thickness two representations as the base case, and assume inductively that for each $9 \leq i < n$ there exists a geometric thickness two representation Γ_i with i vertices and $6i-19$ edges that contains at least one free quadrangle. We now construct a geometric thickness two representation with n vertices and $6n-19$ edges that contains a free quadrangle. By induction, Γ_{n-4} has a free quadrangle. We add four vertices to this quadrangle and complete the triangulation in each planar layer by adding 24 new edges in total, as shown in Figure 2(b). Consequently, the new geometric thickness two representation Γ_n

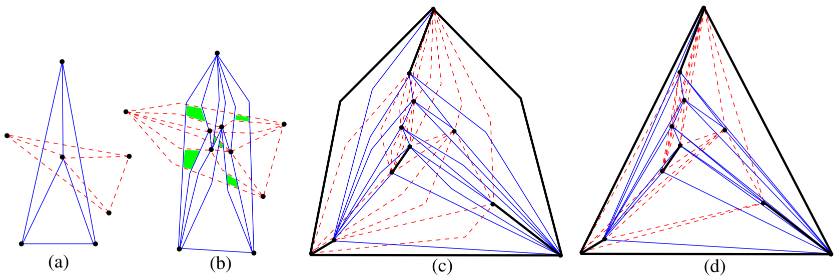


Fig. 2. (a)–(b) Adding vertices to a geometric thickness two drawing. (c)–(d) A graph with 11 vertices, 47 edges and geometric thickness two that does not contain K_8.

contains $6(n-4) - 19 + 24 = 6n - 19$ edges. Since the newly added edges create new free quadrangles, Γ_n also contains a free quadrangle.

The existence of such graphs that do not contain K_8 is proved using the graph illustrated in Figure 2(c). The details are omitted due to space constraints. □

3 Thickness Two Graphs with $\overline{\theta}(G) \geq 3$

We enumerate all possible geometric thickness two drawings of K_9' using Aichholzer et al.'s [1] point-set order-type database. Figure 3 illustrates all the three different configurations of nine points that support geometric thickness two drawings of K_9'. It might initially appear that Figures 3(a) and (b) are the same. However, observe that g lies on the left half-plane of (d, e) in Figure 3(a) and on the right-half plane of (d, e) in Figure 3(b). We enumerated these geometric thickness two representations by performing the steps S_1 and S_2 below for every point-set order-type P that consists of nine points.

S_1. Construct a straight-line drawing Γ of K_9 on P.
S_2. For every edge e^* in Γ, execute the following.
- Delete e^* and test whether the proper intersection graph[1] determined by the remaining straight lines is 2-colorable. If the graph is 2-colorable, then Γ is a geometric thickness two representation of K_9'.
- Reinsert e^* in Γ.

Let $\Gamma_i, 1 \leq i \leq 3$, be the drawings of K_9' depicted in Figures 3(a)–(c), respectively. The seven black edges in each of these drawings do not contain any crossing, i.e., these edges are shared in both triangulations. By E_i and E_i' we denote the set of all edges, and the set of black edges in Γ_i, respectively. Let $E_i'' = E_i \setminus E_i'$. We verify that the partition of the edges of E_i'' into red and blue is unique by checking that the proper intersection graph G_i of E_i'' is connected.

[1] Each vertex in a *proper intersection graph* G *of a set of straight line segments* corresponds to a distinct line, and two vertices of G are adjacent if and only the corresponding straight lines properly cross.

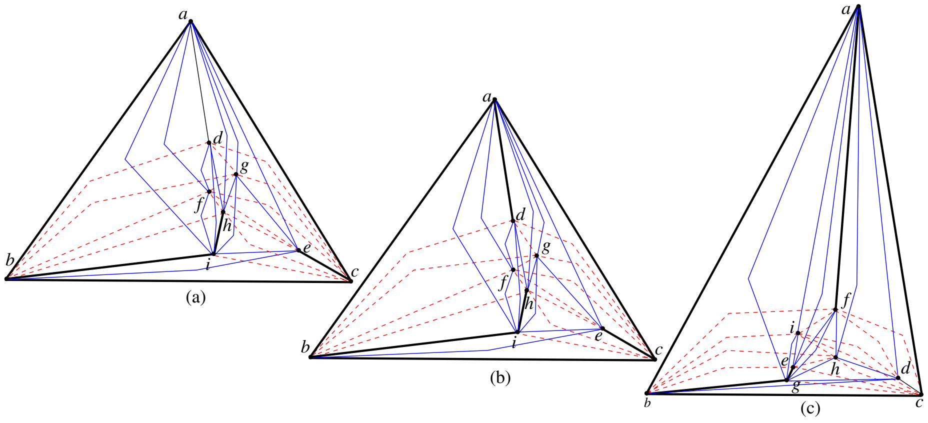


Fig. 3. (a)–(c) Geometric thickness two representations of K_9', where $K_9' = K_9 \setminus (d, e)$. Edges are drawn with polylines for clarity.

Fact 1 *Let Γ be a geometric thickness two representation of K_9'. Then the partition of the straight-line edges of Γ, except the seven edges that do not contain any proper crossing, into two planar layers is unique.*

We now categorize the vertices of a K_9' into two types: *unsaturated* (vertices of degree 7), and *saturated* (vertices of degree 8). The vertices d and e of Figures 3(a)–(c) are unsaturated, and all other vertices are saturated vertices. Take a new vertex and make it adjacent to the two unsaturated vertices and any five saturated vertices of a K_9'. Let the resulting graph with 10 vertices be G_s. The following theorem shows that $\overline{\theta}(G_s) = 3 > \theta(G_s) = 2$, whose proof is omitted due to space constraint. The idea of the proof is first to show a thickness two representation of G_s, and then to show that G_s contains a vertex v that is not straight-line visible to all of its neighbors in any geometric thickness two representation of $G_s \setminus v$. Finally, the proof shows that for every graph G with at most 9 vertices, $\overline{\theta}(G) = \theta(G)$.

Theorem 2. *The smallest graph G (with respect to the number of vertices) such that $\overline{\theta}(G) = 3 > \theta(G) = 2$ contains* 10 *vertices.*

4 Geometric Thickness Two Graph Recognition

Our proof that testing whether $\overline{\theta}(G) \leq 2$ is NP-hard is inspired by a technique of [13]. We reduce the 3SAT problem that given a CNF-system with a set U of variables and a collection C of clauses over U, where each clause consists of exactly three literals, asks whether there is a satisfying truth assignment for U.

Given an instance $I(U, C)$ of 3SAT, we construct a graph G such that there exists a satisfying truth assignment for U if and only if there exists a geometric thickness two representation of G. Before describing the construction of G, we observe some properties of the geometric thickness two representations of K_9'.

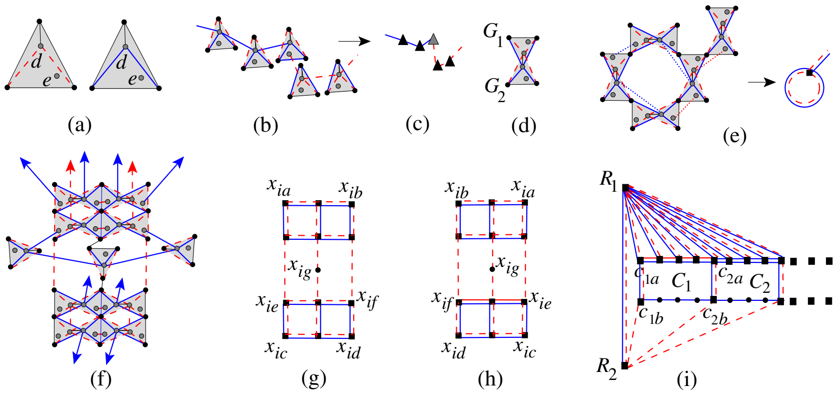


Fig. 4. (a) Hypothetical geometric thickness two representations of K'_9. (b)–(c) A path through the unsaturated vertices, and its hypothetical representation. (d)–(e) A geometric thickness two representation with 5 copies of Γ_H, and its hypothetical representation. Note that the order (inside or outside) among the red (dashed) and blue (solid) lines are not important. (f) A literal gadget, the arrows denote possible connections with other gadgets. (g)–(h) Hypothetical representation of a literal gadget, when the literal is (g) true, and (h) false. (i) Illustration for clause gadgets.

By Fact 1, observe that every geometric thickness two drawing of K'_9 can be denoted by one of the two hypothetical representations shown in Figure 4(a). Each black (respectively, gray) vertex of Figure 4(a) is a saturated (respectively, unsaturated) vertex[2]. We denote the two planar layers of a drawing by the *red layer* L_r (containing the red edges) and *blue layer* L_b (containing the blue edges). Each black edge can be assigned an arbitrary layer unless it is crossed by some other edge. Observe from Figure 3 that if the unsaturated vertex d is surrounded by a blue (respectively, red) triangle, then the other unsaturated vertex e is surrounded by a red (respectively, blue) triangle. Therefore, if we create a path connecting the unsaturated vertices of several copies of K'_9, as shown in Figure 4(b), then the edges of that path must be of same color. Although here we require the copies of K'_9 to be non-overlapping and non-nesting, this will not be significant for our reduction. In the hypothetical representation, we denote each K'_9 with either a black triangle (if its incident edges are of same color), or a gray triangle (if its incident edges are of different colors).

Let G_1 and G_2 be two distinct copies of K'_9. Let s_i and u_i, be a saturated and an unsaturated vertex of G_i, $1 \le i \le 2$, respectively. Let H be the graph that is obtained by merging s_1 and u_1 with u_2 and s_2, respectively, and then removing the resulting multi-edges (if any). Observe that a geometric thickness two representation of H can be constructed by taking two copies of the drawing of Figure 3(a), and then placing one copy on top of the other copy by rotating it such that the two drawings share the edge (s_i, u_i). Figure 4(d) illustrates a

[2] As defined in Section 3, a vertex v is saturated if it has degree 8, and unsaturated if it has degree 7.

hypothetical geometric thickness two drawing Γ_H of H. Examining every candidate pair $((s_1, u_1), (s_2, u_2))$ in the drawing of Figures 3(a)–(c) we can observe that in every geometric thickness two drawing of H, the vertex $u_1(=s_2)$ must lie on the convex hull of the drawing of G_1. Similarly, the vertex $u_2(=s_1)$ must lie on the convex hull of the drawing of G_2. Consequently, H can be represented by Γ_H. Figure 4(e) shows how to connect several copies of Γ_H to create a geometric thickness two representation where no two Γ_H properly cross, and then shows its hypothetical representation. We sometimes use a black square to illustrate the connection between two different drawings, as shown in Figure 4(e).

Construction of G. Assume that $I(U, C)$ contains l literals and t clauses. For every literal $x_i, 1 \le i \le l$, construct a literal gadget Γ_{x_i} as depicted in Figure 4(f). Figure 4(g) is a simplified representation of the literal gadget, which will correspond to the value true. On the other hand, Figure 4(h) (i.e., the mirror embedding of Figure 4(f)) will correspond to the value false. We call the vertex x_{ig} the *central vertex* of Γ_{x_i}. By the *lower-half* of Γ_{x_i} we denote the subgraph induced by x_{ic}, x_{id}, x_{ie} and x_{if}. The vertices of Γ_{x_i} that are not in the lower-half, induce the *upper-half.* Construct the clause gadgets as shown in Figure 4(i). Observe that the vertices R_1 and R_2 are incident to a set of rectangles, where each rectangle (i.e., *clause box*) B_r corresponds to a clause $C_r, 1 \le r \le t$. The top, left and right sides of B_r constitutes a chain of Γ_H, and the bottom side is a path of three vertices. We merge the central vertices of the three literals of C_r with a distinct vertex of the bottom side of B_r. Let E_b be the set that consists of the edges on the bottom side of the clause boxes. The construction of the clause gadget ensures that the edges of E_b lie on the same planar layer, w.l.o.g., on blue layer L_b. Then the clause boxes force the edges (c_{rb}, R_2) to lie on the other planar layer, i.e., the red layer L_r. For each literal gadget Γ_l, we add an edge between R_2 and the gadget such that the edge is forced to lie on L_r. Similarly, for each literal gadget Γ_l, add an edge between the top side of the clause box and the gadget such that the edge is forced to lie on L_b.

We now add some edges among the literal gadgets that correspond to the same literal. For every literal x_i, we order its literal gadgets according to their appearance in different clauses. Let $l_1, l_2, \ldots, l_{t'}$ be the literal gadgets that correspond to the literal x_i. Then for each index $q, 1 \le q < t'$, we add an edge between the vertex x_{ie} (respectively, x_{if}) of l_q and the vertex x_{ic} (respectively, x_{id}) of l_{q+1}. We denote all these edges by E_l. Figure 5(a) shows how the edges in E_l forces the corresponding literal gadgets to have the same truth value.

Finally, we add a few more edges among the literal gadgets that belong to the same clause. Let x_i, x_j, x_k be the three literals of C_r. We then add a path between x_{ib} and x_{ja} that contains three unsaturated vertices of two copies of K_9', as shown in Figure 5(b). Similarly, we add a path between x_{ka} and x_{jb} that contains three unsaturated vertices of two copies of K_9'.

Theorem 3. *It is NP-hard to determine whether the geometric thickness of an arbitrary graph is at most two.*

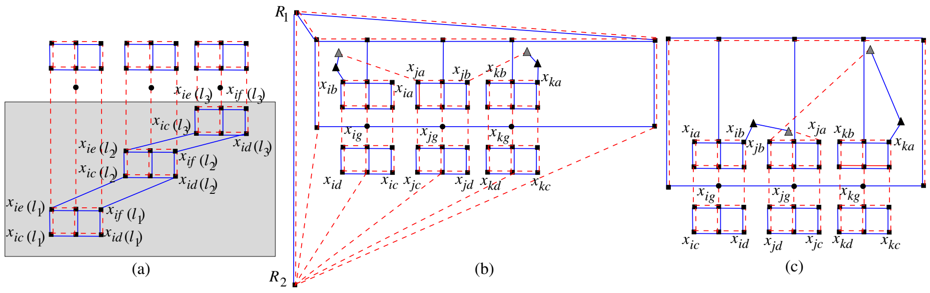


Fig. 5. (a) Literal gadgets that correspond to the same literal, which is true. (b)–(c) Two gadgets: (b) x_j is true, x_i, x_k are false, (c) x_i is true, x_j, x_k are false.

Proof. Given an instance $I(U, C)$ of 3SAT, we first construct the corresponding graph G and then prove that $I(U, C)$ has a satisfying truth assignment if and only if G has a geometric thickness two representation. The proof is similar to the hardness proof for simultaneous straight-line embedding of two planar graphs [13]; thus we give only an outline of the proof.

Assume first that $I(U, C)$ is satisfiable. We now construct a geometric thickness two representation of G. We draw the clause gadgets as shown in Figure 4(i). Then for each literal, we assign a horizontal region and draw its corresponding gadgets as shown in Figure 5. Finally, for each clause $C_r = (x_i \vee x_j \vee x_k)$, we draw the paths between x_{ib} and x_{ja}, and x_{ka} and x_{jb} such that no two edges on the same layer cross, as follows. Observe that at least one literal in C_r is true. If the literal in the middle, i.e., x_j, is true, then we draw these paths as shown in Figure 5(b). Observe that we can adapt this drawing for the case when one of x_i and x_k, or both are true. Similarly, if the literal x_i or x_j is true, w.l.o.g., x_i, then we draw these paths as shown in Figure 5(c). Observe that we can adapt this drawing for the case when one of x_j and x_k, or both are true.

Assume now that G has a geometric thickness two representation. Observe that the graph induced by the edges in L_r in Figure 4(i) is a subdivision of a triconnected planar graph. Consequently, by a theorem of Whitney [22], such a graph has a unique combinatorial embedding up to homomorphisms of the plane. We choose the planar embedding such that the edge (R_1, R_2) lies on the outerface and the clause boxes obtain the same order as depicted in Figure 4(i). Observe that upper-halves of the three literal gadgets of each clause are forced to lie inside the corresponding clause box. Hence the paths between x_{ib} and x_{ja}, and x_{ka} and x_{jb} must be drawn inside the clause box. Consequently, at least one of the literal gadget must correspond to true in each clause box, otherwise, there must be a crossing in the same planar layer. Since the edges in E_l forces the literal gadgets corresponding to the same literal to have consistent embeddings, we find a satisfying truth assignment for $I(U, C)$. □

5 NP-hardness of Colorability

In this section we show the NP-hardness of coloring a graph with geometric thickness t with $4t-1$ colors. By $I(G,T,C)$ we denote the problem of coloring a graph G with C colors, where $\overline{\theta}(G) \le T$. We first introduce a few definitions. A *join* between two graphs is an operation that given two graphs, adds all possible edges that connect the vertices of one graph with the vertices of the other graph. By $\mathcal{G}_t$ we denote a class of thickness t graphs that satisfies the following conditions: (1) $\mathcal{G}_1$ is the class of planar graphs. (2) If $t>1$, then $\mathcal{G}_t$ consists of the graphs obtained by taking a join of K_2 and G, where $G \in \mathcal{G}_{t-1}$.

Observe that $\overline{\theta}(\mathcal{G}_t) \le t$. We now have the following lemma, whose proof is omitted due to space constraints.

Lemma 1. *It is NP-hard to color an arbitrary graph $G \in \mathcal{G}_t$ with $2t+1$ colors.*

We use Lemma 1 to prove the NP-hardness of coloring geometric thickness t graphs with $4t-1$ colors. We employ induction on t. If $t=1$, then coloring a planar graph (i.e., $t=1$) with $4t-1=3$ colors is NP-hard [15]. We now assume inductively that for any $t' < t$, it is NP-hard to color a geometric thickness t' graph with $4t'-1$ colors. To prove the hardness of coloring a geometric thickness t graph with $4t-1$ colors, we reduce the hardness of coloring a geometric thickness $t-1$ graph with $2(t-1)+1$ colors. Given an instance $I(G, t-1, 2(t-1)+1)$, we construct a graph H such that $\overline{\theta}(H) \le t$ and H is $4t-1$ colorable if and only if G is $2(t-1)+1$ colorable.

Let the number of vertices in G be n. Take n copies $H_1, H_2, \ldots, H_n$ of K_{2t}, and join each vertex of G with a distinct H_i, $1 \le i \le n$. Finally, take a copy H' of K_{2t-1} and join it with every H_i. Let the resulting graph be $H(G,t)$. To prove that $\overline{\theta}(H(G,t)) = t$, we first review a construction of Dillencourt et al. [7] that gives a thickness t representation of K_{4t}. They proved that the $4t$ vertices of a K_{4t} can be arranged in two rings of $2t$ vertices each, an outer ring and an inner ring, such that it can be embedded using exactly t planar layers. The vertices of the inner ring are arranged to form a regular $2t$-gon. For each pair of diametrically opposite vertices, a zigzag path is constructed as illustrated in Figure 6(a). This path has exactly one diagonal connecting diametrically opposite points (i.e., the diagonal connecting the two gray points in the figure.) The union of these zigzag paths, taken over all t pairs of diametrically opposite vertices, contains all the edges of K_{2t} in the inner ring, as shown in Figure 6(b). For each pair of diametrically opposite vertices, we can draw rays from each vertex of the zigzag path, in two opposite directions, so that none of the rays crosses any edge of the zigzag path. These rays, in each direction, meet at a common point (e.g., p or q) forming the outer ring, as shown in Figure 6(c).

Lemma 2. $\overline{\theta}(H(G,t)) \le t$, *where $t > 1$ and $G \in \mathcal{G}_{t-1}$.*

Proof. We compute a thickness t representation of $H(G,t)$, as follows. Since $G \in \mathcal{G}_{t-1}$, $\overline{\theta}(G) \le t-1$. Take a thickness $t-1$ representation of G and rotate it

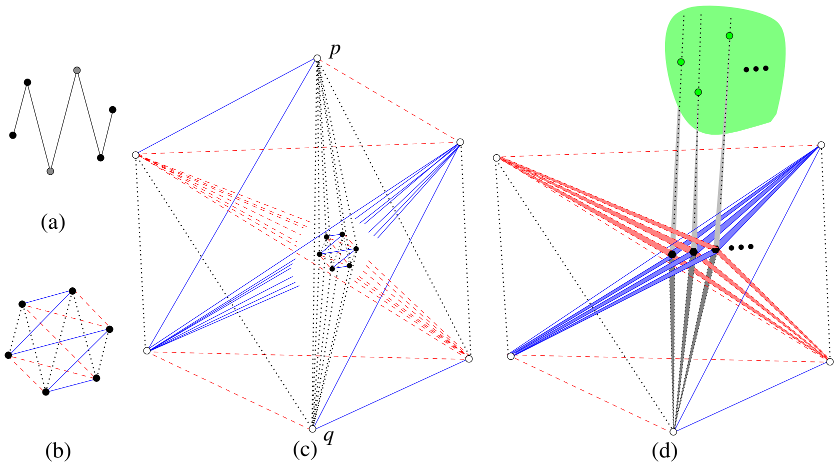


Fig. 6. (a)–(c) Dillencourt et al.'s construction [7]. (a) A zigzag path in the inner ring. (b) K_{2t}, where t=3. (c) K_{4t}, where t=3. (d) The geometric thickness two representation of $H(G,t)$, where t=3. Each subgraph H_i is determined by an inner ring. The vertices of G are in the green region.

(if necessary) such that no two vertices lie on the same vertical line. Let the resulting drawing be Γ. Now construct an outer ring as in Dillencourt et al.'s construction [7], and delete a vertex from the ring to obtain a thickness t drawing of H', as shown in Figure 6(d). For each H_i, construct an inner ring that lies along the vertical line determined by its corresponding vertex in Γ. Figure 6(d) shows this correspondence with dotted lines. All the edges that connect the vertices of G with the vertices of H_i (i.e., the edges in the light-gray region) lie in the t-th layer. Note that the inner rings must be scaled down small enough such that these edges do not create any edge crossing in the t-th layer. □

Theorem 4. *It is NP-hard to color an arbitrary geometric thickness t graph with $4t-1$ colors.*

Proof (Outline). If $t=1$, then coloring a planar graph (i.e., $t=1$) with $4t-1=3$ colors is NP-hard [15]. Assume now that $t>1$. Given an instance $I(G, t-1, 2(t-1)+1)$, where $G \in \mathcal{G}_{t-1}$, we construct a corresponding graph $H(G,t)$. We prove that G is $2(t-1)+1$ colorable if and only if $H(G,t)$ is $4t-1$ colorable. The details are omitted due to space constraints. □

Acknowledgements. We thank the anonymous reviewers for their suggestions to improve the presentation of the paper. The third author received financial support from the Faculty of Graduate Studies at the University of Manitoba for partial travel expenses to present this paper.

References

1. Aichholzer, O., Aurenhammer, F., Krasser, H.: Enumerating order types for small point sets with applications. Order 19(3), 265–281 (2002)
2. Alekseev, V.B., Gonchakov, V.S.: Thickness of arbitrary complete graphs. Math USSR Sbornik 30(2), 187–202 (1976)
3. Appel, K., Haken, W.: Every planar map is four colorable. Part I. Discharging. Illinois Journal of Mathematics 21, 429–490 (1977)
4. Barát, J., Matoušek, J., Wood, D.R.: Bounded-degree graphs have arbitrarily large geometric thickness. Electronic Journal of Combinatorics 13 (2006)
5. Beineke, L.W., Harary, F., Moon, J.W.: On the thickness of the complete bipartite graph. Math. Proc. of the Cambridge Philosophical Society 60, 1–6 (1964)
6. Brandenburg, F.-J., Eppstein, D., Goodrich, M.T., Kobourov, S.G., Liotta, G., Mutzel, P.: Selected open problems in graph drawing. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 515–539. Springer, Heidelberg (2004)
7. Dillencourt, M.B., Eppstein, D., Hirschberg, D.S.: Geometric thickness of complete graphs. Journal of Graph Algorithms and Applications 4(3), 5–17 (2000)
8. Dujmovic, V., Wood, D.R.: Graph treewidth and geometric thickness parameters. Discrete & Computational Geometry 37(4), 641–670 (2007)
9. Duncan, C.A.: On graph thickness, geometric thickness, and separator theorems. Computational Geometry: Theory and Applications 44(2), 95–99 (2011)
10. Duncan, C.A., Eppstein, D., Kobourov, S.G.: The geometric thickness of low degree graphs. In: Proc. of SoCG, pp. 340–346. ACM (2004)
11. Eppstein, D.: Separating thickness from geometric thickness. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 150–161. Springer, Heidelberg (2002)
12. Erten, C., Kobourov, S.G.: Simultaneous embedding of planar graphs with few bends. Journal Graph Algorithms and Applications 9(3), 347–364 (2005)
13. 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)
14. Hutchinson, J.P., Shermer, T.C., Vince, A.: On representations of some thickness-two graphs. Comp. Geom.: Theory and Applications 13(3), 161–171 (1999)
15. Jensen, T.R., Toft, B.: Graph coloring problems. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., New York (1995)
16. Kainen, P.C.: Thickness and coarseness of graphs. Abhandlungen aus dem Mathematischen Seminar der Univ. Hamburg 39(88–95) (1973)
17. Mansfield, A.: Determining the thickness of a graph is NP-hard. In: Mathematical Proceedings of the Cambridge Philosophical Society, vol. 93, pp. 9–23 (1983)
18. McGrae, A.R.A., Zito, M.: The complexity of the empire colouring problem. Algorithmica, 1–21 (2012) (published online)
19. Mutzel, P., Odenthal, T., Scharbrodt, M.: The thickness of graphs: A survey. Graphs Combin. 14, 59–73 (1998)
20. Ringel, G.: Fabungsprobleme auf flachen und graphen. VEB Deutscher Verlag der Wissenschaften (1950)
21. Tutte, W.T.: The thickness of a graph. Indag. Math. 25, 567–577 (1963)
22. Whitney, H.: Congruent graphs and the connectivity of graphs. American Journal of Mathematics 54(1), 150–168 (1932)

The Same Upper Bound for Both: The 2-Page and the Rectilinear Crossing Numbers of the n-Cube

Luerbio Faria[1], Celina M.H. de Figueiredo[2], R. Bruce Richter[3], and Imrich Vrťo[4]

[1] Department of Computation, Institute of Mathematics and Statistics, University of State of Rio de Janeiro

[2] COPPE - Systems and Computation, Federal University of Rio de Janeiro, Brazil[⋆]

[3] Department of Combinatorics & Optimization, Faculty of Mathematics, University of Waterloo, Canada[⋆⋆]

[4] Institute of Mathematics, Slovak Academy of Sciences, Slovak Republic[⋆⋆⋆]

Abstract. We present two main results: a 2-page drawing and a rectilinear drawing of the n-dimensional cube Q_n. Both drawings have the same number $\frac{125}{768}4^n - \frac{2^{n-3}}{3}\left(3n^2 + \frac{9+(-1)^{n+1}}{2}\right)$ of crossings, even though they are given by different constructions. The first improves the current best general 2-page drawing, while the second is the first non-trivial rectilinear drawing of Q_n.

Keywords: two-page crossing number, rectilinear crossing number, n-cube, topological graph theory.

1 Introduction

A *drawing* $D(G)$ of a graph $G = (V, E)$ is a mapping of G to a topological space (usually the plane, but not always). The vertices go into distinct points called *nodes*, and an edge maps into an *arc* – a homeomorphic image of the closed interval $[0, 1]$ – such that its interior contains no node and the endpoints of the arc associated to an edge $e = uv \in E$ are the nodes associated to the end vertices of e: u and v.

A *good drawing* $D(G)$ of a graph G is a drawing where each pair of arcs have at most one point in common which is either a node or a *crossing*. All drawings considered in this paper are good.

An *embedding* of a graph $G = (V, E)$ in a topological space S is a drawing of G in S without crossings.

⋆ The work was supported by FAPERJ, CNPq and CAPES.

⋆⋆ The work was supported by NSERC.

⋆⋆⋆ The work was supported by the Slovak Research and Development Agency under the contract No. ESF-EC-0009-10, and by the VEGA No. 2/0136/12.

Given a graph $G = (V, E)$, the *crossing number* $\nu(G)$ of G is the minimum number of crossings in a drawing of G in the plane [10].

For a positive integer k, and a graph G, a *k-page drawing of* G is a drawing of G in the union H_k of k closed half-planes, all having their boundary lines in common, but otherwise disjoint, so that $V(G)$ is contained in the common boundary line and, for each edge e of G, the arc representing e is contained in one of the half-planes [4]. The *page number* of a graph G is the smallest k such that G embeds in H_k.

The *k-page crossing number* $\nu_k(G)$ of a graph G is the minimum number of crossings of edges in a k-page drawing of G.

Since every 1-page drawing is a 2-page drawing, and every 2-page drawing is a planar drawing, $\nu(G) \leq \nu_2(G) \leq \nu_1(G)$. Yannakakis [13] proved that every planar graph has a 4-page embedding and announced the existence of a planar graph that has no 3-page embedding.

The 2-page crossing number of K_n has been recently determined by Ábrego et al. [1]. de Klerk and Pasechnik [6] used semidefinite programming to find estimates for $\nu_2(K_{m,n})$ and $\nu(K_n)$. Masuda et al. [12] proved that it is NP-complete to determine if there is a 2-page drawing of G having at most k crossings using a given linear order. This problem is called *Fixed Linear Crossing Number.*

A *rectilinear drawing* of a graph $G = (V, E)$ is a drawing of G in the plane such that edges are drawn as straight line segments. The *rectilinear crossing number* $\overline{cr}(G)$ of a graph G is the smallest number of crossings in a rectilinear drawing of G.

For $n \geq 0$, the *n-cube* Q_n has as its vertex set $V(Q_n)$ all binary strings of length n and two vertices are adjacent if and only if the corresponding strings differ in precisely one position. (Note that Q_0 has one vertex and no edges.) Thus, each edge e determines the position at which its incident vertices differ; if this is position i, then *e is in the i-th dimension.*

Very little is known about exact values of $\nu(Q_n)$, $\nu_2(Q_n)$, and $\overline{cr}(Q_n)$, respectively. It is known that if $n \leq 3$, then $\nu(Q_n) = \nu_2(Q_n) = \overline{cr}(Q_n) = 0$. Dean and Richter [5] showed that $\nu(Q_4) = 8$; now appropriate drawings imply that $\nu_2(Q_4) = \overline{cr}(Q_4) = 8$. Buchheim and Zheng [3] used a brute force computer approach with MAX CUT on an auxiliary graph to prove that $\nu_2(Q_5) = 60$, $\nu_2(Q_6) = 368$, and $\nu_2(Q_7) \leq 1874$.

Madej [11] exhibited a 2-page drawing for Q_n, thereby showing that $\nu_2(Q_n) \leq \frac{4^n}{6} - 2^{n-3}n^2 - 2^{n-4}3 + \frac{(-2)^n}{48}$. In these 2-page drawings, Q_5 has 64 crossings, Q_6 has 384 crossings, and Q_7 has 1920 crossings. Madej also exhibited a drawing for Q_5 with 56 crossings; this is not a 2-page drawing. At WG'2003 [8], Faria et al. [9] exhibited drawings that show $\nu(Q_5) \leq 56$, $\nu(Q_6) \leq 352$, and $\nu(Q_7) \leq 1760$. These are in line with the conjecture of Erdős and Guy [7]: $\nu(Q_n) \leq 4^n \frac{5}{32} - \lfloor \frac{n^2+1}{2} \rfloor 2^{n-2}$ [9].

In Section 3, we give a 2-page drawing of Q_n having

$$\frac{125}{768}4^n - \frac{2^{n-3}}{3}\left(3n^2 + \frac{9 + (-1)^{n+1}}{2}\right)$$

crossings. Our drawings of Q_5 and Q_6 achieve the optimal 2-page values of Buchheim and Zheng and improve slightly their upper bound for Q_7.

By way of comparison, our leading term $\frac{125}{768}$ is located within the interval between the leading terms $\frac{5}{32}$ and $\frac{1}{6}$ of the upper bounds of Faria et al. [9] for the crossing number of the n-cube and of Madej's [11] for the 2-page crossing number of the n-cube, respectively: $\frac{5}{32} = \frac{125}{800} < \frac{125}{768} < \frac{125}{750} = \frac{1}{6}$.

In Section 4 we present a rectilinear drawing of Q_5 having 60 crossings. We then use a slight modification of the vertex-cloning technique of [9] and induction to obtain a rectilinear drawing of Q_n. Despite being a completely different construction from our 2-page drawing, remarkably, the two drawings of Q_n we present have the same number of crossings.

All proofs are omitted here and will be included in the journal version.

2 Properties of Madej's Linear Drawing

We use Madej's construction within our construction; in particular, we use his drawing of Q_{n-5} to obtain a drawing of Q_n. This section is devoted to understanding both his drawing and its crossings.

The construction of Madej [11] is done by induction from the drawing of Q_0. Place side by side two copies of the same linear drawing of Q_n, with the order of the vertices in the copy on the right being inverted with respect to the copy on the left. The bit 0 is added to the end of each vertex in the copy on the left and the bit 1 is added to the end of each vertex of the copy on the right. The edges of dimension n are drawn in the upper region if n is even and in the lower region if n is odd, as semicircles joining symmetric vertices in the two copies. Examples of this construction with $n = 0, 1, 2, 3, 4$ are shown in Figure 1. We denote the drawing of Madej for the n-cube by M_n and its number of crossings by $M(n)$.

Let S be a set of vertical straight lines, each one passing through each vertex of M_n. Let u_n be the number of crossings of the edges in the 2-page drawing of Madej with the vertical straight line set S. Madej [11] established Theorem 1:

Theorem 1. $u_n = \frac{4^n}{2} - (n+1)2^{n-1}$.

We show in Figure 1 Madej's drawings for the 0, 1, 2, 3 and 4-cubes with their corresponding values for u_n in column 3.

We denote by D_n the 2-page drawing for the n-cube presented in this paper and by $D(n)$ its number of crossings. For our analysis of D_n, we split the value u_n into the number $U_p(n)$ of crossings with the vertical lines that occur in the upper half plane and the number $L_w(n)$ that are in the lower half plane.

We summarize the relevant facts about these numbers, which are listed, for $n \leq 4$, in Columns 4 and 5 in Figure 1.

Lemma 2. *Let n be a positive integer. Then:*

1. *if n is even, then $U_p(n) = \frac{4^n}{3} - \frac{2^n}{3} - 2^{n-2}n$;*
2. *if n is odd, then $U_p(n) = \frac{4^n}{3} - \frac{2^{n+1}}{3} - 2^{n-2}(n-1)$;*

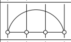
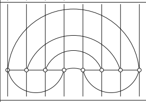
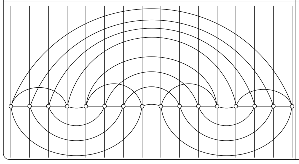
2-page drawing	Q_n	u_n	$U_p(n)$	$L_w(n)$
	Q_0	0	0	0
	Q_1	0	0	0
	Q_2	2	2	0
	Q_3	16	12	4
	Q_4	88	64	24

Fig. 1. The 2-page drawings of Madej [11] for Q_0, Q_1, Q_2, Q_3, and Q_4 with the corresponding values of $u_n, U_p(n)$ and $L_w(n)$

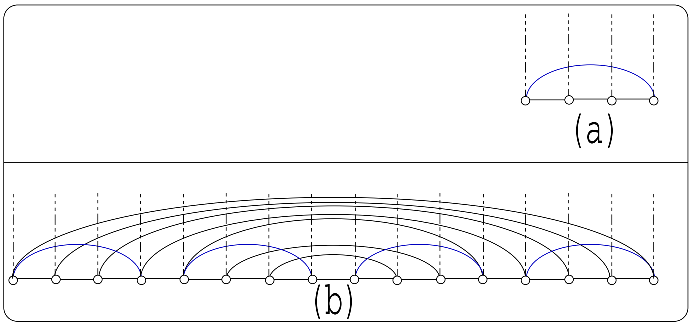


Fig. 2. Drawings for the upper plane of D_n with $n = 2$ (a), and with $n = 4$ (b)

3. $L_w(n + 1) = 2U_p(n)$*; and*
4. $U_p(n) + L_w(n) = u_n$.

3 A 2-page Drawing of Q_n

In this section we describe our drawing of Q_n.

The linear order of the vertices in our drawing of Q_n is the same as Madej's. We start with 2-page drawings of Q_5 (Figure 4), Q_6 (Figure 5), and Q_7 (Figure 6). For $n \geq 7$, the drawing of Q_n consists of 32 copies of Madej's 2-page drawing of Q_{n-5}. The five highest dimensional edges are added so that each vertex of a Q_{n-5} has two on one side and three on the other.

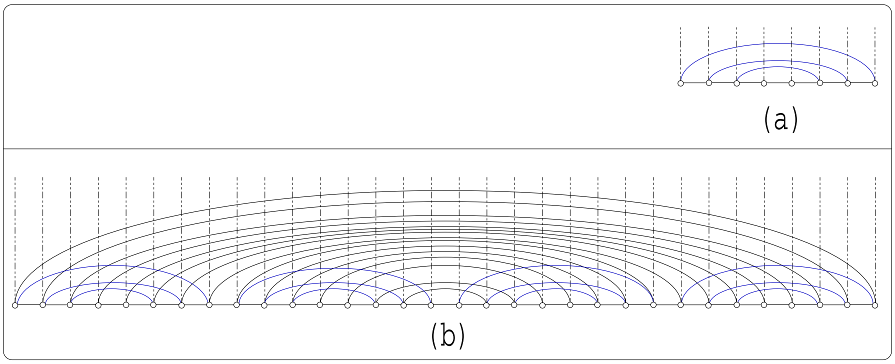


Fig. 3. Drawings for the upper plane of D_n with $n = 3$ (a), and with $n = 5$ (b)

Enumerate the vertices of Q_n as $v_1, v_2, v_3, \ldots, v_{2^n}$ for the vertices of Q_n from left to right in the drawing M_n of Madej. We begin by specifying the locations of the edges of dimension $\geq n-4$. For $n \geq 5$, our 2-page drawing D_n of Q_n uses the following choices.

1. The edges of dimension n either joining the vertices $v_1, v_2, \ldots, v_{2^{n-4}}$ to the vertices $v_{2^{n-4}15+1}, v_{2^{n-4}15+2}, \ldots, v_{2^n-1}, v_{2^n}$, or joining the vertices $v_{2^{n-4}7+1}$, $v_{2^{n-4}7+2}, \ldots, v_{2^{n-1}-1}, v_{2^{n-1}}$ to the vertices $v_{2^{n-1}+1}, v_{2^{n-1}+2}, \ldots, v_{2^{n-1}+2^{n-4}}$ are in the lower half-plane, while all other edges of dimension n are in the upper half-plane.
2. All edges of dimension $n-1$ belong to the lower half-plane.
3. The edges of dimension $n-2$ are divided into five sets. For $j \in \{0, 2^{n-2}, 2^{n-1}, 2^{n-2}3\}$ (four sets), the edges joining the vertices $v_{1+j}, v_{2+j}, v_{3+j}, \ldots, v_{2^{n-5}+j}$, to the vertices $v_{2^{n-5}7+1+j}, v_{2^{n-5}7+2+j}, v_{2^{n-5}7+3+j}, \ldots, v_{2^{n-2}+j}$ are in the upper half-plane. All other edges of dimension $n-2$ (the fifth set) are in the lower half-plane.
4. All edges of dimension $n-3$ belong to the upper half-plane.
5. The edges of dimension $n-4$ are divided into five sets. For each of the two values $j = 0, 2^{n-1}$, we have two sets of edges, one consisting of the edges joining the vertices $v_{1+j}, v_{2+j}, \ldots, v_{2^{n-5}+j}$ to the vertices $v_{2^{n-5}+1+j}, v_{2^{n-5}+2+j}, \ldots,$ $v_{2^{n-4}+j}$, and the other consists of the edges joining the vertices $v_{2^{n-5}14+1+j}$, $v_{2^{n-5}14+2+j}, \ldots, v_{2^{n-5}15+j}$ to the vertices $v_{2^{n-5}15+1+j}, v_{2^{n-5}15+2+j}, \ldots,$ $v_{2^{n-1}+j}$. All four of these sets are in the upper half-plane. The fifth set is all other edges of dimension $n-4$; these are in the lower half-plane.

We complete the construction of D_n by considering 32 sets of edges, each using the drawing of Madej for Q_{n-5}. These 32 drawings are partitioned according to

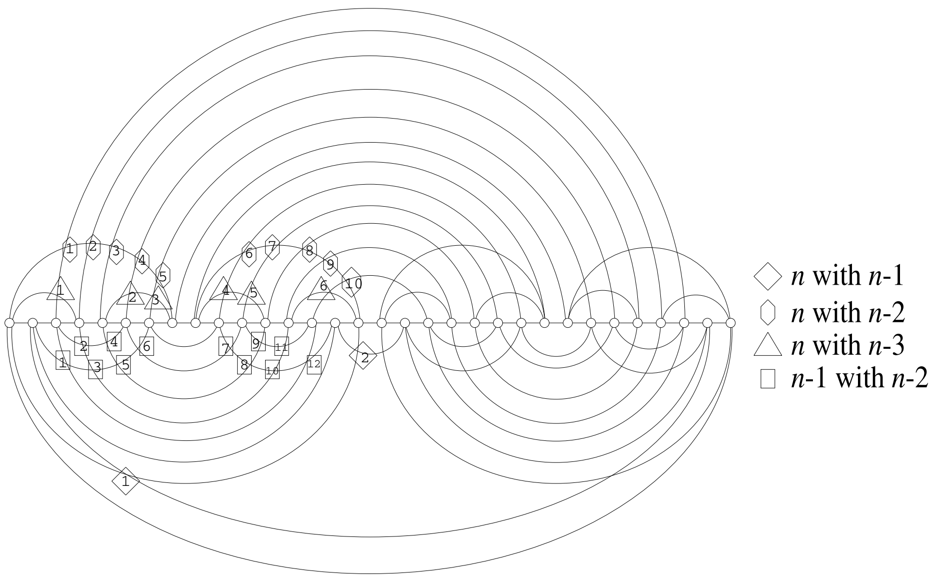


Fig. 4. Optimum 2-page drawing for Q_5 with $\nu_2(Q_5) = 60$. The crossings are labelled according to the pair of dimension of the edges are $(n, n-1)$, $(n, n-2)$, $(n, n-3)$, $(n-1, n-2)$, in the half of the Figure.

the eight values of $j \in \{0, 2^{n-3}, 2^{n-2}, 2^{n-3}3, 2^{n-1}, 2^{n-3}5, 2^{n-2}3, 2^{n-3}7\}$. For each of these j, there are four drawings of Q_{n-5}, as follows:

1. the Q_{n-5} on the vertices $v_{1+j}, v_{2+j}, v_{3+j}, \ldots, v_{2^{n-5}+j}$ is upside down;
2. the Q_{n-5} on the vertices $v_{2^{n-5}3+1+j}, v_{2^{n-5}3+2+j}, v_{2^{n-5}3+3+j}, \ldots, v_{2^{n-3}+j}$ is upside down;
3. the Q_{n-5} on the vertices $v_{2^{n-5}+1+j}, v_{2^{n-5}+2+j}, v_{2^{n-5}+3+j}, \ldots, v_{2^{n-4}+j}$ is rightside up; and
4. the Q_{n-5} on the vertices $v_{2^{n-4}+1+j}, v_{2^{n-4}+2+j}, v_{2^{n-4}+3+j}, \ldots, v_{2^{n-5}3+j}$ is rightside up.

For the convenience of the reader, our 2-page drawings of Q_5, Q_6, and Q_7 are shown in Figures 4, 5, and 6, respectively.

We divide the crossings in this drawing into three sets. In $\mathcal{C}_{\leq n-5}$, we have the crossings between edges of dimension at most $n-5$; each of these occurs within one of the thirty-two Q_{n-5}'s. The set $\mathcal{C}_{\leq\geq}$ consists of those crossings involving an edge of dimension at most $n-5$ with an edge of dimension at least $n-4$. Finally, $\mathcal{C}_{\geq n-4}$ has the crossings between edges that both have dimension at least $n-4$. We determine their sizes, as follows.

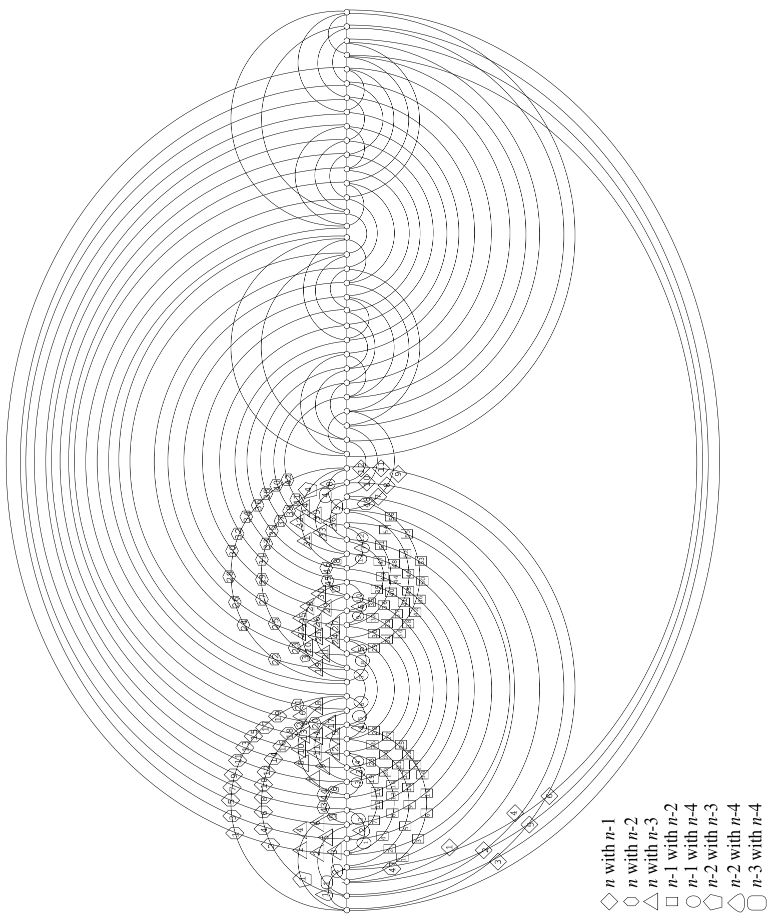


Fig. 5. Optimum 2-page drawing for Q_6 with $\nu_2(Q_6) = 368$

Lemma 3. *If n is an integer with $n \geq 6$, then*

1. $|\mathcal{C}_{\leq n-5}| = 32M(n-5)$.
2. $|\mathcal{C}_{\leq\geq}| = 64U_p(n-5) + 96L_w(n-5)$.
3. $|\mathcal{C}_{\geq n-4}| = 31\cdot 4^{n-4} - 2^{n+1}$.

We can now determine $D(n)$, the number of crossings in our drawing.

Theorem 4. $\nu_2(Q_n) \leq D(n) = \frac{125}{768}4^n - \frac{2^{n-3}}{3}\left(3n^2 + \frac{9+(-1)^{n+1}}{2}\right)$.

Table 1. Values for $5 \leq n \leq 15$ and the corresponding current known best upper bounds for the crossing number of the n-cube [9], the upper bounds for the 2-page crossing number of this paper, and the upper bounds for the 2-page crossing number of Madej [11]

Q_n	Crossing number current best upper bound	This paper	Madej
n	$\frac{4^n 5}{32} - \lfloor \frac{n^2+1}{2} \rfloor 2^{n-2}$	$\frac{4^n 125}{768} - \frac{2^{n-3}}{3}(3n^2 + \frac{9+(-1)^{n+1}}{2})$	$\frac{4^n}{6} - 2^{n-3}n^2 - 2^{n-4}3 + \frac{(-2)^n}{48}$
5	56	60	64
6	352	368	384
7	1760	1856	1920
8	8192	8576	8832
9	35712	37376	38400
10	151040	157696	161792
11	624128	651264	667648
12	2547712	2656256	2721792
13	10311680	10747904	11010048
14	41541632	43286528	44335104
15	166846464	173834240	178028544

4 A Rectilinear Drawing of Q_n

In this section, we introduce our rectilinear drawing of Q_n. We proceed by induction, beginning with the drawing R_5 of Q_5 shown in Figure 7. (Recall that, for $n \leq 4$, optimal rectilinear drawings are known.)

For the induction, we require the following notions.

A *mesh one of index n* is a set M_1^n of points in the plane consisting of the points of n-pairs of parallel straight lines by the points 0 and 1 with non-zero slope, plus the points in the interval $[0, 1]$ in the x-axis. In Figure 8, we show an example of each M_1^1, M_1^2, M_1^3 and M_1^5.

Likewise, a *mesh one of index n without* is a subset $M_1^n w$ of a mesh one of index n M_1^n, obtained by deleting one pair of parallel rays from the lower half-plane. In Figure 9 we show a drawing of each of $M_1^1 w$, $M_1^2 w$, $M_1^3 w$ and $M_1^5 w$.

In Lemma 5 we evaluate the number of crossings of M_1^n and $M_1^n w$.

Lemma 5. *For any positive integer n:*

1. *every drawing M_1^n has $n(n-1)$ crossings;*
2. *every drawing $M_1^n w$ has $(n-1)^2$ crossings.*

In order to construct a rectilinear drawing R_{n+1} of Q_{n+1} from a rectilinear drawing R_n of Q_n, consider in turn each vertex v of a rectilinear drawing of R_n.

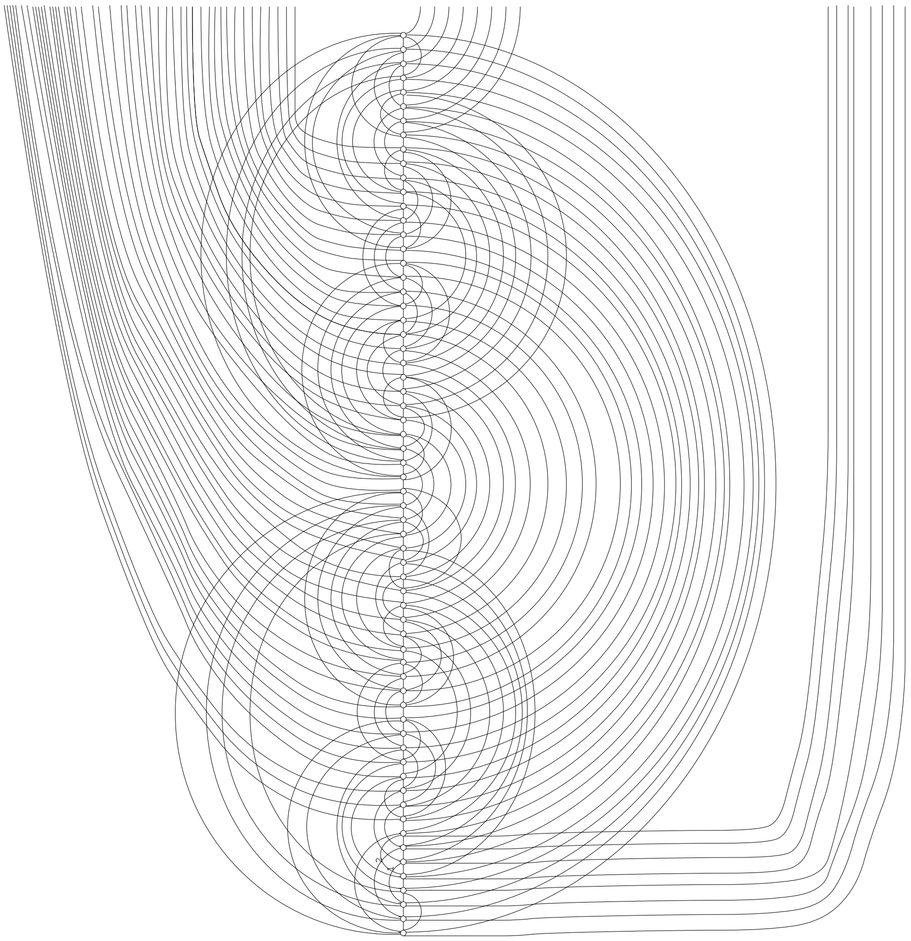


Fig. 6. A 2-page drawing for Q_7 demonstrating that $\nu_2(Q_7) \leq 1856$

Let e_1, e_2, $e_3, \ldots, e_n$ be the straight line segments corresponding to the edges incident to v. We add a twin vertex v' of v, in a tiny neighborhood of v, and edges e'_1, e'_2, e'_3, $\ldots$, e'_n, respectively, with the same size and slope as e_1, e_2, e_3, $\ldots, e_n$, so the corresponding M_1^n or $M_1^n w$ has the minimum number of crossings as determined in Lemma 5 (1) and (2).

We ensure that the twins are placed so that the corresponding new edges match up, so that if uv is an edge of R_n, then $u'v'$ is an edge of R_{n+1}.

Theorem 6. *For $n \geq 5$, $cr(R_n) = \frac{125}{768}4^n - \frac{2^{n-3}}{3}\left(3n^2 + \frac{9+(-1)^{n+1}}{2}\right)$.*

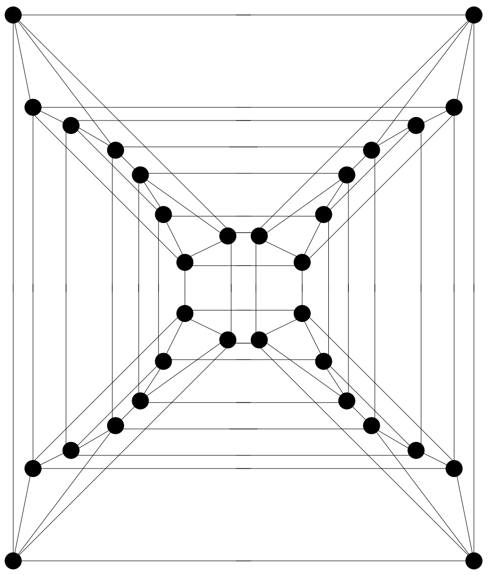


Fig. 7. A rectilinear drawing R_5 for Q_5 with 60 crossings

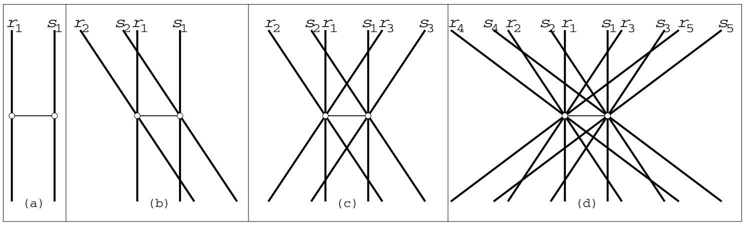


Fig. 8. Drawings of (a) M_1^1, (b) M_1^2, (c) M_1^3 and (d) M_1^5

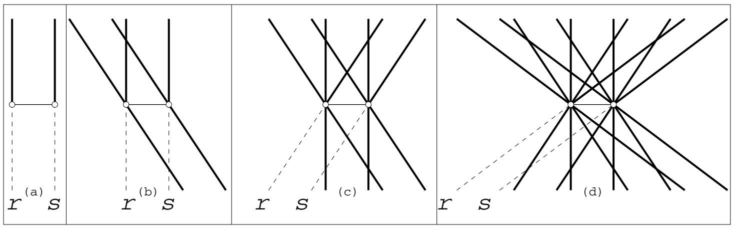


Fig. 9. Drawings of (a) M_1^1w, (b) M_1^2w, (c) M_1^3w and (d) M_1^5w. Dashed straight rays represent the straight lines r and s which are removed in order to define the corresponding drawing to M_1^nw.

5 Final Remarks

Since Madej's construction is suboptimal, it seems plausible that using some other drawing of Q_{n-5} in our construction might reduce the number of crossings. In particular, we might start with our drawing R_5 to get a drawing of Q_{10}.

However, instead of being a strength it is a weakness, since the balance of upper and lower half-plane crossings is not very good. In Madej's drawings, the quotient $U_p(n)/L_w(n)$ is asymptotically equal to 2. Hence, Madej's construction yields a better trade off than this other construction. On the other hand, perhaps one could find another symmetry to explore the same idea with starting point based on another drawing of Q_n.

It is known [1,2] that, for $n \geq 10$, $\nu_2(K_n) < \overline{cr}(K_n)$. On the other hand, if G is a non-Hamiltonian planar triangulation, then $\nu_2(G) > 0$; for such a G, $\overline{cr}(G) < \nu_2(G)$. Therefore, there is no general relation between $\overline{cr}(G)$ and $\nu_2(G)$. It would be interesting if there were a relation like $\nu_2(Q_n) \leq \overline{cr}(Q_n)$. This result would be interesting since there is the Buchheim and Zheng [3] method to evaluate $\nu_2(Q_n)$, and then we would have a method to bound or determine $\overline{cr}(Q_n)$. In particular, such a relation would imply that our rectilinear drawings of R_5 and R_6 are optimal.

References

1. Ábrego, B., Aichholzer, O., Fernández-Merchant, S., Ramos, P., Salazar, G.: The 2-page crossing number of K_n. In: Proc. 2012 Symposium on Computational Geometry, SoCG 2012, pp. 397–404. ACM (2012)
2. Balogh, J., Salazar, G.: k-sets, convex quadrilaterals, and the rectilinear crossing number of K_n. Discrete Comput. Geom. 35(4), 671–690 (2006)
3. Buchheim, C., Zheng, L.: Fixed linear crossing minimization by reduction to the maximum cut problem. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 507–516. Springer, Heidelberg (2006)
4. Chung, F.R.K., Leighton, F.T., Rosenberg, A.L.: Embedding graphs in books: A layout problem with applications to VLSI design. SIAM J. Algebraic Discrete Methods 8, 33–58 (1987)
5. Dean, A.M., Richter, R.B.: The crossing number of $C_4 \times C_4$. J. Graph Theory 19, 125–129 (1995)
6. de Klerk, E., Pasechnik, D.: Improved lower bounds for the 2-page crossing numbers of $K_{m,n}$ and K_n via semidefinite programming. SIAM J. Optim. 22(2), 581–595 (2012)
7. Erdős, P., Guy, R.K.: Crossing number problems. Amer. Math. Monthly 80, 52–58 (1973)
8. Faria, L., de Figueiredo, C.M.H., Sýkora, O., Vro, I.: An improved upper bound on the crossing number of the hypercube. In: Bodlaender, H.L. (ed.) WG 2003. LNCS, vol. 2880, pp. 230–236. Springer, Heidelberg (2003)
9. Faria, L., de Figueiredo, C.M.H., Sýkora, O., Vrto, I.: An improved upper bound on the crossing number of the n-cube. J. Graph Theory 59, 145–161 (2008)
10. Garey, M.R., Johnson, D.S.: Crossing number is NP-complete. SIAM J. Algebraic and Discrete Methods 4, 312–316 (1983)

11. Madej, T.: Bounds for the crossing number of the n-cube. J. Graph Theory 15, 81–97 (1991)
12. Masuda, S., Nakajima, K., Kashiwabara, T., Fujisawa, T.: Crossing minimization in linear embeddings of graphs. IEEE Trans. Comput. 39(1), 124–127 (1990)
13. Yannakakis, M.: Embedding planar graphs in four pages. In: 18th Annual ACM Symposium on Theory of Computing, Berkeley, CA (1986), J. Comput. System Sci. 38(1), 36–67 (1989)

FPT Is Characterized by Useful Obstruction Sets[*]

Michael R. Fellows[1] and Bart M.P. Jansen[2]

[1] Charles Darwin University, Australia
Michael.Fellows@cdu.edu.au
[2] Utrecht University, The Netherlands
B.M.P.Jansen@uu.nl

Abstract. Many graph problems were first shown to be fixed-parameter tractable using the results of Robertson and Seymour on graph minors. We show that the combination of finite, computable, obstruction sets and efficient order tests is not just one way of obtaining strongly uniform FPT algorithms, but that *all* of FPT may be captured in this way. Our new characterization of FPT has a strong connection to the theory of kernelization, as we prove that problems with polynomial kernels can be characterized by obstruction sets whose elements have polynomial size. Consequently we investigate the interplay between the sizes of problem kernels and the sizes of the elements of such obstruction sets, obtaining several examples of how results in one area yield new insights in the other. We show how exponential-size minor-minimal obstructions for pathwidth k form the crucial ingredient in a novel OR-cross-composition for k-PATHWIDTH, complementing the trivial AND-composition that is known for this problem. In the other direction, we show that OR-cross-compositions into a parameterized problem can be used to rule out the existence of efficiently generated quasi-orders on its instances that characterize the NO-instances by polynomial-size obstructions.

1 Introduction

This paper is concerned with the connection between fixed-parameter tractability, kernelization, and the characterization of parameterized problems by efficiently testable obstruction sets. Historically, this connection has been a major impetus to the development of the field of parameterized complexity. The results of the Graph Minors project were applied to obtain some of the first classifications [10] of problems as (nonuniformly) fixed-parameter tractable. Robertson and Seymour proved that the set of unlabeled finite graphs is well-quasi-ordered by the minor relation [21]. By standard well-quasi-order theory, this implies that any set of graphs $\mathcal{F}$ that is closed under taking minors (a *lower ideal in the minor order*) is characterized by a *finite* obstruction set $\mathbb{O}_{\mathcal{F}}$ in the following sense: a graph is contained in $\mathcal{F}$ if and only if it does not contain an element of $\mathbb{O}_{\mathcal{F}}$

[*] This work was supported by the Netherlands Organization for Scientific Research (NWO), project "KERNELS: Combinatorial Analysis of Data Reduction".

as a minor. They also provided an algorithm for each fixed graph H that tests, given a graph G, whether H is a minor of G in $\mathcal{O}(n^3)$ time [20].

The algorithmic implications of this machinery are well known. Consider a parameterized graph problem $\mathcal{Q}$ whose input consists of a graph G and integer k. Assume that $\mathcal{Q}$ is minor-closed, i.e., that (G', k) is a YES-instance whenever (G, k) is a YES-instance and G' is a minor of G. As the YES-instances of a fixed parameter value k form a minor ideal, there is a finite obstruction set $\mathbb{O}_k$ that characterizes the ideal. Thus we can decide whether $(G, k) \in \mathcal{Q}$ by testing for each graph in $\mathbb{O}_k$ whether it is a minor of G, thereby solving $\mathcal{Q}$ in $\mathcal{O}(n^3)$ time for each fixed k. By deriving an algorithm to compute the obstruction sets $\mathbb{O}_k$, this approach yields constructive, uniform FPT algorithms (cf. [8, §7.9.2]).

Our first result in this paper shows that the described tools for developing FPT algorithms — efficient order tests for quasi-orders that characterize the YES-instances of a fixed parameter value by finite obstructions sets — are not just *one* way of obtaining (strongly uniform) FPT characterizations, but that in fact all of FPT can be characterized in this way. For this general result we relax from the minor order and instead consider arbitrary quasi-orders on the set of instances $\Sigma^* \times \mathbb{N}$ of a parameterized problem (see Section 2 for definitions).

We introduce some terminology to state the characterization. A quasi-order is a reflexive and transitive binary relation $\preceq$ on a set S. For elements $x, y \in S$ such that $x \preceq y$ we say that x *precedes* y. If x precedes y and $x \neq y$ then x *strictly precedes* y, denoted $x \prec y$. A quasi-order $\preceq$ is *polynomial-time* if there is an algorithm that decides whether $x \preceq y$ in $\mathcal{O}((|x| + |y|)^{\mathcal{O}(1)})$ time. If S is a subset of a universe U and $\preceq$ is a quasi-order on U, then S is a *lower ideal* of U if $x \in S$ and $x' \preceq x$ together imply that $x' \in S$. Our characterization extends the folklore result stating that all problems in FPT have kernels.

Theorem 1. *For any parameterized problem $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$, the following statements are equivalent:*

1. *Problem $\mathcal{Q}$ is strongly uniformly fixed-parameter tractable.*
2. *Problem $\mathcal{Q}$ is decidable and admits a kernel whose size is computable.*
3. *Problem $\mathcal{Q}$ is decidable and there is a polynomial-time quasi-order $\preceq$ on $\Sigma^* \times \mathbb{N}$ and a computable function $f\colon \mathbb{N} \to \mathbb{N}$ such that:*
   a. *The set $\mathcal{Q}$ is a lower ideal of $\Sigma^* \times \mathbb{N}$ under $\preceq$.*
   b. *For every $(x, k) \notin \mathcal{Q}$ there is an* obstruction *$(x', k') \notin \mathcal{Q}$ of size at most $f(k)$ with $(x', k') \preceq (x, k)$.*

Let us make some remarks about the theorem. Criterion (3.b) is stated in terms of small obstructions rather than finite, computable obstruction sets, to make the subsequent theorem that proves the *non-existence* of such quasi-orders (Theorem 4) stronger. The existence of computable obstruction sets follows directly from the given conditions, as will indeed be exploited in the proof of Theorem 1 in Section 3. The proof also shows that problems with kernels of size $\mathcal{O}(f(k))$ are characterized by obstructions of size $\mathcal{O}(f(k))$ under polynomial-time quasi-orders. Hence problems with polynomial kernels can be characterized by polynomial-size obstructions. This general quantitative connection between kernel sizes and obstruction sizes leads us to investigate the relationship between the two in more concrete settings. While a construction due to Kratsch and Wahlström [18] shows that it is unlikely that all problems characterized by polynomial-size obstructions have polynomial kernels, there is a rich interaction between the two domains, which forms the topic of the remainder of this work.

A Cross-Composition Based on Large Obstructions. In Section 4 we give an example of how properties of obstruction sets can be exploited to obtain kernel bounds. Our example concerns the k-PATHWIDTH problem, which asks whether the pathwidth of a given graph G is at most k. For any sequence of graphs $G_1, \ldots, G_t$, the disjoint union $G_1 \dot{\cup} G_2 \dot{\cup} \ldots \dot{\cup} G_t$ has pathwidth at most k, if and only if each G_i has pathwidth at most k. Hence there is a trivial AND-composition [2] for k-PATHWIDTH. Using existing methods [2,9] this proves that k-PATHWIDTH does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly.

The majority of kernelization lower bounds currently known, however, are not obtained by AND-composition but by OR-(cross-)-composition [2,3]: polynomial-time algorithms that take a sequence of instances as input, and output a single instance of bounded parameter value whose answer is YES if and only if *at least one* (rather than all) of the inputs are YES-instances. Given the nature of the pathwidth problem, it seems to lend itself much better to AND-composition than to OR-(cross-)-composition. However, we show that an OR-cross-composition into k-PATHWIDTH can be obtained by embedding instances of a related problem into a minor-obstruction for pathwidth k containing $\Theta(3^k)$ vertices. The properties of obstructions are exploited to ensure the correctness of this construction. The fact that the size of the obstruction is exponential in k, is crucial to obtaining this superpolynomial kernelization lower bound. The construction illustrates how properties of obstruction sets can be used to obtain kernelization bounds.

Bounds on Obstruction Sizes by Cross-Composition. We study how kernel bounds may be used to derive properties of obstruction sets in Section 5. The OR-cross-composition framework for kernelization lower bounds turns out to have interesting connections to obstruction sizes. We introduce the notion of an *efficiently generated* quasi-order, which, roughly speaking, is a quasi-order such that the elements preceding a given instance (x, k) can appear on the output paths of a polynomial-time nondeterministic Turing machine. If there is an efficiently generated quasi-order on the instances of a parameterized problem, such that each NO-instance (x, k) is preceded by a NO-instance of size $f(k)$ (an obstruction), then this results in a nondeterministic form of kernel, of size $f(k)$. As an OR-cross-composition together with a polynomial kernel implies that NP $\subseteq$ coNP/poly [3], even in the nondeterministic setting [17], this gives us the means to prove that certain parameterized problems are unlikely to be characterized by efficiently generated quasi-orders with polynomial-size obstructions. Using our OR-cross-composition for k-PATHWIDTH we can conclude that obstructions to k-PATHWIDTH are not only of superpolynomial size in the minor order, but must be of superpolynomial size for all efficiently generated quasi-orders under which k-PATHWIDTH is closed. Other examples of the connection between kernels and obstructions are discussed in Section 6.

Related Work. There are many alternative characterizations of FPT, as described for example by Flum and Grohe [12, §1.6]. Obstruction sets form a popular topic of study (e.g., [6,7,15,22,23]). The task of computing obstruction sets has also been investigated thoroughly (e.g., [4,11,19]). Dinneen [5, Theorem 5] related properties of obstruction sets to complexity-theoretic assumptions. He showed that the number of elements in obstruction sets corresponding to NP-hard minor-closed graph problems with parameter k cannot be polynomial in k, unless NP $\subseteq$ coNP/poly.

2 Preliminaries

Parameterized Complexity and Kernels. A parameterized problem $\mathcal{Q}$ is a subset of $\Sigma^* \times \mathbb{N}$, the second component being the *parameter*. For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$ we define the *size* of (x, k) to be $|(x, k)| := |x| + k$. A parameterized problem is (strongly uniformly) *fixed-parameter tractable* if there exists an algorithm to decide whether $(x, k) \in \mathcal{Q}$ in time $f(k)|x|^{\mathcal{O}(1)}$ where f is a computable function. A *kernelization algorithm* (or *kernel*) of size $f \colon \mathbb{N} \to \mathbb{N}$ for a parameterized problem $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ is a polynomial-time algorithm that, on input $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs an instance (x', k') of size at most $f(k)$ such that $(x, k) \in \mathcal{Q} \Leftrightarrow (x', k') \in \mathcal{Q}$. If $f(k) \in \mathcal{O}(k^{\mathcal{O}(1)})$ then this is a *polynomial kernel* (cf. [1]). We refer to the textbooks [8,12] for more background on parameterized complexity. The set $\{1, 2, \ldots, n\}$ is abbreviated as $[n]$.

Cross-Composition. We use the framework of cross-composition to prove kernel lower bounds, including the definition of a *polynomial equivalence relation* and a *cross-composition* as given by Bodlaender et al. [3]. To highlight the differences between OR and AND compositions, we call the type of cross-composition defined by Bodlaender et al. [3] OR-*cross-composition.*

Theorem 2 ([3]). *If a set $\mathcal{L} \subseteq \Sigma^*$ is NP-hard under Karp reductions and $\mathcal{L}$* OR-*cross-composes into the parameterized problem $\mathcal{Q}$, then there is no polynomial kernel for $\mathcal{Q}$ unless NP $\subseteq$ coNP/poly.*

Theorem 2 has been extended to the co-nondeterministic setting in recent publications. Kratsch et al. [17, Theorem 2] exploited the fact that the lower bound machinery also works if the cross-composition is co-nondeterministic.

Graphs. All graphs we consider are finite, simple, and undirected. An undirected graph G consists of a vertex set $V(G)$ and an edge set $E(G)$, whose members are 2-element subsets of $V(G)$. We write $G \subseteq H$ if graph G is a subgraph of graph H. The *clique number* $\omega(G)$ of G is the size of a largest clique in G. For a set of vertices X in a graph G we use $G - X$ to denote the graph that results after deleting all vertices of X and their incident edges. When deleting a single vertex v, we write $G - v$ rather than $G - \{v\}$. Graph H is a *minor* of graph G if H can be obtained from a subgraph of H by edge contractions. If $H \neq G$ is a minor of G, then H is a *proper minor* of G. A vertex of degree at most one is a *leaf.* A *path decomposition* of a graph G is a sequence $(\mathcal{X}_1, \ldots, \mathcal{X}_r)$ of subsets of $V(G)$, called *bags*, such that: (i) $\bigcup_{i \in [r]} \mathcal{X}_i = V(G)$, (ii) for each edge $\{u, v\} \in E(G)$ there is a bag $\mathcal{X}_i$ containing v and w, and (iii) for each $v \in V(G)$, the bags containing v are consecutive in the sequence. The *width* of a path decomposition is $\max_{1 \leq i \leq r} |\mathcal{X}_i| - 1$. The *pathwidth* of a graph G, denoted $\textsc{pw}(G)$, is the minimum width over all path decompositions of G. We say that an edge $\{u, v\}$ is *realized* by any bag that contains u and v. Condition (iii) is also called the *convexity property* of path decompositions. Proofs for statements marked by a star (★) had to be omitted from this extended abstract due to space restrictions.

3 Characterizing Problems in FPT by Small Obstructions

In this section we present the proof of Theorem 1 and consider some of its consequences.

Proof (of Theorem 1). Let $\mathcal{Q}$ be a parameterized problem. It is well-known that conditions (1) and (2) are equivalent [1, Theorem 1]. We prove that (3)⇒(1) and that (2)⇒(3).

(3)⇒(1). Consider a combination of $\preceq$ and $f\colon \mathbb{N} \to \mathbb{N}$ that satisfies the preconditions to (3). We obtain an FPT algorithm that decides $\mathcal{Q}$ by showing that there is an algorithm that computes bounded-size obstruction sets to membership in $\mathcal{Q}$. Let $k \in \mathbb{N}$ and define O_k as the NO-instances of $\mathcal{Q}$ that have size at most $f(k)$. Let $\mathbb{O}_k$ be the elements of O_k that are minimal under $\preceq$, i.e., those elements of O_k that are not preceded by another element of O_k.

Claim. Let $k \in \mathbb{N}$. For any $x \in \Sigma^*$ we have $(x, k) \in \mathcal{Q}$ if and only if there is no element in $\mathbb{O}_k$ that precedes (x, k).

Proof. Fix some $k \in \mathbb{N}$ and consider some $x \in \Sigma^*$. If (x, k) is a YES-instance then all elements that precede it under $\preceq$ are YES-instances, by (3.a). If (x, k) is a NO-instance, then by (3) there is an obstruction (x', k') of size at most $f(k)$ that is a NO-instance of $\mathcal{Q}$ and precedes (x, k). But then, using transitivity of $\preceq$, there is a minimal NO-instance with these properties, which is contained in $\mathbb{O}_k$ by definition. Hence there is an element of $\mathbb{O}_k$ that precedes (x, k). ♢

There is an algorithm that, on input $k \in \mathbb{N}$, computes the set $\mathbb{O}_k$: this follows from the facts that $\mathcal{Q}$ is decidable, f is computable, and $\preceq$ is polynomial-time. From the algorithm that computes the obstruction sets $\mathbb{O}_k$ we obtain a strongly uniformly fixed-parameter tractable algorithm for $\mathcal{Q}$, as follows. On input (x, k), compute the set $\mathbb{O}_k$. Test if there is an obstruction in $\mathbb{O}_k$ that precedes (x, k) using the order testing algorithm for $\preceq$. By the claim, the answer to (x, k) is YES if and only if there is no such preceding element. The running time is bounded by $g(k)|x|^{\mathcal{O}(1)}$ for some computable function g: the time to compute $\mathbb{O}_k$ is computable, while the $|\mathbb{O}_k|$ order tests take $\mathcal{O}((f(k) + |(x, k)|)^{\mathcal{O}(1)})$ time each.

(2)⇒(3). Let K be a kernelization algorithm for $\mathcal{Q}$ that maps instances (x, k) to equivalent instances (x', k') of size at most f, for some computable function f. We define a polynomial-time quasi-order $\preceq$ by giving an algorithm that decides, given (x, k) and (x', k'), whether $(x', k') \preceq (x, k)$. The algorithm proceeds as follows. If $(x', k') = (x, k)$ then it immediately outputs YES. Otherwise, it sets $(x^*, k^*) := (x, k)$. While $|K(x^*, k^*)| < |(x^*, k^*)|$ it replaces (x^*, k^*) by $K(x^*, k^*)$, i.e., it repeatedly applies the kernelization algorithm until this no longer decreases the total size of the instance. It then outputs YES if and only if (x', k') equals the resulting instance (x^*, k^*).

Claim. The relation $\preceq$ defined by the algorithm is a polynomial quasi-order and $\mathcal{Q}$ is a lower ideal under $\preceq$.

Proof. The number of iterations made by the algorithm on inputs (x, k) and (x', k') is bounded by $|x| + k$, as the length of the instance is decreased in each iteration. As each invocation of K takes polynomial time, the entire comparison algorithm executes in polynomial time.

It is obvious that $\preceq$ is reflexive. To prove that it is a quasi-order, it remains to prove transitivity. Consider three instances such that $(x'', k'') \preceq (x', k') \preceq (x, k)$. We shall prove that $(x'', k'') = (x', k')$ or $(x', k') = (x, k)$, which obviously implies that $(x'', k'') \preceq (x, k)$. So assume that $(x', k') \neq (x, k)$. By definition of the algorithm that decides $\preceq$, it then follows that (x', k') is the unique instance that is obtained from (x, k) by repeatedly applying the kernelization algorithm K until it no longer strictly shrinks the size of the instance. Hence for (x', k') we know that $|K(x', k')| \geq |(x', k')|$. But then any instance (x^*, k^*) with $(x^*, k^*) \preceq (x', k')$ must be identical to (x', k'), by that same definition. Thus $(x'', k'') = (x', k')$, which implies that $(x'', k'') \preceq (x, k)$. Hence $\preceq$ is transitive.

Finally let us establish that $\mathcal{Q}$ is a lower ideal of $\Sigma^* \times \mathbb{N}$ under $\preceq$. Since a kernelization maps an instance to an equivalent instance, it is easily seen that if $(x', k') \preceq (x, k)$ then $(x', k') \in \mathcal{Q} \Leftrightarrow (x, k) \in \mathcal{Q}$. Hence $(x, k) \in \mathcal{Q}$ and $(x', k') \preceq (x, k)$ together imply that $(x', k') \in \mathcal{Q}$. ◇

Claim. For every $(x, k) \notin \mathcal{Q}$ there is an *obstruction* $(x', k') \notin \mathcal{Q}$ of size at most $f(k)$ with $(x', k') \preceq (x, k)$.

Proof. Consider some $(x, k) \notin \mathcal{Q}$. Let (x', k') be the result of applying kernelization K to the instance, as long as its total size decreases by this operation. By definition of $\preceq$ we have $(x', k') \preceq (x, k)$. Since the kernelization preserves the membership status in $\mathcal{Q}$ we find that (x', k') is a NO-instance. Since K is a kernel of size $f(k)$ we have $|K(x, k)| \leq f(k)$, which implies that $|(x', k')| \leq f(k)$. ◇

The two claims show that the combination of $\preceq$ and the function f satisfy the requirements of property (3), concluding the proof. □

In the proof of Theorem 1, the size of the obstructions of (3.b) matches the size bound of the kernel from which the quasi-order $\preceq$ is derived. Hence problems with polynomial kernels can be characterized by polynomial-size obstructions.

Corollary 1. *If $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ is a decidable parameterized problem with a kernel of size $\mathcal{O}(k^c)$, then there is a polynomial-time quasi-order $\preceq$ on $\Sigma^* \times \mathbb{N}$ and a function f that together satisfy statement (3) of Theorem 1, with $f(k) \in \mathcal{O}(k^c)$.*

It follows from a construction by Kratsch and Wahlström [18] that the converse of Corollary 1 is false, assuming NP $\not\subseteq$ coNP/poly. We give a concrete example of a problem that is characterized by efficiently testable obstructions of polynomial size, yet is unlikely to admit a polynomial kernel.

3-COLORING [COMP. SIZE]
Input: An undirected graph G and an integer k that bounds the maximum size of a connected component in G.
Parameter: k.
Question: Is there a proper 3-coloring of the vertices of G?

Lemma 1 (★, Cf. [18]). 3-COLORING [COMP. SIZE] *does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly, but there is a polynomial-time quasi-order on its instances that satisfies statement (3) of Theorem 1 with $f(k) \in \mathcal{O}(k^2)$.*

4 OR-Cross-Composition into k-Pathwidth

A *minor-minimal obstruction to pathwidth k* is a graph of pathwidth $k+1$, such that all its proper minors have pathwidth $\leq k$. Minor-minimal obstructions to pathwidth k of size $\Theta(3^k)$ form the crucial ingredient for an OR-cross-composition of an NP-complete problem into k-PATHWIDTH. The following *improvement* version of the problem serves as the starting point for the composition.

PATHWIDTH IMPROVEMENT
Input: A graph G, an integer k with $2 \leq k \leq |V(G)|$, and a path decomposition $\mathcal{P}$ of G having width $k-1$.
Question: Is the pathwidth of G at most $k-2$?

Lemma 2 (★). PATHWIDTH IMPROVEMENT *is NP-complete.*

The path decomposition in the input of PATHWIDTH IMPROVEMENT makes it possible to verify in polynomial time that the pathwidth of the graph does not exceed $k-1$. The additive terms are chosen to simplify the correctness proof of the OR-cross-composition. The exponential-size obstructions to pathwidth that we need for our construction are defined as follows.

Definition 1. *For $i \in \mathbb{N}_0$, let $\mathbb{T}^i$ denote the complete ternary tree of height i with 3^i leaves. Let $\widetilde{\mathbb{T}}^i$ be the graph obtained from $\mathbb{T}^i$ by adding, for each leaf v of $\mathbb{T}^i$, a new vertex that is only adjacent to v.*

Lemma 3 (★, Cf. [15,23]). *For $k \in \mathbb{N}_0$ the graph $\widetilde{\mathbb{T}}^k$ is a minor-minimal obstruction to pathwidth k.*

In our OR-cross-composition, we need to *inflate* obstructions before being able to embed a series of input instances into them.

Definition 2. *Let G be a graph and let $k \in \mathbb{N}$. The graph $G \diamond k$, called the* inflation of G by k*, is defined as follows:*

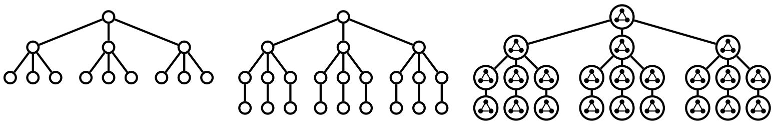

(a) The graph $\mathbb{T}^2$. (b) The graph $\widetilde{\mathbb{T}}^2$. (c) The graph $\widetilde{\mathbb{T}}^2 \diamond 3$.

Fig. 1. A ternary tree, the corresponding obstruction, and its inflation. All possible edges between connected groups of vertices are present.

– $V(G \diamond k) := \bigcup_{v \in V(G)} \{v_1, \ldots, v_k\}$.
– *Vertices u_i and v_j are adjacent in $G \diamond k$ if $u = v$ or $\{u, v\} \in E(G)$.*

For a vertex $v \in V(G)$ we call the vertices $v_1, \ldots, v_k$ in $G \diamond k$ the copies of *v.*

Refer to Fig. 1 for an example. Inflation of a graph has a straight-forward effect on its pathwidth.

Lemma 4 (★). *For any graph G and $k \in \mathbb{N}$: $\textsc{pw}(G \diamond k) + 1 = k \cdot (\textsc{pw}(G) + 1)$.*

Theorem 3 (★). *The* Pathwidth Improvement *problem* or*-cross-composes into k-*Pathwidth.

Proof (Sketch). Using a suitable choice of polynomial equivalence relation, permitted by the cross-composition framework [3], it suffices to give a polynomial-time algorithm of the following form. The input is a sequence $(G_1, k, \mathcal{P}^1), \ldots, (G_t, k, \mathcal{P}^t)$ of instances of Pathwidth Improvement that all share the same value of k, and the output is a single instance (G', k') of k-Pathwidth, with k' polynomial in $\max_{i \in [t]} |V(G_i)| + \log t$, such that $\textsc{pw}(G') \leq k'$ if and only if there is a yes-instance among the inputs. By standard arguments we may assume that t is a power of three, so let $t = 3^s$ for $s \in \mathbb{N}$.

The construction of (G', k') is based on the minor-minimal obstruction $\widetilde{\mathbb{T}}^s$. Label the $3^s = t$ leaves of $\widetilde{\mathbb{T}}^s$ as $x^1, \ldots, x^t$, and let $y^1, \ldots, y^t$ be the parents of those leaves. As $s \geq 1$ each vertex y_i has degree exactly two in $\widetilde{\mathbb{T}}^s$. We cross-compose the instances into a single graph G'. It is obtained by inflating $\widetilde{\mathbb{T}}^s$ by a factor k and replacing each k-vertex clique containing the copies of a leaf x_i by the graph G_i. More formally, we obtain G' as follows.

– Initialize G' as the inflation $\widetilde{\mathbb{T}}^s \diamond k$. For each leaf x^i the copies created by the inflation form a clique of size k on vertices $x^i_1, \ldots, x^i_k$.
– For each $i \in [t]$, remove the vertices $x^i_1, \ldots, x^i_k$ from G' and replace them by a copy of the graph G_i. Make all vertices of G_i adjacent to the copies of the parent of x^i, i.e., to the vertices $y^i_1, \ldots, y^i_k$.

Refer to Fig. 2 for an example. Let $k' := k(s+2) - 2 \in \mathcal{O}(\max_{i \in [t]} |V(G)| \cdot \log t)$.

Claim. $\textsc{pw}(G') \leq k'$ if and only if there is an $i \in [t]$ such that $\textsc{pw}(G_i) \leq k - 2$

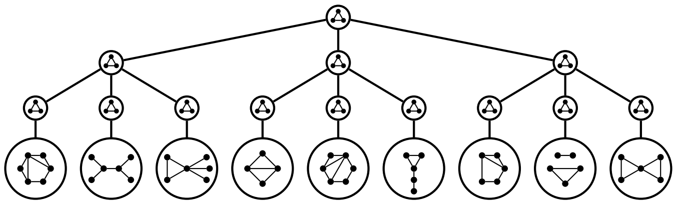


Fig. 2. Result of OR-cross-composing nine inputs with $k = 3$ into one.

The "if" direction is proven as follows. Suppose that for the instance $(G_i, k, \mathcal{P}^i)$ of Pathwidth Improvement we have $\textsc{pw}(G_i) \leq k - 2$. As $\textsc{pw}(\widetilde{\mathbb{T}}^s - x_i) = s$ by Lemma 3, the inflation satisfies $\textsc{pw}((\widetilde{\mathbb{T}}^s - x_i) \diamond k) + 1 = k \cdot (s + 1)$ by Lemma 4. From a path decomposition $\mathcal{P}'$ of $(\widetilde{\mathbb{T}}^s - x_i) \diamond k$ we obtain a path decomposition of $G' - V(G_i)$ of the same width: for each inserted instance G_j with $j \neq i$ the vertices $x^j_1, \ldots, x^j_k, y^j_1, \ldots, y^j_k$ form a clique in $\widetilde{\mathbb{T}}^s - x_i$. Hence $\mathcal{P}'$ has a bag containing all those vertices, and we may replace $x^j_1, \ldots, x^j_k$ by the width-$(k - 1)$ decomposition $\mathcal{P}^j$ of G_j without increasing the width. From a path decomposition of $G' - V(G_i)$ we obtain a decomposition of G' by inserting a width-$(k - 2)$ decomposition for G_i in the appropriate place, increasing the total width by $k - 1$ to $k \cdot (s + 1) - 1 + (k - 1) = k \cdot (s + 2) - 2 = k'$.

The "only if' direction of the claim is proven by contraposition. We use that the pathwidth of a graph equals $\min_H(\omega(H) - 1)$ over its interval supergraphs H. Suppose all inputs have pathwidth at least $k-1$, implying all interval supergraphs of the inputs have clique number at least k. An interval supergraph H' of G' contains interval supergraphs of $G_1, \ldots, G_t$. As the latter all contain a clique of size at least k, graph H' is a supergraph of $\widetilde{\mathbb{T}}^s \diamond k$. But then H' has pathwidth at least $k \cdot (s + 2) - 1$ by Lemmata 3 and 4, implying the same for G'.

The claim shows that (G', k') acts as the OR of the inputs. As it can be built in polynomial time, this concludes the proof. □

Lemma 2 and Theorem 3 provide a new way of proving that k-Pathwidth does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly, by Theorem 2.

5 Proving Nonexistence of Small Obstructions

In this section we show how the kernelization lower-bound framework of OR-cross-composition can be used to prove that a problem is *not* characterized by polynomial-size obstructions under any quasi-order of the following form.

Definition 3. *A quasi-order $\preceq$ on $\Sigma^* \times \mathbb{N}$ is* efficiently generated *if there is a polynomial-time nondeterministic Turing machine that, on input $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs an instance $(x', k') \in \Sigma^* \times \mathbb{N}$ on each computation path such that:*

- *each output instance (x', k') precedes (x, k) under $\preceq$, and*
- *all instances preceding (x, k) appear as the output of some computation path.*

Many well-known containment relations on graphs are efficiently generated. As a concrete example, consider the relation on parameterized graphs (G, k) encoded by adjacency matrices, where $(G', k) \preceq (G, k)$ if G' is a minor of G. This order is efficiently generated. A NDTM nondeterministically selects a subgraph G' of its input (G, k), then selects a set of edges to contract to obtain the minor G'', and outputs (G'', k). By using a nondeterministically selected order on the vertices when encoding G'' as an adjacency matrix, all isomorphism classes of the minor G'' are generated. The correctness of the procedure is easy to verify.

Other efficiently generated quasi-orders on parameterized graphs, encoded as adjacency matrices, include the topological minor order, the (induced) subgraph order, the immersion order, and the contraction order (cf. [8, §7.8]). The quasi-order constructed in the proof of Theorem 1 is also efficiently generated.

The following lemma, along with the notion of coNP-kernelization, could be considered folklore. Since the material never appeared in print, and has consequences for our discussion of obstruction sets, we present it here. A *coNP-kernelization algorithm* (or *coNP-kernel*) of size $f\colon \mathbb{N} \to \mathbb{N}$ for a parameterized problem $\mathcal{Q}$ is a polynomial-time nondeterministic Turing machine that, on input $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs an instance $(x', k') \in \Sigma^* \times \mathbb{N}$ of size at most $f(k)$ on each computation path, such that: (i) if $(x, k) \in \mathcal{Q}$ then all computation paths output YES-instances, and (ii) if $(x, k) \notin \mathcal{Q}$ then at least one computation path outputs a NO-instance.

Lemma 5 (★). *Let $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. If there is a polynomial $p\colon \mathbb{N} \to \mathbb{N}$ and an efficiently generated quasi-order $\preceq$ such that:*

a. $\mathcal{Q}$ is a lower ideal under $\preceq$, and
b. for any $(x, k) \notin \mathcal{Q}$ there is an obstruction *$(x', k') \notin \mathcal{Q}$ of size at most $p(k)$ with $(x', k') \preceq (x, k)$,*

then $\mathcal{Q}$ has a coNP-kernel of size $p(k) + \mathcal{O}(1)$.

The coNP-kernel is built as follows: on input (x, k), generate the elements preceding it. If a generated element has size at most $p(k)$ then output it, otherwise output a constant-size YES-instance as the result of the computation path. The following theorem follows directly from Lemma 5 together with the co-nondeterministic variant of Theorem 2 (see [17, Theorem 2]).

Theorem 4. *Let $\mathcal{L}$ be a language that is NP-hard under Karp reductions and that* OR*-cross-composes into a parameterized problem $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$. Assuming NP $\not\subseteq$ coNP/poly there is no efficiently generated quasi-order $\preceq$ on $\Sigma^* \times \mathbb{N}$ and polynomial $p\colon \mathbb{N} \to \mathbb{N}$ such that: (a) $\mathcal{Q}$ is a lower ideal under $\preceq$, and (b) for any $(x, k) \notin \mathcal{Q}$ there is an* obstruction *$(x', k') \notin \mathcal{Q}$ of size at most $p(k)$ with $(x', k') \preceq (x, k)$.*

Theorem 4 shows that an OR-cross-composition of an NP-hard set into $\mathcal{Q}$ makes it unlikely that $\mathcal{Q}$ admits an efficiently generated quasi-order on its instances that characterizes the problem by obstructions of polynomial size. The strength of the theorem comes from the fact that it excludes the existence of *efficiently generated* quasi-orders. As a *polynomial-time* quasi-order that characterizes $\mathcal{Q}$ by polynomial-size obstructions places $\mathcal{Q}$ in coNP, no NP-complete problem is characterized by polynomial-size obstructions under a polynomial-time quasi-order, unless NP = coNP.

Applying Theorem 4 to k-PATHWIDTH, we obtain some interesting information about the properties of the pathwidth measure. While it was already known that the minor-minimal obstructions to pathwidth k can have size exponential in k, Theorem 4 shows that *any* efficiently generated quasi-order under which the YES-instances are closed, must have superpolynomial size obstructions. As many natural quasi-orders on graphs are efficiently generated, this shows that a nice characterization of pathwidth in terms of polynomial-size obstructions is unlikely to exist, for *any* efficiently generated quasi-order.

6 Conclusion

The thesis underlying this paper is that the sizes of problem kernels and the sizes of obstructions in a quasi-order are intimately related, and should be studied together. We gave a general characterization of FPT in terms of problems admitting efficiently testable quasi-orders that characterize NO-instances by obstructions of bounded size. In Sections 4 and 5 we showed how properties of obstruction sets can be used to derive kernelization bounds, and vice versa. There are various other examples of the strong connection between kernel sizes and obstruction sizes in the literature. We briefly discuss three of them.

1) Obstructions to list-colorability played a crucial role in the analysis of kernels for structural parameterizations of q-COLORING by Jansen and Kratsch [14]. They proved that the existence of polynomial kernels for q-COLORING, parameterized by a vertex modulator to a graph class $\mathcal{F}$, is determined by the existence of a bound on the size of obstructions to q-list-colorability of graphs in $\mathcal{F}$.

2) Fomin et al. [13] studied the $\mathcal{F}$-DELETION problem. It asks for a fixed, finite family $\mathcal{F}$, a graph G, and an integer k, whether k vertices can be removed from G to ensure that the remainder does not contain a graph in $\mathcal{F}$ as a minor. The YES-instances of parameter value at most k form a minor ideal $\mathcal{G}_{\mathcal{F},k}$ [10, Theorem 6]. Fomin et al. proved that $\mathcal{F}$-DELETION admits a polynomial kernel for every family of connected graphs $\mathcal{F}$ that contains a planar graph. A byproduct of their kernel shows [13, Theorem 3] that for every such $\mathcal{F}$, there is a polynomial p, such that $\mathcal{G}_{\mathcal{F},k}$ is characterized by obstructions of size $p(k)$.

3) The kernelization lower bound for k-RAMSEY given by Kratsch [16], is similar in spirit to Theorem 2: it composes a sequence of instances of an NP-hard problem by embedding them in a larger host graph, whose size is superpolynomial with respect to the associated parameter value. The employed host graph is related to the Turán graph, which is extremal in Ramsey-settings.

We conclude with some directions for future research. Can Theorem 3 be adapted for k-TREEDEPTH? Can the contrapositive of Corollary 1 be used to give kernel lower bounds? Are there problems, whose kernelization complexity is still unknown, for which the nonexistence of an OR-cross-composition can be proven by the contrapositive of Theorem 4? We expect a further investigation of the interplay between kernels and obstructions to yield interesting insights.

Acknowledgments. We thank Stefan Kratsch and Rudolf Fleischer for insightful discussions, and Hans L. Bodlaender for a simple proof of Lemma 2.

References

1. Bodlaender, H.L.: Kernelization: New upper and lower bound techniques. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 17–37. Springer, Heidelberg (2009)
2. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. Syst. Sci. 75(8), 423–434 (2009)
3. Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Cross-composition: A new technique for kernelization lower bounds. In: Proc. 28th STACS, pp. 165–176 (2011)
4. Cattell, K., Dinneen, M.J., Downey, R.G., Fellows, M.R., Langston, M.A.: On computing graph minor obstruction sets. Theor. Comput. Sci. 233, 107–127 (2000)
5. Dinneen, M.J.: Too many minor order obstructions. J. UCS 3, 1199–1206 (1997)
6. Dinneen, M.J., Cattell, K., Fellows, M.R.: Forbidden minors to graphs with small feedback sets. Discrete Math. 230(1-3), 215–252 (2001)
7. Dinneen, M.J., Lai, R.: Properties of vertex cover obstructions. Discrete Math. 307(21), 2484–2500 (2007)
8. Downey, R., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer, New York (1999)
9. Drucker, A.: New limits to classical and quantum instance compression. In: Proc. 53rd FOCS, pp. 609–618 (2012)
10. Fellows, M.R., Langston, M.A.: Nonconstructive tools for proving polynomial-time decidability. J. ACM 35(3), 727–739 (1988)
11. Fellows, M.R., Langston, M.A.: An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations (extended abstract). In: Proc. 30th FOCS, pp. 520–525 (1989)
12. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer-Verlag New York, Inc. (2006)
13. Fomin, F.V., Lokshtanov, D., Misra, N., Saurabh, S.: Planar $\mathcal{F}$-Deletion: Approximation, kernelization and optimal FPT algorithms. In: Proc. 53rd FOCS, pp. 470–479 (2012)
14. Jansen, B.M.P., Kratsch, S.: Data reduction for graph coloring problems. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 90–101. Springer, Heidelberg (2011)
15. Kinnersley, N.G.: The vertex separation number of a graph equals its path-width. Inf. Process. Lett. 42(6), 345–350 (1992)
16. Kratsch, S.: Co-nondeterminism in compositions: a kernelization lower bound for a ramsey-type problem. In: Proc. 23rd SODA, pp. 114–122 (2012)
17. Kratsch, S., Pilipczuk, M., Rai, A., Raman, V.: Kernel lower bounds using co-nondeterminism: Finding induced hereditary subgraphs. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 364–375. Springer, Heidelberg (2012)
18. Kratsch, S., Wahlström, M.: The AND-conjecture may be necessary. In: Parameterized Complexity Newsletter. FPT Wiki (November 2011)

19. Lagergren, J.: Upper bounds on the size of obstructions and intertwines. J. Comb. Theory, Ser. B 73(1), 7–40 (1998)
20. Robertson, N., Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. J. Comb. Theory, Ser. B 63(1), 65–110 (1995)
21. Robertson, N., Seymour, P.D.: Graph minors. XX. Wagner's conjecture. J. Comb. Theory, Ser. B 92(2), 325–357 (2004)
22. Rué, J., Stavropoulos, K.S., Thilikos, D.M.: Outerplanar obstructions for a feedback vertex set. Eur. J. Comb. 33(5), 948–968 (2012)
23. Takahashi, A., Ueno, S., Kajitani, Y.: Minimal acyclic forbidden minors for the family of graphs with bounded path-width. Discrete Math. 127, 293–304 (1994)

Excluding Graphs as Immersions in Surface Embedded Graphs

Archontia C. Giannopoulou[1,⋆,⋆⋆], Marcin Kamiński[2], and Dimitrios M. Thilikos[3,⋆⋆⋆]

[1] Department of Informatics, University of Bergen,
P.O. Box 7803, N-5020 Bergen, Norway
`Archontia.Giannopoulou@ii.uib.no`

[2] Département d'Informatique, Université Libre de Bruxelles
and Instytut Informatyki, Uniwersytet Warszawski
`mjk@mimuw.edu.pl`

[3] Department of Mathematics,
National and Kapodistrian University of Athens
and CNRS, LIRMM
`sedthilk@thilikos.info`

Abstract. We prove a structural characterization of graphs that forbid a fixed graph H as an immersion and can be embedded in a surface of Euler genus γ. In particular, we prove that a graph G that excludes some connected graph H as an immersion and is embedded in a surface of Euler genus γ has either "small" treewidth (bounded by a function of H and γ) or "small" edge connectivity (bounded by the maximum degree of H). Using the same techniques we also prove an excluded grid theorem on bounded genus graphs for the immersion relation.

Keywords: Surface Embeddable Graphs, Immersion Relation, Treewidth, Edge Connectivity.

1 Introduction

A graph H is an immersion of a graph G if it can be obtained from G by removing vertices or edges, and splitting off adjacent pairs of edges. The class of all graphs was proved to be well-quasi-ordered under the the immersion relation by Robertson and Seymour in the last paper of their Graph Minors series [20]. Certainly, this work was

⋆ Partially supported by European Research Council (ERC) Grant "Rigorous Theory of Preprocessing", reference 267959.
⋆⋆ Partially supported by a grant of the Special Account for Research Grants of the National and Kapodistrian University of Athens (project code: 70/4/10311).
⋆⋆⋆ Co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: "Thales. Investing in knowledge society through the European Social Fund."

mostly dedicated to minors and not immersions and has been the source of many theorems regarding the structure of graphs excluding some graph H as a minor. Moreover, the minor relation has been extensively studied the past two decades and many structural results have been proven for minors with interesting algorithmic consequences (see, for example, [4, 14, 17–19, 21]). However, structural results for immersions started appearing only recently. In 2011, DeVos et al. proved that if the minimum degree of a graph G is $200t$, then G contains the complete graph on t vertices as an immersion [5]. In [7] Ferrara et al., provided a lower bound (depending on graph H) on the minimum degree of a graph G that ensures that H is contained in G as an immersion. Furthermore, Wollan recently proved a structural theorem for graphs excluding complete graphs as immersions as well as a sufficient condition such that any graph which satisfies the condition admits a wall as an immersion [22]. The result in [22] can be seen as an immersion counterpart of the grid exclusion theorem [17], stated for walls instead of grids and using an alternative graph parameter instead of treewidth.

In terms of graph colorings, Abu-Khzam and Langston in [1] provided evidence supporting the immersion ordering analog of Hadwiger's Conjecture, that is, the conjecture stating that if the chromatic number of a graph G is at least t, then G contains the complete graph on t vertices as an immersion, and proved it for $t \leq 4$. For $t = 5, 6, 7$, see [6, 15]. For algorithmic results on immersions, see [2, 10, 12, 13].

In this paper, we prove structural results for the immersion relation on graphs embeddable on a fixed surface. In particular, we show that if G is a graph that is embeddable on a surface of Euler genus γ and H is a connected graph then one of the following is true: either G has bounded treewidth (by a function that depends only on γ and H), or its edge connectivity is bounded by the maximum degree of H, or it contains H as a (strong) immersion. Furthermore, we refine our results to obtain a counterpart of the grid exclusion theorem for immersions. In particular, we prove (Theorem 3) that there exists a function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that if G is a 4-edge-connected graph embedded on a surface of Euler genus γ and the treewidth of G is at least $f(\gamma) \cdot k$, then G contains the $k \times k$-grid as an immersion. Notice that the edge connectivity requirement is necessary here as big treewidth alone is not enough to ensure the existence of a graph with a vertex of degree 4 as an immersion. Although a wall of height at least h has treewidth at least h, it does not contain the complete graph on t vertices as an immersion, for any $t \geq 5$. Finally, our results imply that when restricted to graphs of sufficiently big treewidth embeddable on a fixed surface, large edge connectivity forces the existence of a large clique as an immersion.

Our result reveals several aspects of the behavior of the immersion relation on surface embeddable graphs. The proofs exploit variants of the grid exclusion theorem for surfaces proved in [8] and [11] and the results of Biedl and Kaufmann [3] on optimal orthogonal drawings of graphs.

The paper is organized as follows. In Section 2 we give some basic definitions and preliminaries. In Section 3 we give a series of main combinatorial results. Based on the results of Section 3, we prove the main theorem and we derive its corollaries in Section 4.

Due to lack of space, the proofs of the results that are marked with $(\star)$ have been omitted.

2 Preliminaries

For every positive integer n, let $[n]$ denote the set $\{1, 2, \ldots, n\}$. A *graph* G is a pair (V, E) where V is a finite set, called the *vertex set* and denoted by $V(G)$, and E is a set of 2-subsets of V, called the *edge set* and denoted by $E(G)$. If we allow E to be a multiset then G is called a multigraph. Let G be a graph. For a vertex v, we denote by $N_G(v)$ its *(open) neighborhood*, that is, the set of vertices which are adjacent to v, and by $E_G(v)$ the set of edges containing v. Notice that if G is a multigraph $|N_G(v)| \leq |E_G(v)|$. The degree of a vertex v is $\deg_G(v) = |E_G(v)|$. We denote by $\Delta(G)$ the maximum degree over all vertices of G.

If $U \subseteq V(G)$ (respectively $u \in V(G)$ or $E \subseteq E(G)$ or $e \in E(G)$) then $G - U$ (respectively $G - u$ or $G - E$ or $G - e$) is the graph obtained from G by the removal of vertices of U (respectively of vertex u or edges of E or of the edge e). We say that a graph H is a subgraph of a graph G, denoted by $H \subseteq G$, if H can be obtained from G after deleting edges and vertices.

We say that a graph H is an *immersion* of a graph G (or H is *immersed* in G), $H \leq_{\text{im}} G$, if there is an injective mapping $f : V(H) \to V(G)$ such that, for every edge $\{u, v\}$ of H, there is a path from $f(u)$ to $f(v)$ in G and for any two distinct edges of H the corresponding paths in G are *edge-disjoint*, that is, they do not share common edges. The function f is called a *model of H in G*.

Let P be a path and $v, u \in V(P)$. We denote by $P[v, u]$ the subpath of P with endvertices v and u. Given two paths P_1 and P_2 who share a common endpoint v, we say that they are *well-arranged* if their common vertices appear in the same order in both paths.

A *tree decomposition* of a graph G is a pair (T, B), where T is a tree and B is a function that maps every vertex $v \in V(T)$ to a subset B_v of $V(G)$ such that:

(i) $\bigcup_{v \in V(T)} B_v = V(G)$,
(ii) for every edge e of G there exists a vertex t in T such that $e \subseteq B_t$, and
(iii) for every $v \in V(G)$, if $r, s \in V(T)$ and $v \in B_r \cap B_s$, then for every vertex t on the unique path between r and s in T, $v \in B_t$.

The width of a tree decomposition (T, B) is $\text{width}(T, B) := \max\{|B_v| - 1 \mid v \in V(T)\}$ and the treewidth of a graph G, denoted by $\mathbf{tw}(G)$, is the minimum over the $\text{width}(T, B)$, where (T, B) is a tree decomposition of G.

Surfaces. A *surface* Σ is a compact 2-manifold without boundary (we always consider connected surfaces). Whenever we refer to a *Σ-embedded graph G* we consider a 2-cell embedding of G in Σ. To simplify notations, we do not distinguish between a vertex of G and the point of Σ used in the drawing to represent the vertex or between an edge and the line representing it. We also consider a graph G embedded in Σ as the union of the points corresponding to its vertices and edges. That way, a subgraph H of G can be seen as a graph H, where $H \subseteq G$ in Σ. Recall that $\Delta \subseteq \Sigma$ is an open (respectively closed) disc if it is homeomorphic to $\{(x, y) : x^2 + y^2 < 1\}$ (respectively $\{(x, y) : x^2 + y^2 \leq 1\}$). The *Euler genus* of a non-orientable surface Σ is equal to the non-orientable genus $\tilde{g}(\Sigma)$ (or the crosscap number). The *Euler genus* of an orientable surface Σ is $2g(\Sigma)$, where $g(\Sigma)$ is the orientable genus of Σ. We refer to the book of Mohar and Thomassen [16] for more details on graphs embeddings. The *Euler genus* of a graph G (denoted by $\mathbf{eg}(G)$) is the minimum integer γ such that G can be embedded on a surface of the Euler genus γ.

Walls. Let k and r be positive integers where $k, r \geq 2$. The $(k \times r)$*-grid* $\Gamma_{k,r}$ is the Cartesian product of two paths of lengths $k-1$ and $r-1$ respectively. A *wall of height* k, $k \geq 1$, is the graph obtained from a $((k+1) \times (2 \cdot k + 2))$-grid with vertices (x, y), $x \in \{1, \ldots, 2 \cdot k + 2\}$, $y \in \{1, \ldots, k+1\}$, after the removal of the "vertical" edges $\{(x, y), (x, y+1)\}$ for odd $x + y$, and then the removal of all vertices of degree 1. We denote such a wall by W_k. The *corners* of the wall W_k are the vertices $c_1 = (1, 1)$, $c_2 = (2 \cdot k + 1, 1)$, $c_3 = (2 \cdot k + 1 + (k + 1 \mod 2), k + 1)$ and $c_4 = (1 + (k + 1 \mod 2), k + 1)$. (The square vertices in Figure 1.)

A *subdivided wall* W *of height* k is a wall obtained from W_k after replacing some of its edges by paths without common internal vertices. We call the resulting graph W a *subdivision* of W_k and the new vertices *subdivision vertices*. The non-subdivision vertices are called *original*. For example, in the wall of Figure 1, the black (respectively white) vertices are the original (respectively subdivision) vertices. The *perimeter* P of a subdivided wall (grid) is the cycle defined by its boundary.

Let W be a subdivided wall in a graph G and K' be the connected component of $G \setminus P$ that contains $W \setminus P$. The *compass* K of W in G is the graph $G[V(K') \cup V(P)]$. Observe that W is a subgraph of K and K is connected.

The *layers* of a subdivided wall W of height k are recursively defined as follows. The first layer of W, denoted by L_1, is its perimeter. For $i = 2, \cdots, \lceil \frac{k}{2} \rceil$, the i-th layer of W, denoted by L_i, is the $(i-1)$-th layer of the subwall W' obtained from W after removing from W its perimeter and (recursively) all occurring vertices of degree 1 (see Figure 1).

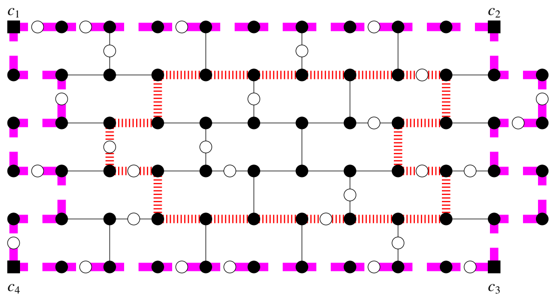


Fig. 1. The first (magenta-dashed) and second (red-dotted) layers of a wall of height 5

Given a graph G, a wall W, and a layer L of W let W' be the subwall of W with perimeter L. W' is also called the *subwall of* W *defined by* L and $C(L)$ denotes the compass of W' in G. When L is different from the perimeter of W we call the following vertices, *important* vertices of L; the original vertices of W that belong to L and have

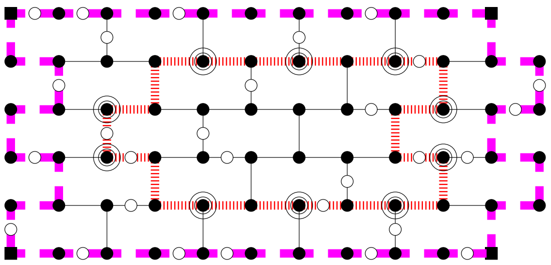


Fig. 2. The important vertices the second layer of a wall of height 5

degree 2 in the underlying non-subdivided wall of W' but are not the corners of W' (where we assume that W' shares the original vertices of W). (See Figure 2)

Observation 1. *A layer L of a wall W that is different from its perimeter and defines a subwall W' of W of height k contains exactly $4k-2$ important vertices.*

Let G be a graph that contains a subdivided wall of height k as a subgraph and let $\mathcal{W}$ be the family of all the subdivided walls of height k that are subgraphs of G. For every $W \in \mathcal{W}$ let $\mathcal{L}_w = \{L_i^W \mid i \in \lceil\frac{k}{2}\rceil\}$ be the set of all layers of W. A subdivided wall $W \in \mathcal{W}$ is called *tight* if for every $i \in \lceil\frac{k}{2}\rceil$ the graph $C(L_i^W)$ is edge-maximal, that is, there is no $W' \in \mathcal{W}$ such that for some $i \in \lceil\frac{k}{2}\rceil$, $E(C(L_i^W)) \subseteq E(C(L_i^{W'}))$ and $E(C(L_i^W)) \neq E(C(L_i^{W'}))$.

If W is a subdivided wall of height k, we call *brick* of W any facial cycle whose non-subdivided counterpart in W_k has length 6. We say that two bricks are *neighbors* if their intersection contains an edge.

Let W_k be a wall. We denote by $P_j^{(h)}$ the shortest path connecting vertices $(1, j)$ and $(2 \cdot k + 2, j)$ and call these paths the *horizontal paths of* W_k. Note that these paths are vertex-disjoint. We call the paths $P_{k+1}^{(h)}$ and $P_1^{(h)}$ the *southern path of* W_k and *northern path of* W_k respectively.

Similarly, we denote by $P_i^{(v)}$ the shortest path connecting vertices $(i, 1)$ and $(i, k+1)$ with the assumption that for, $i < 2\cdot k+2$, $P_i^{(v)}$ contains only vertices (x, y) with $x = i, i+1$. Notice that there exists a unique subfamily $\mathcal{P}_v$ of $\{P_i^{(v)} \mid i < 2\cdot k+2\}$ of $k+1$ vertical paths with one endpoint in the southern path of W_k and one in the northern path of W_k. We call these paths *vertical paths of* W_k and denote them by $P_i^{[v]}$, $i \in [k]$, where $P_1^{(v)} = P_1^{[v]}$ and $P_{2\cdot k+1}^{(v)} = P_{k+1}^{[v]}$. (See Figure 3.)

The paths $P_1^{[v]}$ and $P_{k+1}^{[v]}$ are called the *western path of* W_k and the *eastern path of* W_k respectively. Note that the perimeter of the wall can alternatively be defined as the cycle $P_1^h \cup P_{k+1}^h \cup P_1^{[v]} \cup P_{k+1}^{[v]}$.

Notice now that each vertex $u \in V(W_k) \setminus V(P)$, is contained in exactly one vertical path, denoted by $P_u^{(v)}$, and in exactly one horizontal path, denoted by $P_u^{(h)}$, of W_k. If W is

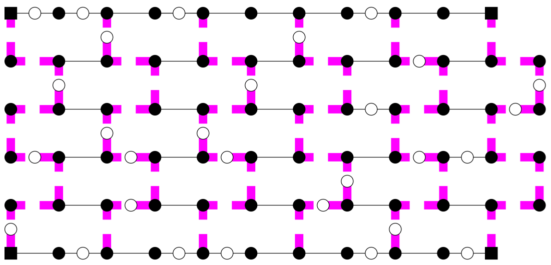


Fig. 3. The vertical paths of a wall of height 5

a subdivision of W_k, we will use the same notation for the paths obtained by the subdivisions of the corresponding paths of W_k, with further assumption that u is an original vertex of W.

From Lemma 6 in [8] and Lemma 3 in [11] we obtain the following.

Lemma 1. *Let G be a graph embedded in a surface of Euler genus γ. If $\mathbf{tw}(G) \geq 48 \cdot (\gamma + 1)^{\frac{3}{2}} \cdot (k + 5)$, G contains as a subgraph a subdivided wall of height k, whose compass in G is embedded in a closed disk Δ.*

Confluent paths. Let G be a graph embedded in some surface Σ and let $x \in V(G)$. We define a *disk around* x as any open disk Δ_x with the property that each point in $\Delta_x \cap G$ is either x or belongs to the edges incident to x. Let P_1 and P_2 be two edge-disjoint paths in G. We say that P_1 and P_2 are *confluent* if for every $x \in V(P_1) \cap V(P_2)$, that is not an endpoint of P_1 or P_2, and for every disk Δ_x around x, one of the two disks contained in Δ_x after the removal of the points that belong to P_1 does not contain any point of P_2. We also say that a collection of paths is *confluent* if the paths in it are pairwise confluent.

Moreover, given two edge-disjoint paths P_1 and P_2 in G we say that a vertex $x \in V(P_1) \cap V(P_2)$ that is not an endpoint of P_1 or P_2 is an *overlapping vertex of* P_1 *and* P_2 if there exists a Δ_x around x such that both connected components of $\Delta_x \setminus P_1$ contain points of P_2. (See, Figure 4.) For a family of paths $\mathcal{P}$, a vertex v of a path $P \in \mathcal{P}$ is called an *overlapping vertex of* $\mathcal{P}$ if there exists a path $P' \in \mathcal{P}$ such that v is an overlapping vertex of P and P'.

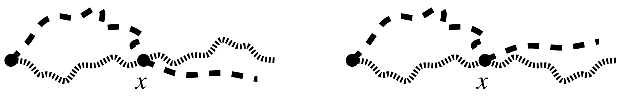


Fig. 4. The vertex x is an overlapping vertex of the two paths on the left (dashed and dotted), while it is not an overlapping vertex of the paths on the right

Orthogonal drawings. An *orthogonal drawing* of a graph G in a grid Γ is a mapping which maps vertices $v \in V(G)$ to subgrids $\Gamma(v)$ (called *boxes*) such that for every $u_1, u_2 \in V(G)$ with $u_1 \neq u_2$, $\Gamma(u_1) \cap \Gamma(u_2) = \emptyset$, and edges $\{u_1, u_2\} \in E(G)$ to (u_1', u_2')-paths whose internal vertices belong to $\Gamma - \bigcup_{v \in V(G)} \Gamma(v)$, their endpoints u_i' (called *joining vertices of $\Gamma(u_i)$*) belong to the perimeter of $\Gamma(u_i)$, $i \in [2]$, and for every two disjoint edges $e_i \in E(G)$, $i \in [2]$, the corresponding paths are edge-disjoint.

We need the following result.

Lemma 2 ([3]). *If G is a simple graph then it admits an orthogonal drawing in an $(\frac{m+n}{2} \times \frac{m+n}{2})$-grid. Furthermore, the box size of each vertex v is $\frac{\deg(v)+1}{2} \times \frac{\deg(v)+1}{2}$.*

3 Preliminary Combinatorial Lemmata

Before proving the main result of this section we first state the following lemma which we will need later on.

Lemma 3 ([9]). *Let r be a positive integer. If G is a graph embedded in a surface Σ, $v, v_1, v_2, \ldots, v_r \in V(G)$, and $\mathcal{P}$ is a collection of r edge-disjoint paths from v to $v_1, v_2, \ldots, v_r$ in G, then G contains a confluent collection $\mathcal{P}'$ of r edge-disjoint paths from v to $v_1, v_2, \ldots, v_r$ such that $E(\bigcup_{P \in \mathcal{P}'} P) \subseteq E(\bigcup_{P \in \mathcal{P}} P)$.*

Detachment tree of $\mathcal{P}$ in u. Let G be a graph embedded in a closed disk Δ, $v, v_1, v_2, \ldots, v_k$ be distinct vertices of G, and $\mathcal{P} = \{P_i \mid i \in [k]\}$ be a family of k confluent edge-disjoint paths such that P_i is a path from v to v_i, $i \in [k]$. Let also $u \in V(G) \setminus \{v, v_i \mid i \in [k]\}$ be an internal vertex of at least two paths in $\mathcal{P}$. Let $\mathcal{P}_u = \{P_{i_1}, P_{i_2}, \ldots, P_{i_r}\}$ denote the family of paths in $\mathcal{P}$ that contain u and Δ_u be a disk around u. Given any edge e with $u \in e$ we denote by u_e its common point with the boundary of Δ_u. Moreover, we denote by $e_{i_r}^1$ and $e_{i_r}^2$ the edges of P_{i_j} incident to u, $j \in [r]$.

We construct a tree T_u in the following way and call it *detachment tree of $\mathcal{P}$ in u.* Consider the outerplanar graph obtained from the boundary of Δ_u by adding the edges $\{u_{e_{i_j}^1}, u_{e_{i_j}^2}\}$, $j \in [r]$. We subdivide the edges $\{u_{e_{i_j}^1}, u_{e_{i_j}^2}\}$, $j \in [r]$, resulting to a planar graph. For every bounded face f of the graph, let $V(f)$ denote the set of vertices that belong to f. We add a vertex v_f in its interior and we make it adjacent to the vertices of $(V(f) \cap \{u_e \mid e \in u\}) \setminus \{u_{e_{i_j}^1}, u_{e_{i_j}^2} \mid j \in [r]\}$. Finally we remove the edges that lie in the boundary of Δ_u. We call this tree T_u. Notice that for every e with $u \in e$, the vertex u_e is a leaf of T_u. (See Figure 5.)

We replace u by T_u in the following way. We first subdivide every edge $e \in G$ incident to u, and denote by u_e the vertex added after the subdivision of the edge e. We denote by G_s the resulting graph. Consider now the graph $G^r = (G^s \setminus u) \cup T_u$ (where, without loss of generality, we assume that $V(G \setminus u) \cap V(T_u) = \{u_e \mid u \in e\}$). The graph G^r is called *the graph obtained from G by replacing u with T_u*. Notice here that, by construction of T_u, $u \in V(T_u)$ and thus, $u \in V(G^r)$.

Observation 2. ($\star$). *Let k, h be positive integers and G be a multigraph containing as a subgraph a subdivided wall W of height h, whose compass C is embedded in a closed disk Δ. Furthermore, let v, v_i, $i \in [k]$, be vertices of W such that there exists a*

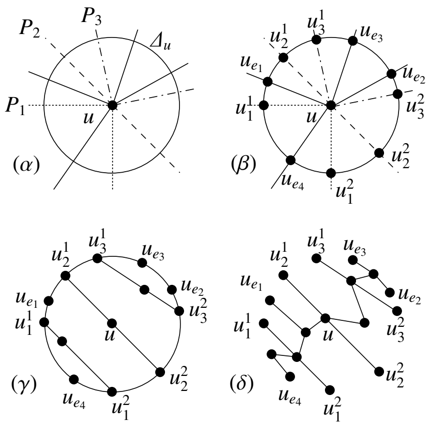


Fig. 5. Example of the construction of a detachment tree

confluent family $\mathcal{P}$ of k edge-disjoint paths from v to the vertices v_i, $i \in [k]$. Finally, let $u \in V(C) \setminus \{v, v_i \mid i \in [k]\}$ belong to more than one of the paths of $\mathcal{P}$. The graph G^r obtained from G by replacing u with T_u contains as a subgraph a subdivided wall W′ of height h, whose compass is embedded in Δ and there exists a family $\mathcal{P}'$ of k confluent edge-disjoint paths from v to v_i, $i \in [k]$, in W′ whose paths avoid u.

We now state the following auxiliary definitions. Let G be a multigraph that contains a wall of height k whose compass is embedded in a closed disk. Let $v \in A_{\lceil \frac{k}{2} \rceil}$, that is, let v be a vertex contained in the closed disk defined by the innermost layer of W, and let P be a path from v to the perimeter of W. For each layer j of the wall, $2 \leq j \leq \lceil \frac{k}{2} \rceil$, we denote by x_P^j the first vertex of P (starting from v) that also belongs to L_j and we call it *incoming vertex of P in L_j*.

We denote by P^j the maximal subpath of P that contains v and is entirely contained in the wall defined by L_j. Moreover, we denote by y_P^j its endpoint in L_j and call it *outgoing vertex of P in L_j*. Notice that x_P^j and y_P^j are not necessarily distinct vertices.

Lemma 4. ($\star$). *Let λ and k be positive integers. Let G be a graph and W be a tight subdivided wall of G of height k, whose compass is embedded in a closed disk Δ. Let also v be a vertex such that $v \in A_{\lceil \frac{k}{2} \rceil}$. If there exist λ vertex-disjoint paths P_i, $i \in [\lambda]$, from v to vertices of the perimeter then there is a brick B of W with $B \cap \overset{o}{A}_{j-1} \neq \emptyset$ that contains both $y_{P_i}^j$ and $x_{P_i}^{j-1}$.*

Lemma 5. ($\star$) *Let k be a positive integer and G be a multigraph that contains as a subgraph a subdivided wall W of height at least $4 \cdot k^2 + 1$, whose compass K is embedded in a closed disk Δ. Let also V be a set of k non-corner vertices lying in the perimeter P of W, whose mutual distance in the underlying non-subdivided wall is at least 2. If there exist a vertex $v \in A_{2 \cdot k^2 + 1}$ and k internally vertex-disjoint paths from v to vertices of P, then there exist k internally vertex-disjoint paths from v to the vertices of V in K.*

We now state the main result of this section.

Lemma 6. ($\star$) *Let k be a positive integer and G be a k-edge-connected multigraph embedded in a surface of Euler genus γ that contains a subdivided wall W of height at least $4 \cdot k^2 + 1$ as a subgraph, whose compass C is embedded in a closed disk Δ. Let also S be a set of non-corner vertices in the perimeter of W, whose mutual distance in the underlying non-subdivided wall is at least 2. If $|S| \leq k$ then there exist a vertex v in W and $|S|$ edge-disjoint paths from v to the vertices of S.*

4 Main Theorem

Given a graph G, let $\mathbf{n}(G)$ and $\mathbf{m}(G)$ denote the number of vertices and edges of G respectively. By combining Lemmata 6, 1 and 2 we obtain the following.

Theorem 1. *There exists a computable function $f : \mathbb{N} \to \mathbb{N}$ such that for every multigraph G of Euler genus γ and every connected graph H one of the following holds:*

1. $\mathbf{tw}(G) \leq f(\gamma) \cdot \lambda \cdot k$, *where* $\lambda = \Delta(H)$ *and* $k = \mathbf{m}(H)$
2. *G is not λ-edge-connected,*
3. $H \leq_{im} G$.

Proof. Let

$$f(\gamma, \lambda, k) = 48 \cdot (\gamma + 1)^{\frac{3}{2}} \cdot \left(\frac{4\,(4\lambda + 1)\,k}{2} + 5 \right),$$

and assume that $\mathbf{tw}(G) \geq f(\gamma, \lambda, k)$ and G is λ-edge-connected. From Lemma 1, we obtain that G contains as a subgraph a subdivided wall W of height $2(2\lambda + 1)k$ whose compass is embedded in a closed disk.

In what follows we will construct a model of H into the wall. From Lemma 2, H admits an a orthogonal drawing ψ in an

$$\left(\frac{\mathbf{m}(H) + \mathbf{n}(H)}{2} \times \frac{\mathbf{m}(H) + \mathbf{n}(H)}{2} \right)\text{-grid},$$

where the box size of each vertex $v \in V(H)$ is $\frac{\deg(v)+1}{2} \times \frac{\deg(v)+1}{2}$.

Notice now that ψ can be scaled to an orthogonal drawing ϕ to the grid Γ of size

$$\left(\frac{2\,(4\lambda + 1)\,(\mathbf{m}(H) + \mathbf{n}(H))}{2} + 1 \right) \times 2 \left(\frac{2\,(4\lambda + 1)\,(\mathbf{m}(H) + \mathbf{n}(H)) + 2}{2} + 1 \right),$$

where the box size of each vertex is $(4(\deg(v))^2 + 2) \times 2(4(\deg(v))^2 + 2)$, the joining vertices of each box have mutual distance at least 2 in the perimeter of the box and no joining vertex is a corner of the box.

Moreover, for every vertex u, $u \in \mathbf{Im}(\phi) \setminus \cup_{v \in V(H)} \Gamma(v)$ of degree 4, that is, for every vertex in the image of ϕ that is the intersection of two paths, there is a box in the grid of size $(4 \deg(u)^2 + 2) \times 2(4 \deg(u)^2 + 2)$, denoted by $Q(u)$, containing only this vertex and vertices of the paths it belongs to. We denote by u^i, $i \in [4]$, the vertices of $\mathbf{Im}(\phi)$

belonging to the boundary of $Q(u)$ and, for uniformity, also call them *joining vertices of* $Q(u)$.

Towards finding a model of H in the wall observe that the grid Γ contains as a subgraph a wall of height $(4\lambda + 1)(\mathbf{m}(H) + \mathbf{n}(H))$ such that each one of the boxes, either $\Gamma(v)$, $v \in V(H)$, or $Q(v)$, where v is the intersection of two paths in the image of ϕ contains a wall $W(v)$ of height $4\deg(v)^2 + 1$ and the joining vertices of $\Gamma(v)$ (the vertices v^i, $i \in [4]$, respectively) belong to the perimeter of the wall and have distance at least 2 in it. Consider now the mapping of H to W where the boxes $\Gamma(v)$ and $Q(v)$ are mapped into subwalls $W(v)$ of W of height $4\deg(v)^2 + 1$ joined together by vertex-disjoint paths as given by the orthogonal drawing ϕ. From Lemma 6, as every $W(v)$ has height $4\deg(v)^2 + 1$ and its compass is embedded in a closed disk, there exist a vertex $z_v \in V(W(v))$ and $\deg(v)$ edge-disjoint paths from z_v to the joining vertices of $W(v)$. It is now easy to see that W contains a model of H.

Notice now that in the case when $\Delta(H) = O(1)$ we get the following.

Theorem 2. *There exists a computable function $f : \mathbb{N} \to \mathbb{N}$ such that for every multigraph G of Euler genus γ and every connected graph H one of the following holds:*

1. $\mathbf{tw}(G) \leq f(\gamma) \cdot \mathbf{n}(H)$,
2. *G is not $\Delta(H)$-edge-connected,*
3. $H \leq_{im} G$.

The following two corollaries are immediate consequences of Theorems 1 and 2.

Corollary 1. *There exists a computable function $f : \mathbb{N} \to \mathbb{N}$ such that for every multigraph G of Euler genus γ and every $k \in \mathbb{N}$ one of the following holds:*

1. $\mathbf{tw}(G) \leq f(\gamma) \cdot k^3$,
2. *G is not k-edge-connected,*
3. $K_{k+1} \leq_{im} G$.

Corollary 2. *There exists a computable function $f : \mathbb{N} \to \mathbb{N}$ such that for every multigraph G of Euler genus γ and every $k \in \mathbb{N}$ one of the following holds:*

1. $\mathbf{tw}(G) \leq f(\gamma) \cdot k^2$,
2. *G is not 4-edge-connected,*
3. *$(k \times k)$-grid is an immersion of G.*

However, when H is the grid a straightforward argument gives the following result.

Theorem 3. *There exists a computable function $f : \mathbb{N} \to \mathbb{N}$ such that for every multigraph G that is embedded in a surface of Euler genus γ and every $k \in \mathbb{N}$ one of the following holds:*

1. $\mathbf{tw}(G) \leq f(\gamma) \cdot k$.
2. *G is not 4-edge-connected.*
3. *$(k \times k)$-grid is an immersion of G.*

Proof. Let

$$f(\gamma, k) = 48 \cdot (\gamma + 1)^{\frac{3}{2}} \cdot ((4^3 + 3) \cdot k + 5).$$

Assume that G is 4-edge-connected and that $\mathbf{tw}(G) \geq f(\gamma, k)$. As $\mathbf{tw}(G) \geq f(\gamma, k)$, from Lemma 1 it follows that G contains as a subgraph a subdivided wall W of height $(4^3 + 3)k$, whose compass in G is embedded in a closed disk Δ.

Consider the k^2 subwalls of W of height $(4^3 + 1)$ that occur after removing from it the paths $P^{[v]}_{(4^3+3)j}$, $P^{[h]}_{(4^3+3)j}$, $i, j \in [k]$. For every $i, j \in [k]$, we denote by $W_{(i,j)}$ the subwall that is contained inside the disk that is defined by the paths $P^{(h)}_{(4^3+3)(i-1)}$, $P^{(h)}_{(4^3+3)i}$, $P^{[v]}_{(4^3+3)(j-1)}$, and $P^{[v]}_{(4^3+3)j}$. In the case where $j = 1$ and $i = 1$, we abuse notation and consider as $P^{(h)}_{(4^3+3)(j-1)}$ and $P^{[v]}_{(4^3+3)(j-1)}$ the paths $P^{(h)}_1$ and $P^{[v]}_1$, respectively.

From Lemma 6 and the hypothesis that G is 4-edge-connected, for $k = 4$, it follows that in the compass of each one of the subwalls $\{W_{(i,j)} \mid i, j \in [k]\}$ we may find a vertex $v_{(i,j)}$ and four edge-disjoint paths from $v_{(i,j)}$ to the vertices $v^n_{(i,j)}$, $v^s_{(i,j)}$, $v^w_{(i,j)}$, and $v^e_{(i,j)}$, that lie in the northern, southern, western, and eastern path of the wall, respectively.

Finally, we consider the function $g((i, j)) = v_{(i,j)}$ that maps the vertex (i, j) of the $(k \times k)$-grid to the vertex $v_{(i,j)}$ of the wall $W_{(i,j)}$. Is now easy to see that g is an immersion model of the $(k \times k)$-grid in the compass of the wall W and the theorem follows as f is linear on k.

5 Conclusions

In this paper, we proved sufficient conditions for the containment of any connected graph H as an immersion in graphs of bounded genus. We would like to remark here that our proofs also hold if we, instead, consider the strong immersion relation where we additionally ask that the paths of the model f of H in G that correspond to the edges of H are internally disjoint from $f(V(H))$.

In our results, it appears that both big treewidth and the edge connectivity requirement are necessary in order to enforce the appearance of a graph as an immersion. A natural open problem to investigate is the existence of counterparts of our results for the case of the topological minor relation. Certainly, here edge connectivity should be replaced by vertex connectivity. However, what we can only report is that stronger conditions than just asking for sufficiently big treewidth are required for such an extension.

Acknowledgement. We would like to thank the anonymous referees for helpful comments and remarks that improved the presentation of this paper.

References

1. Abu-Khzam, F.N., Langston, M.A.: Graph coloring and the immersion order. In: Warnow, T., Zhu, B. (eds.) COCOON 2003. LNCS, vol. 2697, pp. 394–403. Springer, Heidelberg (2003)
2. Belmonte, R., van 't Hof, P., Kamiński, M., Paulusma, D., Thilikos, D.M.: Characterizing graphs of small carving-width. In: Lin, G. (ed.) COCOA 2012. LNCS, vol. 7402, pp. 360–370. Springer, Heidelberg (2012)

3. Biedl, T.C., Kaufmann, M.: Area-efficient static and incremental graph drawings. In: Burkard, R.E., Woeginger, G.J. (eds.) ESA 1997. LNCS, vol. 1284, pp. 37–52. Springer, Heidelberg (1997)
4. 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)
5. DeVos, M., Dvořák, Z., Fox, J., McDonald, J., Mohar, B., Scheide, D.: Minimum degree condition forcing complete graph immersion. ArXiv e-prints (January 2011)
6. DeVos, M., Kawarabayashi, K.-I., Mohar, B., Okamura, H.: Immersing small complete graphs. Ars Math. Contemp. 3(2), 139–146 (2010)
7. Ferrara, M., Gould, R.J., Tansey, G., Whalen, T.: On h-immersions. Journal of Graph Theory 57(3), 245–254 (2008)
8. Fomin, F.V., Golovach, P.A., Thilikos, D.M.: Contraction obstructions for treewidth. J. Comb. Theory, Ser. B 101(5), 302–314 (2011)
9. Giannopoulou, A.C., Kaminski, M., Thilikos, D.M.: Forbidding kuratowski graphs as immersions. CoRR, abs/1207.5329 (2012)
10. Giannopoulou, A.C., Salem, I., Zoros, D.: Effective computation of immersion obstructions for unions of graph classes. In: Fomin, F.V., Kaski, P. (eds.) SWAT 2012. LNCS, vol. 7357, pp. 165–176. Springer, Heidelberg (2012)
11. Giannopoulou, A.C., Thilikos, D.M.: Optimizing the graph minors weak structure theorem. CoRR, abs/1102.5762 (2011)
12. Grohe, M., Ichi Kawarabayashi, K., Marx, D., Wollan, P.: Finding topological subgraphs is fixed-parameter tractable. In: STOC, pp. 479–488 (2011)
13. Ichi Kawarabayashi, K., Kobayashi, Y.: List-coloring graphs without subdivisions and without immersions. In: Rabani, Y. (ed.) SODA, pp. 1425–1435. SIAM (2012)
14. Kostochka, A.V.: Lower bound of the hadwiger number of graphs by their average degree. Combinatorica 4(4), 307–316 (1984)
15. Lescure, F., Meyniel, H.: On a problem upon configurations contained in graphs with given chromatic number. In: Andersen, L.D., Jakobsen, I.T., Thomassen, C., Toft, B., Vestergaard, P.D. (eds.) Graph Theory in Memory of G.A. Dirac. Annals of Discrete Mathematics, vol. 41, pp. 325–331. Elsevier (1988)
16. Mohar, B., Thomassen, C.: Graphs on surfaces. Johns Hopkins University Press, Baltimore (2001)
17. Robertson, N., Seymour, P., Thomas, R.: Quickly excluding a planar graph. J. Combin. Theory Ser. B 62(2), 323–348 (1994)
18. Robertson, N., Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. J. Combin. Theory Ser. B 63(1), 65–110 (1995)
19. Robertson, N., Seymour, P.D.: Graph minors. XVI. Excluding a non-planar graph. J. Comb. Theory, Ser. B 89(1), 43–76 (2003)
20. Robertson, N., Seymour, P.D.: Graph minors XXIII. Nash-Williams' immersion conjecture. J. Comb. Theory, Ser. B 100(2), 181–205 (2010)
21. Thomason, A.: The extremal function for complete minors. J. Combin. Theory Ser. B 81(2), 318–338 (2001)
22. Wollan, P.: The structure of graphs not admitting a fixed immersion. CoRR, abs/1302.3867 (2013)

OBDD-Based Representation of Interval Graphs

Marc Gillé[*]

TU Dortmund, LS2 Informatik, Germany

Abstract. A graph $G = (V, E)$ can be described by the characteristic function of the edge set χ_E which maps a pair of binary encoded nodes to 1 iff the nodes are adjacent. Using *Ordered Binary Decision Diagrams* (OBDDs) to store χ_E can lead to a (hopefully) compact representation. Given the OBDD as an input, symbolic/implicit OBDD-based graph algorithms can solve optimization problems by mainly using functional operations, e. g., quantification or binary synthesis. While the OBDD representation size can not be small in general, it can be provable small for special graph classes and then also lead to fast algorithms. In this paper, we show that the OBDD size of unit interval graphs is $O(|V|/\log|V|)$ and the OBDD size of interval graphs is $O(|V|\log|V|)$ which both improve a known result from Nunkesser and Woelfel (2009). Furthermore, we can show that using our variable order and node labeling for interval graphs the worst-case OBDD size is $\Omega(|V|\log|V|)$. We use the structure of the adjacency matrices to prove these bounds. This method may be of independent interest and can be applied to other graph classes. We also develop a maximum matching algorithm on unit interval graphs using $O(\log|V|)$ operations and evaluate the algorithm empirically.

1 Introduction

The development of graph algorithms is a classic and intensively studied area of computer science. But the requirements on graph algorithms have changed by the emergence of massive graphs, e. g., the internet graph or social networks. There are applications, e. g., dealing with a state transition graphs in circuit verification, where even polynomial running time may not be feasible or the input does not fit into the main memory. In order to deal with such massive graphs, *implicit* graph algorithms have been investigated, where the input is represented by the characteristic function χ_E of the edge set and the nodes are encoded by binary numbers. Implicit representations can be significantly smaller than explicit representations on structured graphs. χ_E can be represented by *Ordered Binary Decision Diagrams* (OBDDs) introduced by Bryant [8] which are a commonly used data structure for Boolean functions since they support many important functional operations efficiently. A research area came up concerning the design and analysis of implicit (graph) algorithms on OBDD represented inputs ([12,13,16,25,26,27,30]). Implicit algorithms are successful in many practical applications, e. g., model checking [9], integer linear programming [18] and logic

[*] Supported by Deutsche Forschungsgemeinschaft, grant BO 2755/1-1.

minimization [11]. One of the first implicit graph algorithm was the maximum flow algorithm on 0-1-networks presented by Hachtel and Somenzi [16] which was able to solve instances up to 10^{36} edges and 10^{27} nodes in reasonable time.

The number of functional operations of an implicit algorithm is an important measure of difficulty [2]. Algorithms using a polylogarithmic number of operations were designed for instance for topological sorting [30], maximal matching [6] and minimum spanning tree [3] where a matching M, i. e., a set of edges without a common vertex, is called maximal if M is no proper subset of another matching. The actual running time depends on the OBDD sizes which are used for these operations which are hard to determine in general. So the practical performance is often evaluated experimentally, e. g., for the maximum matching problem in bipartite graphs [4] or for the maximum flow problem [16,25].

For a good running time of an implicit algorithm the size of the OBDD representing the input graph should be small. Nunkesser and Woelfel [22] investigated the OBDD size of restricted graph classes such as interval graphs. An interval graph is an intersection graph of intervals on the real line, i. e., two intervals (nodes) are adjacent iff they have a nonempty intersection. If the intervals have a length of 1, then the graph is called unit interval graph. (Unit) Interval graphs were extensively studied and have many applications, e. g., in genetics, archaeology, scheduling, and much more [15]. Nunkesser and Woelfel [22] proved that general interval graphs with N nodes can be represented by OBDDs of size $O(N^{3/2}\log^{3/4} N)$ while the OBDD size of unit interval graphs is $O(N/\sqrt{\log N})$. They also proved a lower bound of $\Omega(N)$ for general interval graphs and $\Omega(N/\log N)$ for unit interval graphs which means that the worst-case OBDD size of a graph from these classes is bounded below by these values.

As in [22], we use $n = \lceil \log N \rceil$ bits, i. e., the minimal number of bits, to encode the nodes of a graph. Since the worst-case OBDD size is exponentially large in the number of input bits, using χ_E in an implicit algorithm motivates to use a minimal amount of input bits to avoid a large worst-case OBDD size. Aiming for a good compression of χ_E, Meer and Rautenbach [19] investigated graphs with bounded clique-width or tree-width and increases the number of bits used for the node labeling to $c \cdot \log N$ with constant c and were able to improve for instance the OBDD size of cographs from $O(N \log N)$ [22] to $O(N)$.

Our Contribution. In Section 3 we present a new method to show upper and lower bounds of the size of an OBDD representing a graph. Using some known structure of the adjacency matrix of interval graphs [21], we improve the bound on general interval graphs to $O(N \log N)$ while using a more convenient way to label the nodes than in [22]. Using a probabilistic argument, we prove a lower bound of $\Omega(N \log N)$ if we use the same labeling and variable order as for our upper bound. We can also close the gap of the upper bound and the lower bound in the case of unit interval graphs and show that the OBDD size is $\Theta(N/\log N)$.

In Section 4 we present a maximum matching algorithm for unit interval graphs using only $O(\log N)$ functional operations. We were able to compute the transitive closure of a unit interval graph using only $O(\log N)$ operations instead of $O(\log^2 N)$ operations, which are needed in general. To the best of the author's knowledge, this is the first time that the labeling of nodes is used to speed up an implicit algorithm for a large graph class as unit interval graphs and to improve the number of functional operations. In order to implement the algorithm efficiently, we have to extend a known result due to Woelfel [30] to a different variable order for constructing OBDDs representing multivariate threshold functions. In Section 5 we evaluate the implicit matching algorithm experimentally and see that it is both very fast and space efficient.

A simple implicit representation of an interval graph is a list of N intervals using $\Theta(\log N)$ bits for each endpoint summing up to $\Theta(N \log N)$ space. Our results show that in the worst case the OBDD representation is almost as good as the interval representation with the advantage that it is possible to use $o(N \log N)$ space for some instances. Together with our implicit algorithm, this shows that the representation of at least unit interval graphs with OBDDs enables a good compression without loosing the usability in algorithms.

2 Preliminaries

Omitted proofs and a more detailed version of this paper can be found in [14].

OBDDs. We denote the set of Boolean functions $f : \{0,1\}^n \to \{0,1\}$ by B_n. Let $(x_0, \ldots, x_{n-1}) = x \in \{0,1\}^n$ be a binary number of length n and $|x| := \sum_{i=0}^{n-1} x_i \cdot 2^i$ the value of x. Further, let $l \in \mathbb{N}$ be a natural number then we denote by $[l]_2$ the corresponding binary number of l, i.e., $|[l]_2| = l$. Let $G = (V, E)$ be a directed graph with node set $V = \{v_0, \ldots, v_{N-1}\}$ and edge set $E \subseteq V \times V$. Here, an undirected graph is interpreted as a directed symmetric graph. Implicit algorithms are working on the characteristic function $\chi_E \in B_{2n}$ of E where $n = \lceil \log N \rceil$ is the number of bits needed to encode a node of V and $\chi_E(x, y) = 1$ if and only if $(v_{|x|}, v_{|y|}) \in E$. In order to deal with Boolean functions, OBDDs were introduced by Bryant [8] to get a compact representation, which supports a bunch of functional operations efficiently.

Definition 1 (Ordered Binary Decision Diagram (OBDD)).
Order. *A* variable order *π on the input variables $X = \{x_0, \ldots, x_{n-1}\}$ of a Boolean function $f \in B_n$ is a permutation of the index set $I = \{0, \ldots, n-1\}$.*

Representation. *A π-*OBDD *is a directed, acyclic and rooted graph G with two sinks labeled by the constants* 0 *and* 1*. Each inner node is labeled by an input variable from X and has exactly two outgoing edges labeled by* 0 *and* 1*. Each edge (x_i, x_j) has to respect the variable order π, i. e., $\pi(i) < \pi(j)$.*

Evaluation. *An assignment $a \in \{0,1\}^n$ of the variables defines a path from the root to a sink by leaving each x_i-node via the a_i-edge. A π-OBDD G_f represents f iff for every $a \in \{0,1\}^n$ the defined path ends in a sink with label $f(a)$.*

Complexity. *The* size *of a π-OBDD G, denoted by $|G|$, is the number of nodes in G. The π-OBDD size of a function f is the minimum size of a π-OBDD representing f. The OBDD size of f is the minimum π-OBDD size over all variable orders π. The* width *of G is the maximum number of nodes labeled by the same input variable.*

In the following we describe some important operations on Boolean functions which we will use in this paper (see, e. g., Section 3.3 in [29] for a detailed list). Let f and g be Boolean functions in B_n on the variable set $X = \{x_0, \ldots, x_{n-1}\}$ and G_f and G_g be OBDDs representing f and g which are also the inputs for the operations. The *negation* $\overline{f} \in B_n$ of f can be computed in time $O(1)$, i. e., given the OBDD G_f it is possible to compute the OBDD $G_{\overline{f}}$ in $O(1)$ time. Let $i \in \{0, \ldots, n-1\}$ be an index and a $c_i \in \{0,1\}$. The *replacement by constant*, i. e., the subfunction $f_{|x_i=c_i}$ can be computed in time $O(|G_f|)$. Let $\otimes \in B_2$ be a binary Boolean operation. The *synthesis* of f and g w.r.t. $\otimes$, i. e., the function $h \in B_n$ with $h := f \otimes g$, can be computed in time $O(|G_f| \cdot |G_g|)$. Finally, the *quantification* $h := Qx_i : f$ of f with quantifier $Q \in \{\exists, \forall\}$ is defined by $\exists x_i : f := f_{|x_i=0} \vee f_{|x_i=1}$ and $\forall x_i : f := f_{|x_i=0} \wedge f_{|x_i=1}$. The time needed to compute the quantification is determined by the computation time of two replacements by constant and one synthesis. In the rest of the paper quantifications over k Boolean variables $Qx_1, \ldots, x_k : f$ are denoted by $Qx : f$, where $x = (x_1, \ldots, x_k)$.

In implicit graph algorithms, the following operation (see, e. g., [27]) is useful to reverse the edges of a given graph.

Definition 2. *Let $k \in \mathbb{N}$, ρ be a permutation of $\{1, \ldots, k\}$ and $f \in B_{kn}$ with input vectors $x^{(1)}, \ldots, x^{(k)} \in \{0,1\}^n$. The argument reordering $\mathcal{R}_\rho(f) \in B_{kn}$ with respect to ρ is defined by $\mathcal{R}_\rho(f)(x^{(1)}, \ldots, x^{(k)}) := f(x^{(\rho(1))}, \ldots, x^{(\rho(k))})$.*

This operation can be computed by just renaming the variables and repairing the variable order using $3(k-1)n$ functional operations (see [5]).

An important variable order is the interleaved variable order which is defined on vectors of length n where the variables with the same significance are tested one after another.

Definition 3. *Let $x^{(1)}, \ldots, x^{(k)} \in \{0,1\}^n$ be input vectors and π be a permutation of $\{0, \ldots, n-1\}$. Then $\pi_{k,n} = (x^{(1)}_{\pi(0)}, x^{(2)}_{\pi(0)}, \ldots, x^{(k)}_{\pi(0)}, \ldots, x^{(1)}_{\pi(n-1)}, \ldots, x^{(k)}_{\pi(n-1)})$ is called k-*interleaved variable order for $x^{(1)}, \ldots, x^{(k)}$. If $\pi = (n-1, \ldots, 0)$ then we say that the variables are tested with decreasing significance.*

An OBDD-based graph algorithm computes an output χ_O represented as an OBDD given a characteristic function χ_E as an input by mainly using functional operations. The running time depends on the actual size of the OBDDs used for the operations during the computation which is difficult to bound in general.

However, if the input OBDD size representing a graph is large, any algorithm using this OBDD is likely to have an inadequate running time. Beside the variable order, the labeling of the nodes is another optimization parameter with huge influence on the input size. For OBDDs representing state transitions in finite state machines, Meinel and Theobald [20] showed that there can be an exponential blowup of the OBDD size from a good labeling to a worst-case labeling. Nevertheless, a small input OBDD size does not guarantee a good running time since the sizes of the intermediate OBDDs do not have to be small. Indeed, an exponential blowup from input to output size is possible [27,3].

We denote by $f_{|x_{\pi(0)}=a_{\pi(0)},\ldots,x_{\pi(i-1)}=a_{\pi(i-1)}}$ the subfunction where $x_{\pi(j)}$ is replaced by the constant $a_{\pi(j)}$ for $0 \leq j \leq i-1$. The function f *depends essentially* on a variable x_i iff $f_{|x_i=0} \neq f_{|x_i=1}$. A characterization of minimal π-OBDDs due to Sieling and Wegener [28] can be often used to bound the OBDD size.

Theorem 1 ([28]). *Let $f \in B_n$ and for all $i = 0, \ldots, n-1$ let s_i be the number of different subfunctions which result from replacing all variables $x_{\pi(j)}$ with $0 \leq j \leq i-1$ by constants and which essentially depend on $x_{\pi(i)}$. Then the minimal π-OBDD representing f has s_i nodes labeled by $x_{\pi(i)}$.*

Basic Functions and Implicit Algorithms. The OBDD size of the equality $EQ(x,y)$ and greater than function $GT(x,y)$ with $EQ(x,y) = 1 \Leftrightarrow |x| = |y|$ and $GT(x,y) = 1 \Leftrightarrow |x| > |y|$ is linear in the number of input bits for an interleaved variable order (see, e. g., [29]). For the sake of code readability, we use $|x| = |y|$ and $|x| > |y|$ to denote $EQ(x,y)$ and $GT(x,y)$ in our algorithms. Furthermore, by $|x| > c$ ($|x| = c$) for some constant c we denote the function $GT(x,y)_{|y=[c]_2}$ ($EQ(x,y)_{|y=[c]_2}$). Every function $R(x,y) \in B_{2n}$ can be seen as a binary relation R on the set $\{0,1\}^n$ with $x\,R\,y \Leftrightarrow R(x,y) = 1$. The transitive closure of $R(x,y)$ can be computed implicitly by $O(n^2)$ functional operations using the so called iterative squaring or path doubling technique (see, e. g., [14]).

Interval Graphs. Let $\mathcal{I} = \{[a_i, b_i] \mid a_i < b_i \text{ and } 0 \leq i \leq N-1\}$ be a set of N intervals on the real line. The *interval graph* $G_\mathcal{I} = (V, E)$ has one node for each interval in $\mathcal{I}$ and two nodes $v \neq w$ are adjacent iff the corresponding intervals intersect. If no interval is properly contained in another interval, $G_\mathcal{I}$ is called *proper interval graph.* If the length of every interval in $\mathcal{I}$ is equal to 1 then $G_\mathcal{I}$ is called *unit interval graph.* Notice that the set of all interval graphs does not change if we restrict ourselves to sets $\mathcal{I}$ where all endpoints are different. The definitions of proper and unit interval graphs are equivalent in the sense that they generate the same class of interval graphs [23]. Hence, in the following we only use the term of unit interval graphs. An undirected graph H is a (unit) interval graph iff there is a set of (unit) intervals $\mathcal{I}$ such that $H = G_\mathcal{I}$. Due to the one-to-one correspondence of the nodes of $G_\mathcal{I}$ and the elements of $\mathcal{I}$, we use the notion of node and interval synonymously.

3 OBDD Size of Interval Graphs

We present a way to count the subfunctions of the characteristic function χ_E of the edge set of a graph using the adjacency matrix of the graph which can give us a more graph theoretic approach to subfunctions.

The rows (columns) of an adjacency matrix correspond to the x-variables (y-variables) of $\chi_E(x,y)$. We can sort the rows of the adjacency matrix according to a variable order π by connecting the i-th row to the input x with $\sum_{l=0}^{n-1} x_{\pi(n-l-1)} \cdot 2^l = i$, i. e., we let the l-th x-variable in π have significance 2^{n-l-1} to sort the rows. This can be done analogously to sort the columns. Thus, the variable order π defines a permutation of the rows and columns of the adjacency matrix resulting in a new matrix which we call π-ordered adjacency matrix.

Definition 4. *Let $G = (V, E)$ be a graph and $\pi := \pi_{2,n}$ be a 2-interleaved variable order for the characteristic function $f := \chi_E$. The* π-ordered *adjacency matrix A_π of G is defined as follows: $a_{ij} = 1$ iff $f(x, y) = 1$ with $\sum_{l=0}^{n-1} x_{\pi(n-l-1)} \cdot 2^l = i$ and $\sum_{l=0}^{n-1} y_{\pi(n-l-1)} \cdot 2^l = j$.*

Notice that the π-ordered adjacency matrix is equal to the "normal" adjacency matrix where the rows and columns are sorted by the node labels iff the variables in π are tested with decreasing significance. The π-ordered adjacency matrix gives us a visualization of the subfunctions in terms of *blocks* of the matrix.

Definition 5. *Let $n \in \mathbb{N}$ and A be a $2^n \times 2^n$ matrix. For $0 \le k \le n$ and $0 \le i, j \le 2^k - 1$ the* block B_{ij}^k *of A is defined by the quadratic submatrix of size $2^n/2^k \times 2^n/2^k$ which is formed by the intersection of the rows $i \cdot 2^n/2^k, \ldots, (i + 1) \cdot 2^n/2^k - 1$ and the columns $j \cdot 2^n/2^k, \ldots, (j + 1) \cdot 2^n/2^k - 1$.*

Let i be even. Then the block $B_{|a|,|b|}^{i/2}$ represents the function table of the subfunction which results from replacing the first $i/2$ x-variables w.r.t π by $a \in \{0, 1\}^{i/2}$ and the first $i/2$ y-variables by $b \in \{0, 1\}^{i/2}$. Thus, counting the number of different blocks $B_{|a|,|b|}^{i/2}$ is equivalent to counting the number of different subfunctions.

Bollig and Wegener [7] use a similar approach to visualize subfunctions of a storage access function by building a matrix whose columns and rows are sorted according to the variable order and correspond to variables (not assignments as in our π-ordered matrix). Notice that A_π is not the communication matrix which is often used to show lower bounds of the OBDD size.

Theorem 2. *Let $\pi := \pi_{2,n}$ be the interleaved variable order with decreasing significance and $G = (V, E)$ be an interval graph with $N := |V|$ nodes. The π-OBDD size of χ_E can be bounded above by $O(N \log N)$.*

Proof. Let $f := \chi_E$, $1 \le k \le n$ and s_k be the number of different subfunctions $f_{|\alpha,\beta}$ of f where $\alpha \in \{0, 1\}^k$ is an assignment to the variables $x_{n-1}, \ldots, x_{n-k}$ and $\beta \in \{0, 1\}^k$ is an assignment to the variables $y_{n-1}, \ldots, y_{n-k}$, respectively. It is enough to bound s_k by above because replacing an additional variable by a constant can at most double the number of subfunctions.

We label the nodes according to their position in the sorted sequence of interval starting points. Recall that $a_{i,j}$ is one if and only if interval i intersects interval j. Now, notice that if $a_{j,i}$ is zero for $j > i$ then no interval $j' > j$ with a larger starting point can cut interval i. Thus, for every column $i \in \{0, \ldots, N-1\}$, the sequence $(a_{i+1,i}, \ldots, a_{N-1,i})$ is zero or starts with a continuous sequence of ones followed by only zeros (see also [21]).

Every subfunction $f_{|\alpha,\beta}$ corresponds to a block $B_{|\alpha|,|\beta|}^k$. Let $\beta = 0^k$ and $|\alpha| \ge 1$, i. e., we consider the blocks $B_{|\alpha|,0}^k$ of size $2^{n-k} \times 2^{n-k}$ (see Fig. 1). As we observed, every column of A_π has (below the diagonal) at most one possible *changing* position k such that $a_{k,i} = 1$ and $a_{k+1,i} = 0$. Looking at the sequence $(B_{1,0}^k, \ldots, B_{2^k-1,0}^k)$ of blocks, this fact implies that a block $B_{i,0}^k$ can only form a new block, i. e., all previous blocks in the sequence are different to this block, if

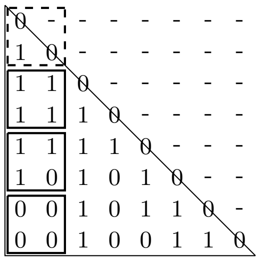


Fig. 1. Possible adjacency matrix with 8 nodes and framed subfunctions $f_{|\alpha,\beta}$ with $\beta = 0^k$, $|\alpha| \geq 1$, and $k = 2$.

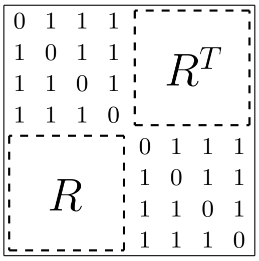


Fig. 2. Random interval graph where only R is generated randomly

there is a changing position in one column inside of $B^k_{i,0}$ or inside the block $B^k_{i-1,0}$ or between these two blocks. Therefore, every changing position can induce at most two different blocks and we can bound the number of different blocks by two times the number of possible changing positions which is at most the number of columns of a block, i. e., $2 \cdot 2^{n-k}$. Since the graph is symmetric and there are 2^k blocks on the diagonal, we can bound the overall number of different blocks by $O(2^{n-k} \cdot 2^k + 2^k) = O(2^n)$ and thus $s_k = O(2^n)$. Summing this up over all possible values of k, the π-OBDD size is at most $O(2^n \cdot n) = O(N \log N)$. $\square$

In [22] it is proved that the OBDD size of unit interval graphs is $\Omega(N/\log N)$ and $O(N/\sqrt{\log N})$ which we can improve by using the π-ordered adjacency matrix.

Theorem 3. *Let π be the interleaved variable order with decreasing significance. The π-OBDD size of χ_E for a unit interval graph $G = (V, E)$ is $O(N/\log N)$.*

The difference between unit and general interval graphs is that in general interval graphs there is no dependence between the columns of the π-ordered adjacency matrix, which is important for our lower bound, while in unit interval graphs, the row number of the last 1 entry in a column is increasing from left to right.

The upper bound proof suggests that the number of blocks $B^k_{i,j}$ with a changing position roughly determines the number of x_{n-k-1}-nodes of the OBDD. However, explicitly constructing a worst-case interval graph with OBDD size of $\Omega(N \log N)$ is difficult because there are many dependencies between blocks for different values of k, since a block $B^k_{i,j}$ results from dividing some block $B^{k-1}_{i',j'}$.

In order to overcome these dependencies, we look at a random interval graphs and compute the expected value of the number of different blocks for $\Omega(n)$ values of k. We choose the length of the 1-sequence of column j for all $0 \leq j \leq \frac{N}{2} - 1$ uniformly at random from $\{\frac{N}{2} - j, \ldots, N - 1 - j\}$ and for all $\frac{N}{2} \leq j \leq N - 1$ we set the length to $N - 1 - j$. Thus, the lengths of the 1-sequences within the $\frac{N}{2} \times \frac{N}{2}$ lower left submatrix R are uniform at random in $\{1, \ldots, \frac{N}{2}\}$ (see Fig. 2).

Lemma 1. *Let $G = (V, E)$ be a random interval graph generated by the above process. The probability that a fixed block in R of size $L_1 \times L_2$ with $L_1, L_2 \leq 2^{n/2-1}$ has exactly one changing position is at least $\frac{L_2 \cdot (L_1 - 1)}{2^{n-1}} \cdot e^{-1}$.*

Theorem 4. *The worst-case π-OBDD size of an interval graph is $\Omega(N \log N)$ where the nodes are labeled according to the interval starting points and π is an interleaved variable order with decreasing significance.*

Proof. Let $G = (V, E)$ be a random interval graph generated by the above process and $f := \chi_E$. We know that each $n/2+1 \leq k \leq (3/4)n$ induces a grid in R consisting of $2^{k-1} \cdot 2^{k-1}$ blocks $B^k_{i,j}$ of size $2^{n-k} \times 2^{n-k}$. Using Lemma 1, the expected number of blocks with exactly one changing position is at least $\frac{1}{2e} \cdot 2^k \cdot 2^k \cdot \frac{2^{n-k} \cdot (2^{n-k}-1)}{2^n} = \Omega(2^n)$. Now, we have to ensure that these blocks correspond to different subfunctions which are also essentially dependent on x_{n-k-1}. The subfunctions, where, additionally, x_{n-k-1} is replaced by 0 and 1, correspond to a half of the blocks. Thus, a block is symmetric iff the corresponding subfunction is not essentially dependent on x_{n-k-1}. Due to the one changing position in each block, this is not possible. Blocks $B^k_{i,j}$ and $B^k_{i',j}$ with exactly one changing position and $i \neq i'$ clearly correspond to different subfunctions because they are in the same block column. But blocks $B^k_{i,j}$ and $B^k_{i',j'}$ with $j \neq j'$, i. e., from different block columns, do not have to be different. By replacing some columns of the blocks by constants, we ensure that this also holds. Consider the case $k = (3/4)n$. For every $0 \leq j \leq 2^k - 1$ we fix the first k columns of $B^k_{i,j}$ with $0 \leq i \leq 2^k - 1$ such that they represent the binary number $[j]_2$. As a result, the blocks $B^k_{i,j}$ and $B^k_{i',j'}$ with $j \neq j'$ are always different. Since we looked at the finest grid, this also holds for smaller values of k because every larger block is equal to a union of small blocks. For $k = (3/4)n$ the number of fixed columns is $(3/4)n$ and in each $k \to k-1$ step this number is doubled, i. e., for $n/2+1 \leq k \leq (3/4)n$ the number of "free" columns is $2^{n-k} - 2^{(3/4)n-k} \cdot (3/4)n = \Omega(2^{n-k})$ for n large enough. Thus, the expected number of blocks with exactly one changing position remains $\Omega(2^n)$ for every $n/2+1 \leq k \leq (3/4)n$ which means there is an interval graph with π-OBDD size $\Omega(N \log N)$ □

4 Implicit Matching Algorithm on Unit Interval Graphs

In the following, the nodes of the unit interval graphs are labeled according to the sorted sequence of starting points. At first, we have to look into so called multivariate threshold functions.

Definition 6 (see, e. g., [30]). *Let $T \in \mathbb{Z}$ and $W \in \mathbb{N}$ and $w_1, \ldots, w_k \in \{-W, \ldots, W\}$. A Boolean function $f : \{0,1\}^{kn} \to \{0,1\}$ with input vectors $x^{(1)}, \ldots, x^{(k)} \in \{0,1\}^n$ and $f(x^{(1)}, \ldots, x^{(k)}) = 1 \Leftrightarrow \sum_{j=1}^{k} w_j \cdot |x^{(j)}| \geq T$ is called* k-variate threshold function. *The set of k-variate threshold functions $f \in B_{kn}$ with weight parameter W is denoted by $\mathbb{T}^W_{k,n}$.*

Woelfel [30] investigated the OBDD size for the variable order where the variables are tested with increasing significance, i. e., just the reverse of our variable order. Hosaka et al. [17] showed that the difference of the OBDD sizes for this two orders is at most $n - 1$. We can show that an OBDD using our variable order is not only small but can also be constructed efficiently which is important in view of the implementation. The result also implies that we can compute a sequence of $O(1)$ binary synthesis of multivariate threshold functions efficiently using our variable order if k and W are constants.

Theorem 5. *Let $f \in \mathbb{T}_{k,n}^{W}$ and $\pi_{k,n}$ be the k-interleaved variable order where the variables are tested with decreasing significance. Then we can construct a $\pi_{k,n}$-OBDD representing f with width $O(kW)$ and size $O(k^2Wn)$ in time $O(k^2Wn)$.*

We use the arithmetic notation in our algorithm instead of the functional notation, e. g., we denote by $|x| - |y| = 1$ the conjunction of the multivariate threshold functions $f(x, y) = 1 \Leftrightarrow |x| - |y| \geq 1$ and $g(x, y) = 1 \Leftrightarrow |y| - |x| \geq -1$.

Maximum Matching on Unit Interval Graphs. Let $G = (V, E)$ be a unit interval graph. Then we can make a simple observation (see [10]): W.l.o.g. G is connected. Then we have $\{v_i, v_{i+1}\} \in E$ for all $i = 0, \ldots, N-2$. Assume that there is an i such that $\{v_i, v_{i+1}\} \notin E$, then due to the connectivity there has to be another interval with starting point left of v_i and length greater than 1 (which is not possible) intersecting both intervals.

Algorithm 1 uses the characteristic function of the set of nodes which is here equal to $f(x) = 1 \Leftrightarrow |x| < N$. The algorithm computes a directed subgraph of G, which consists of the vertex disjoint paths visiting all nodes in the connected components. Maximum matchings on arbitrary vertex disjoint paths can be computed with $O(\log^2 N)$ functional operations [6]. Here, we know that every path consists of a consecutive sequence of nodes. We compute the set of starting nodes of the paths and the connected components of the graph: Two nodes x and y are connected iff every node z with $|x| \leq |z| < |y|$ has a successor $(v_{|z|}, v_{|z|+1}) \in E$. This can be computed by $O(\log N)$ operations. Next, we can compute the matching by adding every second edge of a path to the matching beginning with the first edge. While computing this set of edges needs $O(\log^2 N)$ operations in general [6], here we can easily determine the set by comparing the difference of two node labels due to the structure of the paths.

Algorithm 1. Implicit maximum matching algorithm for unit interval graphs

```
Input: Unit interval graph $\chi_E$
Output: Matching $\chi_M$
   // Compute path graph
1: $\chi_{\overrightarrow{E}}(x,y) = \chi_E(x,y) \wedge (|y| - |x| = 1)$
   // Compute set of starting nodes
2: $First(z) = (|z| < N) \wedge \forall x : \overline{\chi_{\overrightarrow{E}}(x,z)}$
   // Compute set of reachable nodes
3: $S(z) = \exists z' : \chi_{\overrightarrow{E}}(z,z')$
4: $Reachable(x,y) = (|x| \leq |y|) \wedge \forall z : (|x| \leq |z| < |y|) \Rightarrow S(z)$
5: $Reachable(x,y) = Reachable(x,y) \wedge (|x| < N) \wedge (|y| < N)$
   // Compute matching
6: $F(x) = \exists z, d : First(z) \wedge Reachable(z,x) \wedge (|x| - |z| = 2|d|)$
7: $M(x,y) = \chi_{\overrightarrow{E}}(x,y) \wedge F(x)$
8: $\chi_M(x,y) = M(x,y) \vee M(y,x)$
9: return $\chi_M$
```

Theorem 6. *Algorithm 1 computes a maximum matching for unit interval graphs using $O(\log N)$ functional operations.*

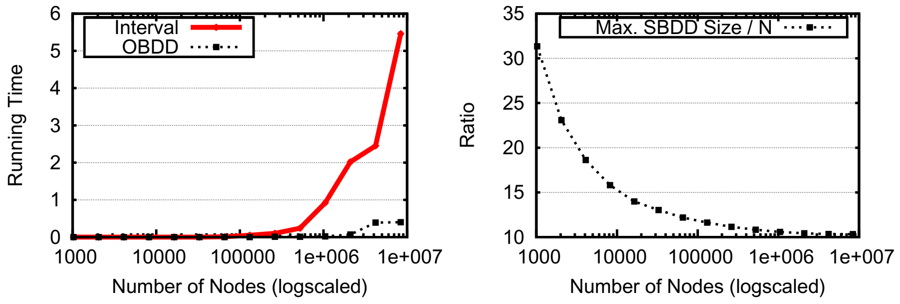


Fig. 3. Runtime and memory of the matching algorithms on random unit interval graphs. Memory plot shows the ratio of $S \log S$ (space usage of the OBDD-based algorithm) and number of nodes.

5 Experimental Evaluation

We evaluated the implicit maximum matching algorithm on unit interval graphs. Unit interval graphs can be represented as balanced nonnegative strings over {'[', ']'} (see, [24]) and such strings are created randomly using the algorithm in [1]. We generated 35 random graphs of size 2^i for $i = 10, \dots, 23$. The nodes of the graphs are encoded as in Section 3. We compare the OBDD-based algorithm to the algorithm which gets the interval representation as an input, sort the intervals according to their starting point and compute a maximum matching by scanning this sorted sequence with the same idea used in the implicit algorithm.

Experimental Setup. We implemented the implicit algorithm with the BDD framework CUDD 2.5.0[1] by F. Somenzi. The algorithms are implemented in C++ and were compiled with Visual Studio 2012 in the default release configuration. All source files, scripts and random seeds are publicly available[2]. The experiments were performed on a computer with a 2.6 GHz Intel Core i5 processor and 4 GB main memory running Windows 7. The runtime is measured by used processor time in seconds and the space usage of the implicit algorithm is given by the maximum SBDD size which came up during the computation, where a SBDD is a collection of OBDDs which can share nodes. Due to the small variance of these values, we only show the mean in the diagrams.

Results. The implicit matching algorithm outperforms the explicit matching algorithm on unit interval graphs (see Fig. 3). Even on graphs with more than 8 million nodes the implicit algorithm computes a maximum matching within 1 seconds. Storing a SBDD of size S needs $O(S \log S)$ bits. The memory diagram shows that the asymptotic space usage of the implicit algorithm on these instances is close to $O(N)$. Recall that the unit interval representation needs $\Theta(N \log N)$ space since $\log N$ bits are needed to represent the starting points.

[1] `http://vlsi.colorado.edu/~fabio/CUDD/`

[2] `http://ls2-www.cs.uni-dortmund.de/~gille/`

I. e., the implicit algorithm needs less space and can compute a maximum matching on larger instances than the explicit one. An interesting consequence of these results is that the submodules of our maximum matching algorithm, namely computing the connected components, a Hamiltonian path in every connected component and a maximum matching on these paths, are also very fast and space efficient which is surprising, since especially the computation of the transitive closure is often a bottleneck in implicit algorithms.

Open Questions. Using the π-ordered adjacency matrix, we think that it is possible to bound the OBDD size for other graph classes with a well structured adjacency matrix, e. g., convex graphs where the nodes can be ordered such that the neighborhood of every node consists of nodes which are consecutive in this order. The gap between the upper and lower bound of the OBDD size of interval graphs is $O(\log N)$. It is an interesting open question whether there is another labeling and/or variable order such that the OBDD size is $O(N)$ or the general lower bound can be increased to $\Omega(N \log N)$.

Even for a fixed variable order, the complexity of computing a node labeling for a given graph, such that the representing OBDD has minimal size, is unknown. The π-ordered adjacency matrix seems to help to prove upper/lower bounds on the OBDD size for a fixed labeling. Using this matrix to bound the size of OBDDs for every labeling could be object of further research.

The investigation of implicit algorithms on special graph classes seems quite promising and it would be interesting if the good performance can also be achieved for other large graph classes.

Acknowledgements. I thank Beate Bollig, Melanie Schmidt and Chris Schwiegelshohn for the valuable discussions and, together with the anonymous referees, for their comments on the presentation of the paper.

References

1. Arnold, D.B., Sleep, M.R.: Uniform random generation of balanced parenthesis strings. ACM Trans. Program. Lang. Syst. 2(1), 122–128 (1980)
2. Bloem, R., Gabow, H.N., Somenzi, F.: An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. Formal Methods in System Design 28(1), 37–56 (2006)
3. Bollig, B.: On symbolic OBDD-based algorithms for the minimum spanning tree problem. TCS 447, 2–12 (2012)
4. Bollig, B., Gillé, M., Pröger, T.: Implicit computation of maximum bipartite matchings by sublinear functional operations. In: Agrawal, M., Cooper, S.B., Li, A. (eds.) TAMC 2012. LNCS, vol. 7287, pp. 473–486. Springer, Heidelberg (2012)
5. Bollig, B., Löbbing, M., Wegener, I.: On the effect of local changes in the variable ordering of ordered decision diagrams. IPL 59(5), 233–239 (1996)
6. Bollig, B., Pröger, T.: An efficient implicit OBDD-based algorithm for maximal matchings. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 143–154. Springer, Heidelberg (2012)
7. Bollig, B., Wegener, I.: Asymptotically optimal bounds for OBDDs and the solution of some basic OBDD problems. Journal of Computer and System Sciences 61(3), 558–579 (2000)

8. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers 35(8), 677–691 (1986)
9. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Information and Computation 98(2), 142–170 (1992)
10. Chung, Y., Park, K., Cho, Y.: Parallel maximum matching algorithms in interval graphs. In: Proc. of the 4th ICPADS, pp. 602–609 (1997)
11. Coudert, O.: Doing two-level logic minimization 100 times faster. In: Proc. of the 6th SODA, pp. 112–121 (1995)
12. Gentilini, R., Piazza, C., Policriti, A.: Computing strongly connected components in a linear number of symbolic steps. In: Proc. of the 14th SODA, pp. 573–582 (2003)
13. Gentilini, R., Piazza, C., Policriti, A.: Symbolic graphs: Linear solutions to connectivity related problems. Algorithmica 50(1), 120–158 (2008)
14. Gillé, M.: OBDD-Based Representation of Interval Graphs. ArXiv e-prints (2013), `http://arxiv.org/abs/1305.2772`
15. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Annals of Discrete Mathematics, vol. 57. North-Holland Publishing Co. (2004)
16. Hachtel, G.D., Somenzi, F.: A symbolic algorithms for maximum flow in 0-1 networks. Formal Methods in System Design 10(2/3), 207–219 (1997)
17. Hosaka, K., Takenaga, Y., Kaneda, T., Yajima, S.: Size of ordered binary decision diagrams representing threshold functions. Theor. Comput. Sci. 180(1-2), 47–60 (1997)
18. Lai, Y.-T., Pedram, M., Vrudhula, S.B.K.: EVBDD-based algorithms for integer linear programming, spectral transformation, and function decomposition. IEEE Transactions on CAD of Integrated Circuits and Systems 13(8), 959–975 (1994)
19. Meer, K., Rautenbach, D.: On the OBDD size for graphs of bounded tree- and clique-width. Discrete Mathematics 309(4), 843–851 (2009)
20. Meinel, C., Theobald, T.: On the influence of the state encoding on OBDD-representations of finite state machines. ITA 33(1), 21–32 (1999)
21. Mertzios, G.B.: A matrix characterization of interval and proper interval graphs. Applied Mathematics Letters 21(4), 332–337 (2008)
22. Nunkesser, R., Woelfel, P.: Representation of graphs by OBDDs. Discrete Applied Mathematics 157(2), 247–261 (2009)
23. Roberts, F.S.: Indifference graphs. In: Harary, F. (ed.) Proof Techniques in Graph Theory, pp. 139–146 (1969)
24. Saitoh, T., Yamanaka, K., Kiyomi, M., Uehara, R.: Random generation and enumeration of proper interval graphs. IEICE Transactions 93-D(7), 1816–1823 (2010)
25. Sawitzki, D.: Implicit flow maximization by iterative squaring. In: Van Emde Boas, P., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2004. LNCS, vol. 2932, pp. 301–313. Springer, Heidelberg (2004)
26. Sawitzki, D.: The complexity of problems on implicitly represented inputs. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 471–482. Springer, Heidelberg (2006)
27. Sawitzki, D.: Exponential lower bounds on the space complexity of OBDD-based graph algorithms. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 781–792. Springer, Heidelberg (2006)
28. Sieling, D., Wegener, I.: NC-algorithms for operations on binary decision diagrams. Parallel Processing Letters 3, 3–12 (1993)
29. Wegener, I.: Branching programs and binary decision diagrams. SIAM Monographs on Discrete Mathematics and Applications (2000)
30. Woelfel, P.: Symbolic topological sorting with OBDDs. Journal of Discrete Algorithms 4, 51–71 (2006)

Tight Upper Bounds for Minimum Feedback Arc Sets of Regular Graphs

Kathrin Hanauer, Franz J. Brandenburg, and Christopher Auer

University of Passau, Germany
{hanauer,brandenb,auerc}@fim.uni-passau.de

Abstract. The Feedback Arc Set problem is one of the classical $\mathcal{NP}$-hard problems. Given a graph with n vertices and m arcs, it asks for a subset of arcs whose deletion makes a graph acyclic. An equivalent is the Linear Ordering problem, where the vertices are ordered from 1 to n, and a feedback arc is an arc that is directed contrarily. Both problems have been studied intensely.

Here, we add a new point of view. We first derive properties of linear orderings, that can be established efficiently. Our main result are upper bounds on the cardinality of a minimum feedback arc set for graphs with degree at most 3 and 4. We prove that the bounds are at most $n/3$ and $m/3$, respectively, and show that both are tight.

1 Introduction

Feedback Arc Set (FAS) is one of the classical $\mathcal{NP}$-hard decision problems. Given a directed graph G and an integer k, is there a subset S of G's arcs, $|S| \leq k$, such that every directed cycle of G contains an arc in S? [11] The optimization problem associated with FAS is to minimize the cardinality k of the feedback arc set S. The complementary problem is known as the Maximum Acyclic Subgraph (MAS) problem. A topological sorting of this acyclic graph leads to a linear ordering of the graph's vertices, which is often visualized by placing the vertices according to the ordering from left to right on a horizontal line. This perspective is known as the Linear Ordering problem (LO), which is equivalent to both FAS and MAS, depending on whether we count the arcs pointing from right to left or vice versa.

There is a number of applications for the Feedback Arc Set problem. Among them are ranking problems, as the LO formulation immediately suggests, penalty graph approaches, e. g., for 2-layered crossing minimization, and graph drawing [16].

Karp proved the $\mathcal{NP}$-hardness of FAS by a reduction from Vertex Cover [11]. Furthermore, FAS is hard to approximate. For arbitrary graphs, the best known ratio is $\mathcal{O}(\log n \log \log n)$ [6]. FAS is $\mathcal{APX}$-hard [10] and, unless $\mathcal{P}$ is equal to $\mathcal{NP}$, it cannot be approximated better than to a factor of 1.36 [4]. For tournaments, however, there is a 3-approximation algorithm [1] and a polynomial time approximation scheme [12] that admits an efficient computation of a solution that is at most by a factor of $1 + \epsilon$ worse than the optimum, for any

$\epsilon > 0$. These results show that FAS is in general a tough problem, which immediately raises the question for special cases, where FAS becomes tractable. Indeed, it can be solved to optimality in polynomial time on planar graphs [2], graphs that do not have the $K_{3,3}$ as a minor [14], or reducible flow graphs [15].

In contrast, FAS remains $\mathcal{NP}$-hard for 3-regular graphs [7]. Although many heuristics have been proposed to attack the Feedback Arc Set problem in general, only few of them admit a special analysis of their quality on regular graphs. Berger and Shor [3] were the first to give an algorithm that establishes bounds depending on a graph's maximum degree. For 3-regular graphs, they achieved $\frac{5}{18}m$. Eades and Lin [5] improved this bound to $\frac{m}{4}$ with an algorithm specifically designed for this class of graphs. The result by Berger and Shor for 4- and 5-regular graphs, $\frac{11}{30}m$, is still the best known so far. However, none of these bounds have been shown to be best possible.

In this paper, we introduce necessary properties for a linear ordering to be optimal and use them to obtain tight upper bounds for 3- and 4-regular graphs. Our proofs are based on the following intuition: Consider a graph along with an optimal linear ordering. Each backward arc, i. e., an arc pointing from right to left, produces a pebble that lies at its head. The pebble must be transported along a path to the backward arc's tail and placed on a vertex on this path that has neither an incident backward arc nor has another pebble already been placed on it. This vertex can then uniquely be assigned to the backward arc that produced the pebble. We will show that such an assignment is possible for every 3-regular graph, and that we can even assign to each backward arc its head and tail. Thereby, we obtain a tight bound of $\frac{n}{3}$. For 4-regular graphs, this approach is not directly applicable. However, using graph operations, we can prove a tight bound of $\frac{m}{3}$ here. Beyond that, we will show that a linear ordering that meets the bounds can be constructed in polynomial time.

2 Preliminaries

We consider simple, directed, strongly connected graphs $G = (V, A)$ with vertex set V and arc set $A \subseteq V \times V$, where $|V| = n$ and $|A| = m$. A graph is *strongly connected*, if for each pair of vertices $v, w \in V$ there is a directed path from v to w and vice versa. For a vertex $v \in V$, denote by $d^+(v)$ the number of outgoing arcs from v, by $d^-(v)$ the number of incoming arcs to v and by $d(v) = d^+(v) + d^-(v)$ its *degree*. Let $\delta(v) = d^+(v) - d^-(v)$ be v's *delta degree*. Note that for every graph $\sum_{v \in V} \delta(v) = 0$. An arc a with tail u and head v is specified by the tuple (u, v). A cycle $\mathcal{C} = (c_0, c_1, \ldots c_k, c_{k+1} = c_0)$, consisting of vertices $c_0, \ldots c_k$, is always directed and simple, i. e., $\forall 0 \le i \le k : (c_i, c_{i+1}) \in A$ and $\forall i \ne j : c_i \ne c_j$.

A *feedback arc set* for a graph G is a set of arcs $\mathcal{B} \subseteq A$ such that the graph $G_{\mathcal{F}} = (V, \mathcal{F})$, $\mathcal{F} = A \setminus \mathcal{B}$, is acyclic. We also say that a set of arcs is *feasible* if it forms a valid feedback arc set. A feedback arc set is called *minimal*, if no arc can be removed from it such that it remains feasible. $\mathcal{B}$ is *minimum* or *optimal*, if its cardinality is minimum among all feedback arc sets of G.

A *linear ordering* π of G is a bijective mapping $V \to [1 \dots n]$ that assigns to each vertex v a unique position $\pi(v)$. An arc (u, v) is called *forward*, if $\pi(u) < \pi(v)$, and otherwise *backward*. For every linear ordering, the set of backward arcs uniquely identifies a feedback arc set. In contrast, the linear ordering that corresponds to a given feedback arc set is in general not unique. However, every topological sorting of $G_{\mathcal{F}}$ yields a linear ordering whose set of backward arcs is a subset of $\mathcal{B}$. We say that π is optimal, if the induced feedback arc set is. The *quality* of a linear ordering is measured by the cardinality of its induced feedback arc set; the smaller it is, the higher is its quality.

A linear ordering π induces a *layout* $\mathcal{L} : V \to {\mathbb{N}_0}^4$ where for each vertex $v \in V$, $\mathcal{L}(v)$ is a 4-tuple (fi, fo, bi, bo) that defines the number of incoming forward, outgoing forward, incoming backward, and outgoing backward arcs of v in π. In order to improve comprehensibility, we use small figures that visualize the layout instead of the 4-tuples.

Observe that we can restrict ourselves to strongly connected graphs, as otherwise, the feedback arc set problem can be solved separately for each strongly connected component.

3 Properties of Optimal Linear Orderings

We now establish some properties of optimal linear orderings that will prove useful later. A condition similar to the following was also observed in [9].

Lemma 1. *(Nesting Property) For every optimal linear ordering π^* of a graph G, there are two injective mappings $l, r : \mathcal{B} \to \mathcal{F}$ such that*
$l(u, v) = (v, x) \Rightarrow \pi^*(x) < \pi^*(u)$ and $r(u, v) = (y, u) \Rightarrow \pi^*(v) < \pi^*(y)$.

Proof. Let π be an arbitrary linear ordering of G. Consider the movement of a vertex v to a new position p within π. If $p > \pi(v)$, then all outgoing forward arcs of v whose heads are at a position $< p$ are turned into backward arcs and all incoming backward arcs with tail in the same range become forward. Likewise, if $p < \pi(v)$, all incoming forward arcs become backward and outgoing backward arcs become forward if the other end vertex lies at a position $> p$.

Apply a 1-OPT algorithm to π that moves each vertex v to a position that minimizes the number of backward arcs incident to v for the current linear ordering. This procedure can be repeated until no vertex position can be improved (cf. Alg. 1). Let π' be the resulting linear ordering.

Define the injective mapping l as follows: For every vertex v, order the outgoing forward arcs $(v, x_1), (v, x_2), \dots (v, x_k)$ increasing in length, i. e., such that $\pi'(x_1) < \pi'(x_2) < \dots < \pi'(x_k)$. Do the same for the incoming backward arcs $(u_1, v), (u_2, v), \dots (u_l, v)$, with $\pi'(u_1) < \pi'(u_2) < \dots < \pi'(u_l)$. Note that $l \leq k$, otherwise moving v to position $\pi'(u_l) + 1$ decreases the number of backward arcs incident to v by $l - k$. For each $0 \leq i \leq l$, set $l(u_i, v) = (v, x_i)$.

Consider the backward arc (u_i, v) with $l(u_i, v) = (v, x_i)$. Suppose $\pi'(x_i) > \pi'(u_i)$. Then, however, moving v to position $\pi'(u_i) + 1$ would turn i backward arcs into forward arcs, but at most $i - 1$ forward arcs would become backward.

Algorithm 1. Nesting Property

```
1: function EstablishNesting($G, \pi$)
2:     repeat
3:         for all $v \in V$ do
4:             $s \leftarrow$ position that minimizes number of $v$'s incident backward arcs
5:             move $v$ to position $s$ in $\pi$
6:     until number of backward arcs did not change in last iteration
```

This contradicts the assumption that there is no position for v in π' that decreases the number of incident backward arcs further.

The injective mapping r can be obtained by processing the outgoing backward arcs and the incoming forward arcs for each vertex, again both increasing in length. The argument can be applied likewise.

In particular, this guarantees the nesting property for every optimal ordering π^*, since by definition, no ordering with strictly less backward arcs exists. □

Observe that $l[\mathcal{B}]$ and $r[\mathcal{B}]$ need not be disjoint. For an example, consider the graph depicted in Fig. 4(a). There, setting $l(5,2) = (2,4) = r(4,1)$ meets all requirements.

The second property introduces the concept of *forward path*s and characterizes a linear ordering that induces a minimal feedback arc set. Recall that every optimal feedback arc set is also minimal.

Lemma 2. *(Path Property) For every backward arc $b = (u, v) \in \mathcal{B}$ according to an optimal linear ordering π^*, there is a* forward path $p = (x_1 = v, x_2, \ldots, x_k = u)$ *such that for all* $1 \le i < j \le k : (x_i, x_j) \in \mathcal{F}$.

Proof. Let π^* be an optimal linear ordering of a graph G. Suppose there is a backward arc $b = (u, v)$ in π^* that lacks a forward path. Consider the feedback arc set $\mathcal{B}$ induced by π^*. Then, $\mathcal{B}$ is optimal. Remove b from $\mathcal{B}$ and insert it into $\mathcal{F}$. Since there is no path from v to u in $\mathcal{F}$, $G_{\mathcal{F}}$ remains acyclic.

Subsequently, $\mathcal{B} \setminus \{b\}$ is feasible, a contradiction to the optimality of $\mathcal{B}$ and therefore also π^*. □

A graph $G = (V, A)$ with a linear ordering π that respects the path property allows to construct a *forward path graph* $G_{\text{fp}} = (V_{\text{fp}}, A_{\text{fp}})$, where $V_{\text{fp}} \subseteq V$ and $A_{\text{fp}} \subseteq A$. Let $\mathcal{B}$ be the feedback arc set induced by π and $\mathcal{F}$ the corresponding set of forward arcs. For the construction of G_{fp}, select exactly one forward path p_b for each backward arc $b \in \mathcal{B}$. Then, $V_{\text{fp}} = \{v \in V \mid \exists b \in \mathcal{B} : p_b = (x_1, x_2, \ldots x_k) \wedge v \in \{x_1, \ldots x_{k-1}\}\}$ and $A_{\text{fp}} = \{a \in \mathcal{F} \mid \exists b \in \mathcal{B} : p_b = (x_1, x_2, \ldots x_k) \wedge a \in \{(x_i, x_{i+1})\}_{1 \le i \le k-2}\}$. Note that p_b always refers to the single forward path that was chosen for b. In other words, G_{fp} consists of the union of all selected forward paths, but the last vertex of each is cut off. See Figs. 2(b) and (c) for an example.

Alg. 2 shows a straightforward approach to ensure the path property for an arbitrary ordering. Note that if a feedback arc set is minimized in this way, then every topological sorting of the remaining acyclic graph must result in the same set of backward arcs.

Algorithm 2. Path Property

```
1: function EstablishForwardPaths($G, \pi$)
2:     repeat
3:         for all $b \in \mathcal{B}$ do
4:             if no forward path exists for $b$ then
5:                 $\mathcal{F} \leftarrow \mathcal{F} \cup \{b\}$
6:                 $\pi \leftarrow$ topological sorting of $G_{\mathcal{F}}$
7:     until no change in last iteration
```

Corollary 1. *Let $\mathcal{B}$ be the feedback arc set induced by a linear ordering π with forward arcs $\mathcal{F}$. Every topological sorting of $G_{\mathcal{F}}$ yields a set of backward arcs that equals $\mathcal{B}$ if and only if π fulfills the path property.*

In the next step, we combine both properties to obtain an even stronger statement. We define new injective mappings l^+ and $r^+ : \mathcal{B} \to \mathcal{F}$ similar to l and r, but with reduced range. If $l^+(u, v) = (v, x)$ for a backward arc (u, v), then there is a forward path for (u, v) starting with (v, x). Analogously, r^+ maps (u, v) to the last arc on a forward path from v to u. The mappings l^+ and r^+ can equally be described via a bipartite matching. For l^+, the neighborhood of every backward arc $(u, v) \in \mathcal{B}$ consists of all forward arcs that are the first on any forward path of (u, v). The mapping l^+ exists if and only if there is a $\mathcal{B}$-saturating matching, i. e., every backward arc can be mapped exclusively to one arc of its neighborhood. r^+ can be defined likewise. Using Hall's marriage theorem [8], we can show that if such a matching does not exist, then a feedback arc set of strictly smaller cardinality can be constructed efficiently.

Lemma 3. *(Combined Nesting and Path Property) For every optimal linear ordering π^*, there exists an injective mapping $l^+ : \mathcal{B} \to \mathcal{F}$ that assigns to every backward arc $b = (u, v) \in \mathcal{B}$ the first arc on a forward path p_l of b, i. e.,*
$l^+(u, v) = (v, x_2) \Rightarrow \exists$ forward path $p_l = (v = x_1, x_2, \dots, x_{k-1}, u = x_k)$.
Likewise, there is an injective mapping $r^+ : \mathcal{B} \to \mathcal{F}$ that assigns to every backward arc $b = (u, v) \in \mathcal{B}$ the last arc on a forward path p_r of b.

Finally, we see that certain layouts can be eliminated from every linear ordering.

Lemma 4. *(Eliminable Layouts) For every linear ordering π of G there is a linear ordering π' of at least equal quality such that π' does not induce one of the following layouts on any vertex v:*
$\mathcal{L}(v) = (x, 1, 1, 0)$ ( ) or $\mathcal{L}(v) = (1, x, 0, 1)$ ( ), where $x \geq 1$.

Proof. Let π be a linear ordering of G. We can assume that π respects the nesting and path property, otherwise, it can be established efficiently. This does not increase the number of backward arcs.

Let v be a vertex with layout within π. Let (u, v) be the only incoming backward arc of v and (v, v_1) be the nesting forward arc, i. e., $l(u, v) = (v, v_1)$ (cf. Fig. 1(a)). Consider the following operation: Modify π by moving v right behind u (cf. Fig. 1(b)). Now, (u, v) counts as forward arc and (v, v_1) is a new

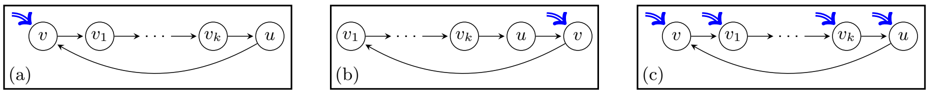


Fig. 1. π before (a) and after (b) moving v; (c): no termination

Algorithm 3. Combined Nesting and Path Property/Layout Elimination

```
 1: function EstablishProperties($G, \pi_0$)
 2:     $\pi_1 \leftarrow$ EstablishNesting($G, \pi_0$) with backward arcs $\mathcal{B}_1$;        ▷ $k_1 \cdot \mathcal{O}(n^2)$
 3:     $\pi_2 \leftarrow$ EstablishForwardPaths($G, \pi_1$) with backward arcs $\mathcal{B}_2$;  ▷ $k_2 \cdot \mathcal{O}(m^2)$
 4:     if $|\mathcal{B}_2| < |\mathcal{B}_1|$ then $\pi_0 \leftarrow \pi_2$; goto 2;
 5:     Compute all candidate forward arcs per backward arc;                  ▷ $k_3 \cdot \mathcal{O}(m^2)$
 6:     Compute maximum matching $M$;                                         ▷ $k_3 \cdot \mathcal{O}(m^{2.38})$
 7:     if $|M| < |\mathcal{B}_2|$ then $\mathcal{B}_3 \leftarrow$ new FAS via alternating paths;    ▷ $k_3 \cdot \mathcal{O}(m^2)$
 8:         $\pi_0 \leftarrow$ TopSort($G_{\mathcal{F}3}$); goto 2;                         ▷ $k_3 \cdot \mathcal{O}(m)$
 9:     if $\pi_2$ contains eliminable layout then                             ▷ $k_4 \cdot \mathcal{O}(n^2)$
10:         $\pi_0 \leftarrow$ reordering of $\pi_2$; goto 2;
```

backward arc, so the cardinality of the induced feedback arc set remains unchanged. Afterwards, the layout of v is . Other vertices whose layouts have changed are u and v_1. Suppose that the layout of u now is . Then, it must have been before the move, which is illegal, since π respects the nesting property. Likewise, u cannot have a layout afterwards, since its layout would have been before. Next, consider v_1. After the movement of v, v_1 has an incoming backward arc, so it cannot have as layout. However, it may now be . Then, its initial layout would have been . Hence, the elimination of the layout at v may in turn produce this layout at at most one new vertex and the process must be iterated.

It remains to show that the procedure eventually terminates. Note that if v_1 is moved behind v, then v's layout changes to . Suppose there is no termination. Then, there must be a cycle that has a vertex v with layout on the smallest position within π, followed by vertices $v_1, \ldots v_k$ with layout and finally a vertex u with layout . The cycle consists of the outgoing forward arcs of $v, v_1, \ldots v_k$ that correspond to the incoming forward arcs of $v_1, \ldots, v_k, u$ (in this order) and is closed by the backward arc (u, v) (cf. Fig. 1(c)). Because of the arcs that are not part of the cycle, there must be at least one vertex w in π that is not part of the cycle. Recall that G is strongly connected. Hence, there is a directed path from every vertex to every other vertex. In particular, this holds for v and w. However, there are no outgoing arcs from vertices within the cycle to vertices outside, so there cannot be a directed path from v to w; a contradiction. In consequence, the procedure terminates. Recall that it does not produce new vertices with layout , either.

We can show with a similar argument that vertices with layout can be eliminated likewise. □

We combine the procedures described above and obtain an algorithm (cf. Alg. 3) that establishes all properties. The comments on the right side in Alg. 3 indicate the running time per iteration in big $\mathcal{O}$ notation and the number of iterations k_1, k_2, k_3, k_4. The running time of EstablishNesting is $\mathcal{O}(n^2)$ per iteration (cf. Alg. 1). For a total number of k_1 iterations, it reduces the cardinality of the feedback arc set by $\mathcal{O}(k_1)$ arcs. EstablishForwardPaths is run each time after EstablishNesting has finished, so $k_2 \leq k_1$. The bipartite graph that needs to be constructed for the matching has $\mathcal{O}(m)$ vertices and $\mathcal{O}(m^2)$ edges. Hence, a maximum bipartite matching requires $\mathcal{O}(m^{2.38})$ time [13]. The part of the algorithm that guarantees the combined nesting and path property (ll. 5–8), is repeated $k_3 \leq k_2$ times. Finally, one vertex with eliminable layout causes at most $\mathcal{O}(n)$ movements, so we have $\mathcal{O}(n^2)$ for all vertices. This subroutine is carried out $k_4 \leq k_3$ times. As $k_4 \leq k_3 \leq k_2 \leq k_1$ and $k_1 \in \mathcal{O}(m)$, we get a total running time of $\mathcal{O}(m^{3.38})$.

Corollary 2. *There is an $\mathcal{O}(m^{3.38})$ time algorithm to construct a linear ordering that respects the combined nesting and path property and does not induce an eliminable layout on any vertex of the graph.*

4 A Tight Bound for Subcubic Graphs

In this section we consider subcubic graphs. A graph $G = (V, A)$ is *cubic* if for all vertices $v \in V : d(v) = 3$, and *subcubic*, if for all vertices $v \in V : d(v) \leq 3$.

Theorem 1. *The cardinality of an optimal feedback arc set of a subcubic graph is at most $\frac{n}{3}$ and this bound is tight.*

Let $G = (V, A)$ be a subcubic graph. Consider a linear ordering π that fulfills the nesting property. Among the vertices with an incident backward arc, we observe six different layouts:
$\mathcal{L}(v) = (0,1,1,0)$: , $\mathcal{L}(v) = (1,1,0,1)$: , $\mathcal{L}(v) = (2,0,0,1)$: ,
$\mathcal{L}(v) = (1,0,0,1)$: , $\mathcal{L}(v) = (1,1,1,0)$: , $\mathcal{L}(v) = (0,2,1,0)$: .
Recall that by Lemma 4, we can eliminate the layouts and without decreasing the quality of π.

Proof. Let π^* be an optimal linear ordering of a subcubic graph $G = (V, A)$. By Lemma 1 and 2, π^* fulfills the nesting and the path property. Furthermore, by Lemma 4, we can assume that π^* does not contain vertices with layout or . Construct a forward path G_{fp} of G according to π^*. Fig. 2 provides an example. As the layouts of vertices with incident backward arcs were one of , , , and , every vertex s that has an incoming backward arc in G is a source in G_{fp} with $\delta(s) = 1$ and there are no other sources. All other vertices in G_{fp} are vertices that are not incident to a backward arc in G. Furthermore, G_{fp} has no vertex v with $\delta(v) = -2$. Otherwise, v must have layout according to π^* with an outgoing forward arc to a vertex with an outgoing backward arc, i. e., one with layout or , and the two incoming forward arcs of v hence would

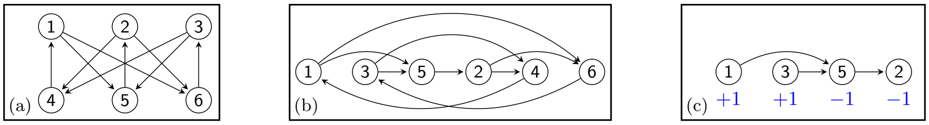


Fig. 2. The complete bipartite graph $K_{3,3}$ (a), an optimal linear ordering (b) and a forward path graph with delta degrees (c)

belong to two different forward paths for the same backward arc. However, for the construction of G_{fp}, exactly one forward path per backward arc was selected.

Consequently, $\forall v \in V_{\text{fp}} : \delta(v) \in \{-1, 0, 1\}$. As $\sum_{v \in V_{\text{fp}}} \delta(v) = 0$, there must be at least as many vertices v with $\delta(v) = -1$ as there are sources. Therefore, for every backward arc in G, there are three exclusive vertices: its tail, its head, which corresponds to a source in G_{fp}, and a vertex v with $\delta(v) = -1$ in G_{fp}. Hence, the cardinality of an optimal feedback arc set for G is at most $\frac{n}{3}$.

The graph in Fig. 2 or a directed triangle testify that this bound is tight. □

By Corollary 2, we can construct a linear ordering that meets the preconditions of Theorem 1 in $\mathcal{O}(m^{3.38})$. Observe that for subcubic graphs, the path property directly implies the combined nesting and path property, so these steps can be omitted in Alg. 3. Furthermore, $m \in \mathcal{O}(n)$ for subcubic graphs, so we obtain:

Corollary 3. *There is an $\mathcal{O}(n^3)$-time algorithm to construct a feedback arc set with cardinality at most $\frac{n}{3}$ for a subcubic graph G.*

5 An Extension for Graphs with Vertices of Degree 4

For a start, we restrict ourselves to quartic graphs. A graph is *4-regular* or *quartic*, if $\forall v \in V : d(v) = 4$. We can later loosen this restriction such that we obtain a bound for *subquartic* graphs, i. e., $\forall v \in V : d(v) \leq 4$.

Theorem 2. *The cardinality of an optimal feedback arc set of a quartic graph is at most $\frac{m}{3}$ and this bound is tight.*

Let $G = (V, A)$ be a quartic graph. Consider again a linear ordering π that fulfills the nesting property. Here, we have nine different layouts for vertices with an incident backward arc:

$\mathcal{L}(v) = (2,1,1,0)$: , $\mathcal{L}(v) = (1,2,0,1)$: , $\mathcal{L}(v) = (1,2,1,0)$: ,
$\mathcal{L}(v) = (2,1,0,1)$: , $\mathcal{L}(v) = (0,3,1,0)$: , $\mathcal{L}(v) = (3,0,0,1)$: ,
$\mathcal{L}(v) = (0,2,2,0)$: , $\mathcal{L}(v) = (2,0,0,2)$: , $\mathcal{L}(v) = (1,1,1,1)$: .

By Lemma 4, we can eliminate the vertex layouts and without decreasing the quality of π. It would be convenient now if we could reuse the arguments from Sect. 4. Our intermediate goal is to reduce the degree of the vertices, and, most notably, the number of incident backward arcs per vertex. At the

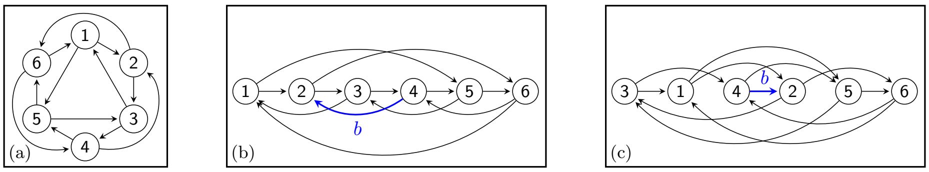
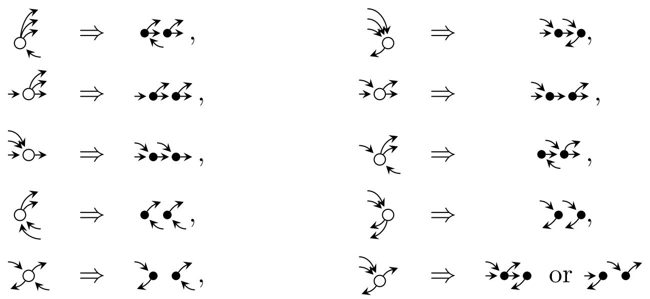


Fig. 3. Quartic graph with 4 disjoint cycles (a), linear ordering that respects all properties (b), and an optimal linear ordering (c)

same time, we want to be able to maintain the combined nesting and path property. We split every vertex v into two vertices, depending on $\mathcal{L}(v)$, as follows:



While most split operations preserve all forward paths by construction, one that definitely does not is the split of vertices with layout into two vertices with layouts . However, it is indeed possible to show that if the forward path property is destroyed thereby, then a feedback arc set of strictly smaller cardinality can be found. For a sketch of the proof, consider the graph depicted in Fig. 3. The linear ordering given in Fig. 3(b) respects the combined nesting and path property and there are no vertices with eliminable layouts. If vertex 3 is split here, backward arc $b = (4, 2)$ loses its only forward path $(2, 3, 4)$. Consider forward arc $(2, 3)$. All cycles running through vertex 3 must enter 3 either through this arc or through the backward arc $(5, 3)$. Moreover, all cycles that run through b and no other backward arc have to leave vertex 2 via $(2, 3)$. Hence, we can replace both b and $(3, 1)$ in the current feedback arc set by the single arc $(2, 3)$ and thus obtain a new feedback arc set of smaller cardinality (cf. Fig. 3(c)).

Lemma 5. *Every vertex with layout in an optimal linear ordering π^* can be split into two vertices such that the combined nesting and path property is preserved.*

This result is mainly a preparation for the correctness of the complete set of split operations. The proof of the following lemma specifies in what order the splits must be carried out and how to decide for the correct split version of vertices with layout .

Lemma 6. *For every optimal linear ordering π^*, each vertex v with $d(v) = 4$ can be split into two vertices such that the resulting ordering fulfills the nesting and the path property.*

Proof. Let π^* be an optimal linear ordering of a quartic graph G. We construct the split graph G_{s} of G as follows: Carry out the split operation for all vertices with layout . By Lemma 5, the combined nesting and path property still holds. Next, apply all other split operations except for vertices with layout . For vertices with layout and , the split must be such that the path property holds afterwards. Observe that the split operations for all other vertices do not affect the existence of forward paths.

Finally, for vertices with layout , we distinguish two cases. Let v be such a vertex. If v is split into two vertices with layout , then all forward paths are preserved. However, we want to avoid this split if the head of the outgoing forward arc of v has an outgoing backward arc, too. We know that all vertices with an outgoing backward arc have the layout after the split. In order to make sure that the path property is not violated, we select exactly one forward path of every backward arc in the split graph. Process the vertices with layout in descending order of their position according to π^*.

Consider a vertex v with layout , whose outgoing forward arc points to a vertex with layout . Recall that this vertex is already a split vertex. Suppose both incoming forward arcs of v belong to one of the selected forward paths. Then, these forward paths can only belong to either the backward arc leaving at v or the backward arc leaving at the vertex with layout . Subsequently, there are exactly two forward paths via v, so v can be split into two vertices with layouts without destroying a forward path. In all other cases, vertices with layout are split into . □

Note that none of the split operations creates a vertex with an eliminable layout. Now all requirements for the proof of Theorem 2 have been set.

Proof (Theorem 2). Let π^* be an optimal linear ordering of a quartic graph G. Fig. 4 shows the following steps for an example graph. Split G as defined above and the vertices with layout exactly as described in the proof of Lemma 6. We refer to the graph obtained after the splitting as G_{s} and the corresponding linear ordering as π^*_{s}. By Lemma 6, π^*_{s} respects the path property. Construct a forward path graph $G_{\mathrm{fp}} = (V_{\mathrm{fp}}, A_{\mathrm{fp}})$ of G according to π^*_{s}. For all vertices $v \in V_{\mathrm{fp}}$ with $d(v) \leq 3$, we have already shown in the proof of Theorem 1 that $\delta(v) \in \{-1, 0, 1\}$. The only vertices of degree 4 in G_{s} are those that were created by splitting either a vertex with layout into two vertices with layout or a vertex with layout into two vertices with layout . We refer to the split vertices of a vertex v as v_a and v_b. In the first case, the vertex of degree 4, v_b, has at least one incoming arc from v_a in G_{fp}, so $\delta(v_b) \leq 1$. Suppose it has no outgoing arcs in G_{fp}. Then, each of both outgoing arcs of v points to a vertex with an outgoing backward arc, hence, v is part of exactly two forward paths. Consequently, one of them is the forward path of the incoming backward arc of v. Let (u, v) be this backward arc. Then, in G the forward path consists of a single arc (v, u), but G is simple and therefore free of two-cycles. Thus, $\delta(v_b) \geq -1$. In the second case, if the vertex with degree 4 has two incoming forward arcs in G_{fp},

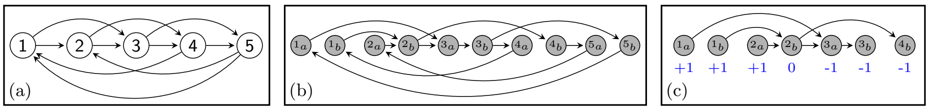


Fig. 4. Quartic graph: optimal linear ordering (a), its split graph (b), and a forward path graph with delta degrees (c)

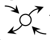
then it also has an outgoing forward arc in G_{fp}. Otherwise, the outgoing forward arc would be the head of a vertex with outgoing backward arc and the other split operation would have been applied. Hence, its delta degree is in $\{-1, 0, 1\}$, too. Furthermore, every vertex that has an incoming backward arc according to π_{s}^* is a source in G_{fp} with exactly one outgoing arc and there are no other sources. As $\sum_{v \in V_p} \delta(v) = 0$, there is a vertex v with $\delta(v) = -1$ for every source in G_{fp}. Consequently, for every backward arc there are again three exclusive vertices in G_{s}. Since every vertex of G has been split in exactly two vertices in order to obtain G_{s}, there are at least $\frac{3}{2}$ vertices per backward arc in G. Hence, the cardinality of an optimal feedback arc set for G is at most $\frac{2}{3}n = \frac{m}{3}$. This bound is tight, as the graph in Fig. 3 shows. □

We now loosen the restriction that every vertex v in the graph must have exactly degree $d(v) = 4$ to $d(v) \leq 4$ and obtain an upper bound for subquartic graphs:

Corollary 4. *The cardinality of an optimal feedback arc set of a subquartic graph is at most $\frac{m}{3}$ and this bound is tight.*

Proof. Combine the procedures for vertices v with degree $d(v) \leq 3$ and those for vertices with $d(v) = 4$. Let π^* be an optimal linear ordering of a subquartic graph G. Eliminate vertex layouts according to Lemma 4. Split all vertices v with $d(v) = 4$ such that the resulting linear ordering still fulfills the nesting and the path property. This can be done in the same way as described in the proof of Lemma 6. All other vertices remain unchanged. Analogously to the proofs of Theorem 1 and 2, to every backward arc, three vertices can be assigned in the split graph.

Let c be the number of vertices v in G with $d(v) \leq 3$ and q the number of vertices with $d(v) = 4$. Then, $n = c + q$ and $m = \frac{3}{2}c + 2q$. Let x be the number of vertices in the split graph, i.e., $x = c + 2q$. We can assign to each backward arc at least three vertices of the split graph. Hence we have $|\mathcal{B}| \leq \frac{x}{3} = \frac{c+2q}{3} = \frac{c}{3} + \frac{2}{3}q \leq \frac{c}{2} + \frac{2}{3}q = \frac{m}{3}$. Since subquartic graphs include quartic graphs, the bound is tight here, too. □

As in the subcubic case, we can construct a linear ordering that meets the preconditions of Theorem 2 in polynomial time and $m \in \mathcal{O}(n)$. Apart from the combined nesting and path property, we have to check whether the split of vertices with layout  violates the forward path property. This can be accomplished in time $\mathcal{O}(n)$ for splitting all vertices with this layout and $\mathcal{O}(n^2)$ for the check. If the property is violated, the new feedback arc set can be constructed in $\mathcal{O}(n)$.

Corollary 5. *There is an $\mathcal{O}(n^{3.38})$-time algorithm to construct a feedback arc set with cardinality at most $\frac{m}{3}$ for a subquartic graph G.*

6 Conclusion

We have established tight bounds for both subcubic and subquartic graphs. Furthermore, we have shown that a linear ordering that meets these bounds can be constructed in polynomial time, using a combination of standard algorithms.

For graphs with maximum degree 5, we conjecture that the bound of $\frac{2}{3}n$ holds, too. For degree 6, there are graphs that require more backward arcs.

References

1. Ailon, N., Charikar, M., Newman, A.: Aggregating inconsistent information: Ranking and Clustering. Journal of the ACM 55(5), Article 23 (2008)
2. Bang-Jensen, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications, 2nd edn. Springer, London (2006)
3. Berger, B., Shor, P.W.: Approximation algorithms for the maximum acyclic subgraph problem. In: Proc. First ACM-SIAM Symposium on Discrete Algorithms, pp. 236–243 (1990)
4. Dinur, I., Safra, S.: On the hardness of approximating minimum vertex cover. Annals of Mathematics 162, 439–485 (2005)
5. Eades, P., Lin, X.: A Heuristic for the Feedback Arc Set Problem. Australasian Journal of Combinatorics 12, 15–25 (1995)
6. Even, G., Naor, J.S., Schieber, B., Sudan, M.: Approximating Minimum Feedback Sets and Multi-Cuts in Directed Graphs. In: Balas, E., Clausen, J. (eds.) IPCO 1995. LNCS, vol. 920, pp. 14–28. Springer, Heidelberg (1995)
7. Gavril, F.: Some $\mathcal{NP}$-complete problems on graphs. In: Proc. 11th Conference on Information Sciences and Systems. Johns Hopkins Univ. (1977)
8. Hall, P.: On representatives of subsets. J. London Math. Soc. 10(1), 26–30 (1935)
9. Helmstädter, E.: Die Dreiecksform der Input-Output-Matrix und ihre möglichen Wandlungen im Wachstumsprozeß. In: Neumark, F. (ed.) Strukturwandlungen einer wachsenden Wirtschaft. Schriften des Vereins für Socialpolitik (1964)
10. Kann, V.: On the Approximability of $\mathcal{NP}$-complete Optimization Problems. Ph.D. thesis, Royal Institute of Technology, Stockholm, Sweden (May 1992)
11. Karp, R.M.: Reducibility Among Combinatorial Problems. In: Complexity of Computer Computations, pp. 85–103 (1972)
12. Kenyon-Mathieu, C., Schudy, W.: How to rank with few errors: A PTAS for Weighted Feedback Arc Set on Tournaments. ECCC 13(144) (2006)
13. Mucha, M., Sankowski, P.: Maximum matchings in planar graphs via Gaussian elimination. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 532–543. Springer, Heidelberg (2004)
14. Nutov, Z., Penn, M.: On the integral dicycle packings and covers and the linear ordering polytope. Discrete Applied Mathematics 60(1-3), 293–309 (1995)
15. Ramachandran, V.: A minimax arc theorem for reducible flow graphs. SIAM Journal on Discrete Mathematics 3(4), 554–560 (1990)
16. Sugiyama, K., Tagawa, S., Toda, M.: Methods for Visual Understanding of Hierarchical System Structures. IEEE Transactions on Systems, Man and Cybernetics 11(2) (February 1981)

A Linear-Time Kernelization for the Rooted k-Leaf Outbranching Problem

Frank Kammer

Institut für Informatik, Universität Augsburg, 86135 Augsburg, Germany
`kammer@informatik.uni-augsburg.de`

Abstract. In the Rooted k-Leaf Outbranching Problem, a digraph $G = (V, E)$, a vertex r of G, and an integer k are given, and the goal is to find an r-rooted spanning outtree of G with $\geq k$ leaves (a tree with vertex set V, all edges directed away from r, and $\geq k$ leaves). We present a linear-time algorithm to compute a problem kernel with $O(k^6)$ vertices and $O(k^7)$ edges for the Rooted k-Leaf Outbranching Problem. By combining the new result with a result of Daligault and Thomassé [IWPEC 2009], a kernel with a quadratic number of vertices and edges can be found on n-vertex m-edge digraphs in time $O(n + m + k^{14})$.

1 Introduction

To solve an NP-hard problem P, a common approach is to consider P as a *parameterized problem* described by a language $L \subseteq \Sigma^* \times I\!N$. An instance of P is a tuple $(x, k) \in \Sigma^* \times I\!N$ composed of the problem specification x and a problem-specific *parameter* k. We call (x, k) a *yes instance* if $(x, k) \in L$, and otherwise a *no instance*. The goal is to solve P in $n^c \cdot f(k)$ time, where n is the size of the input, c is a constant, and f is an arbitrary function. There are two research lines; one that tries to make f grow as slowly as possible and one that tries to achieve the smallest possible c; one example of the latter is the linear-time algorithm for Treewidth [5].

In recent years, several polynomial-time algorithms were published that transform an instance (x, k) into a tuple $(x', k') \in \Sigma^* \times I\!N$ such that $|x'|$ and k' are bounded by a polynomial in k and such that (x, k) is a yes instance if and only if (x', k') is a yes instance. Such an algorithm is called a *kernelization*, and (x', k') is a *kernel* for (x, k). For numerous problems, a race has started to obtain better and better bounds on the size of the kernel and, thus, on the corresponding running times with more and more slowly growing functions f. In contrast, the only linear-time kernelizations known to the author are for Vertex cover [7], for (Bi)Cluster Graph Editing [11] on general undirected graphs, for d-Hitting Set [2] on undirected hypergraphs, and for Feedback Vertex Set [10] and Dominating Set [3,8] on undirected planar graphs.

We present a linear-time algorithm that, given a general digraph G and a parameter $k \in I\!N$, computes a kernel of (G, k) with $O(k^6)$ vertices and $O(k^7)$ edges for the Rooted k-Leaf Outbranching Problem. As usual, an *outtree*

T is a rooted tree in which all edges are directed away from a vertex r called *the root*. Other vertices with only one incident edge are called *leaves*. A vertex u is an ancestor (descendant) of a vertex v in T if there is a directed path from u to v (from v to u) in T. An ancestor or descendant u of a vertex v is called *proper* if $u \neq v$. A *child* (*parent*) of a vertex v is a proper descendant (ancestor) of v that is adjacent to v.

Definition 1. *A* k-leaf outtree (r-rooted outtree) *is an outtree with at least* $k \in \mathbb{N}$ *leaves* (*with root* r). *An* outbranching *of a digraph* $G=(V,E)$ *is an outtree* $T=(V,F)$ *with* $F \subseteq E$. *In the* (Rooted) k-Leaf Outbranching Problem, *we are given a digraph* $G=(V,E)$, $k \in \mathbb{N}$ *(and* $r \in V$*), and the goal is to find an (*r*-rooted)* k*-leaf* outbranching *of* G.

In 2009 Fernau et al. [4] showed that an $O(k^3)$-sized kernel can be found for the Rooted k-Leaf Outbranching Problem in polynomial time, whereas this is not true for the related non-rooted version unless $PH = \Sigma_p^3$. In the same year, Daligault and Thomassé [6] described an algorithm to compute a kernel of size $O(k^2)$ for the Rooted k-Leaf Outbranching Problem. An efficient implementation of the algorithm runs in quadratic time. By running the algorithm of Daligault and Thomassé on the output of the algorithm presented in this paper, we can construct a kernel of size $O(k^2)$ on a digraph (V,E) in time $O(|V|+|E|+k^{14})$. If there is a polynomial-time algorithm (including algorithms still to be discovered) that computes a kernel of a certain size for this problem, we get a kernel of the same size in $O(|V|+|E|+k^{O(1)})$ time with the new algorithm.

2 Reduction Rules

A *reduction rule* for a parameterized problem P described by a language L is a polynomial-time algorithm that, *applied* to an instance (x,k) of P, computes an instance (x',k') of P with $(x,k) \in L \Leftrightarrow (x',k') \in L$. Our algorithm is based on one new and several well-known reduction rules [4,6], which we apply in a certain order. Assume that we are given an instance of the parameterized Rooted k-Leaf Outbranching Problem, i.e., a digraph $G=(V,E)$, $r \in V$, and $k \geq 2$. For all $v \in V$, let $N^-(v) = \{u \mid (u,v) \in E\}$ and $N^+(v) = \{u \mid (v,u) \in E\}$. A set $S \subseteq V$ is called a *separator for a set* $U \subseteq V \setminus S$ if all directed paths from r to a vertex in U contain a vertex in S. Following usual terminology, if $S=\{v\}$ is a separator for $U=\{u\}$, we say that v is a *dominator of* u and that u is *dominated* by v. We call a vertex a *dominator* if it dominates at least one other vertex. An edge (u,v) is *useless* if v is a dominator for u, and *useful* otherwise. A list L of vertices $(v_1,\ldots,v_t)$ ($t \in \mathbb{N}$) is a *bipath* of *length* $t-1$ in a digraph G' if, for all $i=2,\ldots,t-1$, v_i is incident to (v_{i-1},v_i), (v_i,v_{i-1}), (v_i,v_{i+1}), and (v_{i+1},v_i), but to no other edge in G'. We now define our reduction rules.

Rule 1 (unreachable rule (Rule 1 in [4] and 0 in [6])). *If a vertex is not reachable from* r*, reduce the given instance to a trivial no instance.*

Rule 2 (useless-edge rule at (u, v) (Rule 2 in [4])). *If the edge (u, v) is useless, remove it.*

Rule 3 (separator rule at (u, w) (Rule 4 in [4])). *Remove the edge (u, w) if there is a separator $S \subseteq V \setminus \{r, u\}$ for $\{u\}$ of size at most 2 such that an edge (v, w) exists for all $v \in S$.*

Rule 4. (dominator rule at v (Rule 1 in [6] combined with Rule 2 in [4])) *If v is a dominator, then remove v with all its incident edges and add all edges from each vertex in $N^-(v)$ to each vertex in $N^+(v)$ except those that would be useless, self-loops or copies of existing edges.*

Rule 5 (bipath rule (Rule 2 in [6])). *If $P = (v_1, \ldots, v_t)$ is a bipath of length ≥ 5, then replace P by the bipath (v_1, v_2, v_{t-1}, v_t).*

Rule 6 (shortcut rule at v (new)). *If there are pairwise distinct vertices u, x, y with $N^-(u) = \{v\}$ and edges (y, v) and (v, x), then add a new edge (y, x) if it does not already exist.*

The correctness of Rules 1-5 follows from [4,6]. Consider the situation described in Rule 6. A k-leaf outbranching in the original digraph is a k-leaf outbranching in the modified digraph. For the reverse direction, take an outbranching T that uses (y, x). By modifying T, we obtain an outbranching with at least the same number of leaves, but without the edge (y, x) as follows: If v is a descendant of y in T, replace the edge from the parent of v in T to v by the edge (y, v). At this point, v cannot be a descendant of x. Now replace (y, x) by (v, x). Before and after the replacement, v has u as a child. Thus the modifications transform an outtree into an outtree with at least the same number of leaves. Rule 6 is correct.

The separator rule is applied exhaustively by Fernau et al. [4], so that the running time can be bounded only by a polynomial. In contrast, we use the separator rule only if we already know the separator S. Moreover, a single application of the dominator rule, which is used by Daligault and Thomassé [6], can add a quadratic number of edges and, hence, by itself incurs a quadratic running time. Informally speaking, the new shortcut rule allows us to do the modifications of the dominator rule one by one.

If the reduction rules are applied repeatedly in an interspersed fashion, one application of a reduction rule usually scans the whole graph and therefore takes linear time. An additional idea of our algorithm is to structure the application of the reduction rules by means of a breadth-first search (BFS). In contrast to [6,4], our algorithm uses a *dominator tree* [1,9] to apply the useless-edge rule in constant time per edge.

3 Definitions and the Algorithm

Let $G = (V, E)$ be a digraph and let $r \in V$. For an r-rooted outtree $T = (V, F)$ with $F \subseteq E$, we *classify* the edges (u, v) of G with respect to T as follows: each edge in T is a *tree edge*; a non-tree edge (u, v) is a *forward edge* (*back edge*) if T contains a directed path from u to v (from v to u); the remaining edges of G are *cross edges*. When classifying an edge of G w.r.t. T, we also speak of a tree, forward, back, or cross edge of (G, T). For an edge (u, v), we call u and v the *tail* and the *head of* (u, v), respectively. We also say that (u, v) is an *incoming edge of* v and an *outgoing edge of* u. *To contract* an edge (u, v) means to replace u and v by a new vertex w, then to replace the endpoints u and v of all edges by w, and finally to remove self loops and multiple edges.

We also need some non-standard definitions. Call a vertex u of T a *branching vertex* if u is the root of T or has ≥ 2 children in T. A *treepath* of (G, T) is a directed path in T without a branching vertex. For a vertex v belonging to a treepath P of (G, T), edges of the forms (u, v) and (v, w) are called *entering edges of* P and *exiting edges of* P, respectively, if neither they nor their reverses are tree edges and u and w do not belong to P. A maximal (nonextensible) subpath Q of a maximal treepath P such that none of the vertices of Q is the head of an entering edge of P is called an *isolated treepath* of (G, T). If (G, T) or P are clear from the context, we may omit these terms. When we later apply the shortcut rule, we want to add edges connecting so-called essential vertices of the same isolated treepath. A vertex v is *essential* if it is part of an isolated treepath and not dominated by any vertex in the same isolated treepath. Unless stated otherwise, here and below "domination" is meant with respect to the whole graph G. Note that an essential vertex has an incoming back edge unless it is the first vertex of an isolated treepath or the head of a forward edge. The *attachment* of an essential vertex v of an isolated treepath Q is the (possible empty) set consisting of all vertices of Q dominated by v.

Our algorithm is shown in pseudocode below—the concepts of "jumping back edges" and "strongly isolated treepaths" are defined subsequently. Suppose that we are given an input consisting of a digraph G, a prescribed root r and an integer k. An important idea of the algorithm is to apply a sequence of reduction rules to G and corresponding modifications to an easy-to-compute r-rooted outbranching T of G to arrive at a pair $(\tilde{G}, \tilde{T})$ for which, roughly speaking, the number of non-tree edges is polynomial in k. With such a pair $(\tilde{G}, \tilde{T})$, the set V' of vertices incident on non-tree edges is of size $k^{O(1)}$. Assume that $\tilde{T}$ has fewer than k leaves (otherwise, we can return a trivial yes instance). Then the set V'' of branching vertices is smaller than k, and $\tilde{G}$ has fewer than $2k$ maximal treepaths. The vertices outside of $V' \cup V''$ induce $k^{O(1)}$ vertex-disjoint paths in $\tilde{G}$. Let G' be the graph obtained from $\tilde{G}$ by applying the bipath-rule to each of these paths to shrink it to constant length. Since G' has $k^{O(1)}$ vertices and we apply only reduction rules, (G', k) is a kernel.

Initially, we take T to be an arbitrary outbranching computed by a BFS in Step 1 and 2. Thus, (G, T) has no forward edges. As we show later in Lemma 3, this property is maintained during the whole algorithm.

For the next steps, consider an isolated treepath $Q = (q_1, q_2, q_3, \ldots)$ as the example in Fig. 1. Let w be an essential vertex of Q with a non-empty attachment S. By Lemma 4, each $v \in S$ can have only a tree edge and useless back edges

Algorithm 1. A kernel for the r-Rooted k-Leaf Outbranching Problem

```
Input: A digraph $G=(V,E)$, $r \in V$, and $k \geq 2$.  Output: A kernel of size $O(k^7)$.
1.) if some vertex is not reachable from $r$, then return a trivial no instance.
2.) Compute an outbranching $T$ with root $r$ by a BFS.
3.) Remove all useless back edges of $(G,T)$ by using a dominator tree—see Sect. 4.
4.) Traverse $T$ bottom-up, and for each vertex $u$ in $G$ with parent $v$ in $T$ do
      if ($N^-(u) = \{v\}$ /* $v$ is a dominator */ ) and ($|N^-(v)| = 1$)
              and (neither $v$ nor its parent in $T$ is a branching vertex) then
        Apply the dominator rule at $v$ both to $G$ and to $T$.
5.) for each strongly isolated treepath $R$ of $(G,T)$ do for each vertex $v$ of $R$
    that dominates its child and has a non-exiting back edge $(v,x)$ do
      Let $(y,v)$ be a back edge. Apply the shortcut rule at $v$ to add $(y,x)$.
6.) for each strongly isolated treepath $R = (u_1, \ldots, u_t)$ of $(G,T)$ do
      for each vertex $w$ not in $R$ with $\ell \geq 3$ edges of the form (vertex in R, $w$) do
          Choose $\ell$ maximal and $1 < \sigma_1 < \ldots < \sigma_\ell < t$ s.t. $(u_{\sigma_1}, w), \ldots, (u_{\sigma_\ell}, w) \in E$.
          $x := \ell$; if $u_{\sigma_{\ell-1}}$ is a dominator of $u_{\sigma_\ell}$, then $x := \ell - 1$
          Apply the separator rule with $S = \{u_{\sigma_1}, u_{\sigma_x}\}$ at each edge
          $e \in \{(u_{\sigma_2}, w), \ldots, (u_{\sigma_\ell}, w)\} \setminus \{(u_{\sigma_x}, w)\}$ to remove $e$.
7.) if ($T$ has $\geq k$ leaves)
    || (a maximal treepath of $(G,T)$ has $\geq 2k$ different heads of entering edges)
    || (a vertex of a maximal treepath $P$ is the tail of $\geq k$ back edges not exiting $P$)
    || (an isolated treepath of $(G,T)$ has $\geq 4k^2$ heads of jumping back edges)
    then return a trivial yes instance.
8.) for each strongly isolated treepath $R = (u_1, \ldots, u_t)$ of $(G,T)$ do
      for each maximal subpath $R'$ of $R$ consisting exclusively of vertices
        such that each vertex is not the tail of an exiting or jumping back edge do
          Apply the dominator rule to each vertex in $R'$ dominating its child. Let
          $R''$ be the path obtained. Apply the bipath rule to shrink $R''$ to length
          $O(1)$.
9.) Let $G'$ be the digraph obtained in Step 8. return $(G', k)$.
```

as incoming edges. Since there are no useless back edges after their deletion in Step 3 (Lemma 5), each vertex in S has only one incoming edge, which is a tree edge. If we consider the subpath Q' of Q induced by $S \cup \{w\}$, each vertex with a successor is a dominator. In Step 4, the if conditions are satisfied for each dominator v in Q', and we can remove v by applying the dominator rule at v. Afterwards the attachment of w is of size one. To sum up, each isolated treepath consists of a first vertex, essential vertices, and attachments of size one after Step 4.

We next consider subpaths of isolated treepaths. If u and v are vertices of the same isolated treepath Q and if the path from v to u in Q contains ≥ 2 essential vertices $w_1 \neq v$ and $w_2 \neq v$, we call a back edge (u,v) (Q-)*jumping.* Intuitively, such an edge jumps over a vertex that can be made a leaf if we traverse Q backwards—in Fig. 2, a z-rooted outbranching with ≥ 2 leaves needs a jumping back edge. A maximal subpath R of an isolated treepath $Q = (q_1, q_2, q_3, \ldots)$

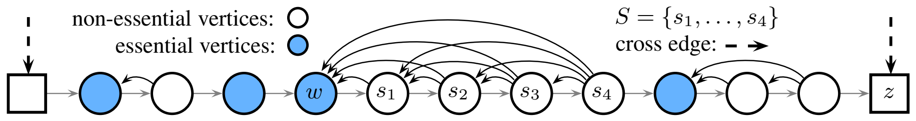


Fig. 1. An isolated treepath consisting of the round vertices with tree edges shown as straight arrows. The essential vertices define which back edges are useless. All possible useless edges are shown as curved arrows. In addition, each essential vertex is head of a useful but non-displayed back edge whose tail is for example z.

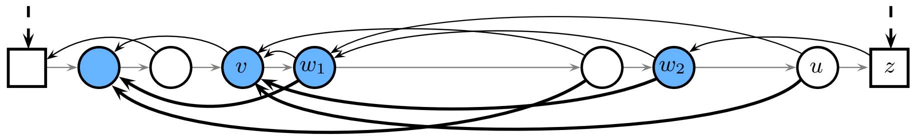


Fig. 2. Jumping (bold) and non-jumping back edges (thin)

without q_1 and without heads of Q-jumping back edges is called a *strongly isolated treepath of* (G, T)—by excluding q_1, R has only essential vertices and attachments of size one.

In Step 6, we need the following property for each strongly isolated treepath R: If a vertex v in R is dominating its child, all back edges $e = (v, x)$ are exiting R. To ensure that the property holds, in Step 5 the algorithm chooses a vertex y such that (y, v) is a back edge—such an edge must exist since v is essential and not the first vertex of an isolated treepath—and adds a jumping back edge $e' = (y, x)$ so that e becomes exiting (for more details see Lemma 6 and its proof). Note that e' is a back edge since (y, v) and e are back edges. In Step 6, for each strongly isolated treepath $R = (u_1, \dots, u_t)$ and each vertex w not in R, we remove all except 2 (cross or jumping back) edges of the form (u_i, w) $(1 \leq i \leq t)$ by applying the separator rule. Lemma 7a shows that the set S chosen by the algorithm is indeed a separator. In Step 7, we check several conditions that reveal an obvious yes instance. The correctness is shown in Section 6. In Step 8, we iterate over all strongly isolated treepaths $R = (u_1, \dots, u_t)$ and all maximal subpaths R' of $(u_2, \dots, u_t)$ without tails of exiting and jumping back edges. By Lemma 7b, the application of the dominator rule at each vertex with a non-empty attachment turns R' into a bipath, which is then shrunk by the bipath rule.

4 Running Time

A outtree T and the classification of the edges in Step 2 can be computed in linear time, i.e., linear in the number of vertices and edges. Recall that an edge (u, v) is useless if and only if v dominates u. Step 3 can also be executed in linear time since a dominator tree allows queries of the form "Is u a dominator of v?" to be answered in constant time and can be constructed in linear time [1,9]. In Step 4, the traversal can be done in linear time since the if condition of $(|N^-(v)| = 1)$

guarantees that we only have to replace each outgoing edge of v by an outgoing edge of its parent. In Step 5, by two scans over all edges, we can split all treepaths in isolated and subsequently in strongly isolated treepath. Using a dominator tree we can find each v that dominates its child. By iterating over all outgoing edges of v, we compute all non-exiting back edges. For each such edge, we add one new edge to G. Thus, the running time of this step is linear. Step 6 can be implemented in linear time by using radix sort to find the exiting edges of each strongly isolated treepath. We so know all vertices w for which the inner for loop is executed. In Step 8, the time to process each path R' is linear in the number of vertices of R' by Lemma 7b.

5 Properties of the Computation

We start with an auxiliary lemma. Many proofs are skipped.

Lemma 2. *Let $\tilde{G}$ be a digraph with an r-rooted outbranching $\tilde{T}$ such that $(\tilde{G}, \tilde{T})$ has no forward edges. Let $e = (p(v), v)$ be a tree edge with $N^-(v) = \{p(v)\}$ and assume that neither $p(v)$ nor v is a branching vertex. Applying the cutvertex rule at v in $\tilde{G}$ and in $\tilde{T}$ does not change the classification (back edge, etc.) of any edge $e' \notin \{(p(v), v), (v, p(v))\}$. Moreover, a useful back edge remains useful after the application of the cutvertex rule.*

Lemma 3. *(G, T) has no forward edges throughout Steps 1 to 7.*

Lemma 4. *Let G and T as in the algorithm after Step 1. Let w be an essential vertex whose attachment contains a vertex v. Then v has as incoming edges only one tree edge and useless back edges.*

Proof. Let (x, v) be an edge of G. By Lemma 3, (x, v) is not a forward edge. Assume that (x, v) is a cross edge. Since w is not a branching vertex, the path P from the root r to x in T cannot contain w. Taking P and the edge (x, v) we obtain a path Q from r to v that avoids w.

Assume that (x, v) is a useful back edge. Let P be a path from r to x avoiding v with a minimal number of cross and back edges. Note that w is not a vertex of P since otherwise, P would use a cross or back edge (w, y) to avoid v—y is consequently not a descendant of v—and the path from r to y consisting of tree edges combined with the subpath of P from y to x avoids v and has fewer cross and back edges than P. Thus, P and (x, v) defines a path Q from r to v that avoids w.

In both cases, v is not part of the attachment of w—contradiction. □

Lemma 5. *(G, T) has no useless back edges throughout Steps 4 to 7.*

Lemma 6. *After Step 5, if a vertex v of a strongly isolated treepath Q dominates its child, each back edge $e = (v, x)$ is exiting Q.*

Proof. The attachment of v is {child of v}. Since a vertex of a strongly isolated treepath is not the first vertex of an isolated treepath, recall that v is not dominated by its parent after Step 4. Thus, v has an incoming non-tree edge $e' = (y, v)$, which must be a non-jumping back edge since v is a vertex of a strongly isolated treepath.

Assume that e is not exiting. Then, e is a non-jumping back edge. See Fig. 3. In Step 5, all conditions regarding v are satisfied and the shortcut rule applied to v adds an edge (y, x). Thus, Q is no strongly isolated treepaths after Step 5. Contradiction, and e must be exiting. □

Lemma 7. *Let $R = (u_1, \ldots, u_t)$ be a strongly isolated treepath.*

a) *In Step 6, $S = \{u_{\sigma_1}, u_{\sigma_\ell}\}$ is a separator for $\{u_{\sigma_2}, \ldots, u_{\sigma_{\ell-1}}\}$ except if $u_{\sigma_{\ell-1}}$ dominates u_{σ_ℓ}, in which case $S = \{u_{\sigma_1}, u_{\sigma_{\ell-1}}\}$ is a separator for $\{u_{\sigma_2}, \ldots, u_{\sigma_{\ell-2}}, u_{\sigma_\ell}\}$.*

b) *If a subpath R' of R consists exclusively of vertices such that each vertex is not the tail of an exiting or jumping back edge, the application of the dominator rule to each vertex of R' that dominates its child turns R' into a bipath and can be done in time $O(|\text{vertices of } R'|)$.*

Proof. After Step 5, the distribution of the essential vertices of R fixes the back edges with both endpoints in R: Each essential vertex v of R is the head of a back edge e. Since e can neither be jumping nor useless and because of Lemma 6, the tail of e is the first essential vertex w after v in R if w does not dominate its child, and the only vertex in the attachment of w otherwise. Fig. 4 shows examples of all cases. To sum up, if repeated visits are forbidden, there is a unique way to visit the vertices of R if we start with the first (last) vertex of R. In the forward direction we can use only the tree edges. In the backward direction, we must use a tree edge at each vertex with a non-empty attachment, and a back edge otherwise.

a) Note that each path P from the root r to a vertex of Umust visit one endpoint $\tilde{u}$ of R. If we now consider the subpath of P starting in $\tilde{u}$, we can observe that P has to visit a vertex of S before it can reach $U = \{u_{\sigma_1}, \ldots, u_{\sigma_\ell}\} \setminus S$. In each of the two cases, S is a separator for U.

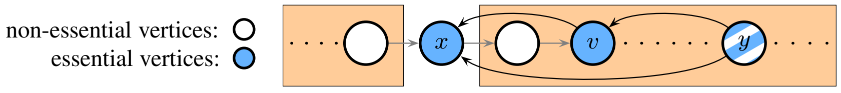


Fig. 3. A strongly isolated treepath (consisting of all vertices above) is split into two parts by Step 5, and (y, x) and (v, x) are exiting

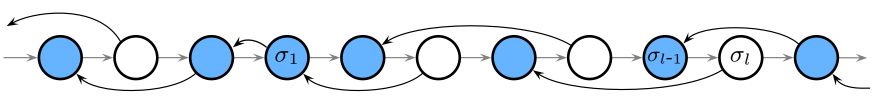


Fig. 4. A strongly isolated treepath after Step 5. Exiting edges are not shown.

b) Let us group the vertices of R'. Each vertex dominated by its parent builds a group with its parent. Every other vertex forms a group by itself. Then there is exactly one back edge from each group to the previous group. The application of the dominator rule contracts each group, which turns R' into a bipath, and this can be done in $O(1)$ time per group. □

6 Correctness of Step 7

If the first condition of Step 7 holds, the answer of the algorithm is correct. Thus, for the remaining analysis, we can assume that T has fewer than k branching vertices, and the number of maximal treepaths of (G, T) is less than $2k$.

The correctness of the second and the third condition of Step 7 is shown by Lemmata 8 and 9, respectively. The requirements of the lemmata are satisfied by Lemmata 3 and 5. Since we want to use Lemma 8 again in the proof of Lemma 10, we allow a certain kind of forward edges.

Lemma 8. *Let $\tilde{G}$ be a digraph, and let $\tilde{T}$ be an outbranching of $\tilde{G}$ with root r such that every forward edge of $(\tilde{G}, \tilde{T})$ has tail r. If no back edge of $(\tilde{G}, \tilde{T})$ is useless and if a treepath of $(\tilde{G}, \tilde{T})$ has $\geq 2k$ different heads of entering edges, then $\tilde{G}$ has a k-leaf outbranching.*

Lemma 9. *Let $\tilde{G}$ be a digraph with an outbranching $\tilde{T}$ such that $(\tilde{G}, \tilde{T})$ has no forward edges. If a vertex u of a maximal treepath P is the tail of $\geq k$ useful back edges that do not exit P, $\tilde{G}$ has a k-leaf outbranching.*

Assume that in Step 7 an isolated treepath P of (G, T) has $\geq 4k^2$ heads of jumping back edges. Let D be the set of essential vertices of (G, T) that have an attachment. Recall that the following property holds: each vertex v in D with its parent $p(v)$ in the same isolated treepath is not dominated by $p(v)$ after Step 4.

For the analysis, consider the digraph $\tilde{G}$, the outtree $\tilde{T}$, and the directed path $\tilde{P}$ obtained from G, T, and P, respectively, by contracting each vertex $v \in D$ with the only vertex in the attachment of v. By Lemmata 3 and 5, (G, T) has neither forward nor useless back edges. Lemma 2 shows that this property is maintained in $(\tilde{G}, \tilde{T})$. Moreover, each jumping back edge (u, v) of (G, T) is a jumping back edge of $(\tilde{G}, \tilde{T})$ since the number of essential vertices on the v-u-path is not changed by the contract operation. Also the number of heads of jumping back edges do not change. Let $v_1, \ldots, v_t$ be the vertices of $\tilde{P}$, and for each vertex v_i $(2 \leq i \leq t)$, denote by $p_{\tilde{T}}(v_i)$ its parent in $\tilde{T}$. By the property mentioned in the last paragraph, each vertex v_i $(2 \leq i \leq t)$ is not dominated by $p_{\tilde{T}}(v_i)$ in $\tilde{G}$, i.e., v_i is the head of a non-tree edge e. Since v_i cannot be the head of a cross edge and since no edges are useless, e is a useful back edge. By Lemma 10, $\tilde{G}$ has an outbranching with at least k leaves. It is easy to see by "undoing" the contraction from above that the outbranching for $\tilde{G}$ can be transformed into an outbranching for G without decreasing the number of leaves. Thus, the last condition of Step 7 is correct.

Lemma 10. *Let $\tilde{G}$ be a digraph with an outbranching $\tilde{T}$ such that $(\tilde{G}, \tilde{T})$ has no forward and no useless back edges. Let r be the root of $\tilde{T}$, and let $P = (v_1, \ldots, v_t)$ ($t \in I\!N$) be an isolated treepathsuch that each vertex v_i ($i \geq 2$) is the tail of a useful back edge. If P contains $\geq 4k^2$ heads of jumping back edges, $\tilde{G}$ has an r-rooted outbranching with $\geq k$ leaves.*

7 Kernel Size

Next, we want to analyze the size of the kernel returned in Step 9. For this purpose, it is interesting to bound the number of maximal subpaths of a strongly isolated treepath consisting exclusively of vertices that are not a tail of an exiting or jumping back edge.

Lemma 11. *After Step 7, the number of heads of entering edges of a maximal treepath of (G, T) is less than $2k$ and the number of exiting edges with a tail in a fixed isolated treepath of (G, T) is at most $44k^4$.*

Proof. T has fewer than k branching vertices by the first condition of Step 7. Thus, (G, T) has at most $2k$ maximal treepaths. Moreover, each treepath of (G, T) has fewer than $2k$ heads of entering edges due to the second condition of Step 7. Let z be the number of exiting edges with a tail in a fixed strongly isolated treepath R that is a subpath of a maximal treepath P. Because of Step 6, z is bounded by twice the number of heads of entering edges of the other maximal treepaths $P' \neq P$ plus the number of branching vertices since a head of an exiting edge of R is either a branching vertex or the head of an entering edge of some other maximal treepath $P' \neq P$. By the pigeon-hole principle R has fewer than $2(k + 2k \cdot 2k) \leq 10k^2$ tails of exiting edges. By the fourth condition of Step 7, each isolated treepath Q has fewer than $4k^2$ heads of jumping back edges, i.e., Q can be divided into at most $4k^2$ strongly isolated treepaths. Since each vertex of Q either belongs to a strongly isolated treepath or is one of the at most $4k^2$ remaining vertices, there are at most $4k^2 \cdot 10k^2 + 4k^2 = 44k^4$ exiting edges with a tail in Q. □

Lemma 12. *After Step 7, for an isolated treepath P, the number of heads and tails of jumping back edges is less than $4k^2$ and $32k^4$, respectively.*

Proof. By the fourth condition of Step 7, P can be divided into at most $4k^2$ strongly isolated treepaths. By Step 6, each vertex can be the head of two jumping back edges with a tail in the same strongly isolated treepath P' subpath of P. In total, each vertex can be the head of at most $8k^2$ jumping back edges. Since P has at most $4k^2$ vertices being a head of jumping back edge, P has at most $32k^4$ tails of jumping back edges. □

By the last two lemmata, the number of isolated treepaths is $k \cdot 2k = O(k^2)$ in total, and each such treepath has $44k^4 + 4k^2 + 32k^4 = O(k^4)$ tails of exiting edges and endpoints of jumping back edges. Thus, we have $O(k^6)$ directed paths R' in Step 8, which are shrunk to length $O(1)$. Since the number of vertices between two such path is $O(1)$, we have $O(k^6)$ vertices. Let $\mathcal{P}$ be the set of treepaths that are shrunk to length $O(1)$ in Step 8. Let T' be the outtree obtained from T by shrinking each directed path in $\mathcal{P}$ in the same way. T' is then an outbranching for G' without forward edges. Therefore, edges that are neither tree nor exiting edges must be back edges. (G', T') has $O(k)$ tree edges, $O(|\text{isolated treepaths}|) \cdot 44k^4 + |\text{vertices outside all isolated treepaths}|^2 = O(k^2) \cdot 44k^4 + O(k^2)^2 = O(k^6)$ exiting edges, and $O(k) \cdot O(k^6) = O(k^7)$ edges incident to the $O(k)$ branching vertices. By the third condition of Step 7, there are $< k$ back edges incident to a non-branching vertex. To sum up, G' has $O(k^7)$ edges.

Theorem 13. *In linear time, a kernel with $O(k^6)$ vertices and $O(k^7)$ edges can be found for the* ROOTED k-LEAF OUTBRANCHING PROBLEM.

References

1. Alstrup, S., Harel, D., Lauridsen, P.W., Thorup, M.: Dominators in linear time. SIAM J. Comput. 28, 2117–2132 (1999)
2. van Bevern, R.: Towards optimal and expressive kernelization for d-hitting set. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 121–132. Springer, Heidelberg (2012)
3. van Bevern, R., Hartung, S., Kammer, F., Niedermeier, R., Weller, M.: Linear-time computation of a linear problem kernel for dominating set on planar graphs. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 194–206. Springer, Heidelberg (2012)
4. Binkele-Raible, D., Fernau, H., Fomin, F.V., Lokshtanov, D., Saurabh, S., Villanger, Y.: Kernel(s) for problems with no kernel: On out-trees with many leaves. ACM Transactions on Algorithms 8(4), 38 (2012)
5. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25(6), 1305–1317 (1996)
6. Daligault, J., Thomassé, S.: On finding directed trees with many leaves. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 86–97. Springer, Heidelberg (2009)
7. Downey, R.G., Fellows, M.R.: Parameterized complexity. Springer (1999)
8. Hagerup, T.: Simpler linear-time kernelization for planar dominating set. In: Marx, D., Rossmanith, P. (eds.) IPEC 2011. LNCS, vol. 7112, pp. 181–193. Springer, Heidelberg (2012)
9. Harel, D.: A linear time algorithm for finding dominators in flow graphs and related problems. In: ACM Symp. on Theory of Computing (STOC), vol. 17, pp. 185–194 (1985)
10. Kloks, T., Lee, C.-M., Liu, J.: New algorithms for k-face cover, k-feedback vertex set, and k-disjoint cycles on plane and planar graphs. In: Kučera, L. (ed.) WG 2002. LNCS, vol. 2573, pp. 282–295. Springer, Heidelberg (2002)
11. Protti, F., Dantas da Silva, M., Szwarcfiter, J.: Applying modular decomposition to parameterized cluster editing problems. Theory of Computing Systems 44, 91–104 (2009)

On Retracts, Absolute Retracts, and Folds in Cographs

Ton Kloks[1] and Yue-Li Wang[2]

[1] Department of Computer Science
National Tsing Hua University, Taiwan
[2] Department of Information Management
National Taiwan University of Science and Technology
ylwang@cs.ntust.edu.tw

Abstract. Let G and H be two cographs. We show that the problem to determine whether H is a retract of G is NP-complete. We show that this problem is fixed-parameter tractable when parameterized by the order of H. When restricted to the class of threshold graphs or to the class of trivially perfect graphs, the problem becomes tractable in polynomial time. The problem is also solvable in linear time when one cograph is given as an induced subgraph of the other. We characterize absolute retracts for the class of cographs. Foldings generalize retractions. We show that the problem to fold a trivially perfect graph onto a largest possible clique is NP-complete. For a threshold graph this folding number equals its chromatic number and achromatic number.

Keywords: Retract, Absolute Retract, Fold, Cograph.

1 Introduction

Graph homomorphisms have regained a lot of interest by the recent characterization of Grohe of the classes of graphs for which $\text{Hom}(\mathcal{G}, -)$ is tractable [15]. To be precise, Grohe proves that, unless $FPT = W[1]$, deciding whether there is a homomorphism from a graph $G \in \mathcal{G}$ to some arbitrary graph H is polynomial if and only if the graphs in $\mathcal{G}$ have bounded treewidth modulo homomorphic equivalence. The treewidth of a graph modulo homomorphic equivalence is defined as the treewidth of its core, i.e., a minimal retract. This, and other recent results make it desirable to have algorithms that compute cores, or general retracts in graphs.

For any graph G, all the cores of G are isomorphic subgraphs of G. Therefore, one speaks of *the core* of a graph. However, a fixed copy of the core in G is not necessarily a retract. Therefore, when studying retracts or cores one usually assumes that the objective is given as an induced subgraph of G. When H is given as an induced subgraph of cograph G, it can be determined in linear time whether H is a retract. We prove this in Section 5. In the rest of the paper we do not assume that the graph H is given as an induced subgraph of G. In that case the problem turns out to be NP-complete. We prove that in Section 4.

In this paper we consider the retract problem for cographs. The related surjective graph homomorphism problem was recently studied in [12] in which it was shown that the problem to decide whether there is a surjective homomorphism from one connected cograph to another connected cograph is NP-complete. The surjective homomorphism problem is also NP-complete of the set of proper interval graphs and linear forests in addition to the union of complete graphs. Let us mention also the classic result of Damaschke, which is that the induced subgraph isomorphism problem is NP-complete for cographs [7].

The retract problem for cographs can be perceived as a pattern recognition problem for labeled trees. Many pattern recognition variants have been investigated and classified. However, the pattern recognition problem that corresponds with the retract problem on cographs seems to have eluded all these investigations [16]. For basic terminology on graph homomorphisms we refer to [17,20].

Definition 1. *Let G and H be graphs. A homomorphism $\phi : G \to H$ is a map $\phi : V(G) \to V(H)$ which preserves edges, that is,*

$$\{x, y\} \in E(G) \quad \Rightarrow \quad \{\phi(x), \phi(y)\} \in E(H).$$

We write $G \to H$ if there is a homomorphism $\phi : G \to H$. Notice that

$$G \to K_k \quad \Leftrightarrow \quad \chi(G) \leq k \quad \text{and also that} \quad K_k \to G \quad \Leftrightarrow \quad \omega(G) \geq k,$$

where $\chi(G)$ and $\omega(G)$ denote the chromatic and clique, respectively, numbers of G.

Definition 2. *Let G and H be graphs. The graph H is a retract of G if there exist homomorphisms $\rho : G \to H$ and $\gamma : H \to G$ such that $\rho \circ \gamma = id_H$, which is the identity map $V(H) \to V(H)$. The functions ρ and γ are called the retraction and co-retraction, respectively.*

When H is a retract of G then H is isomorphic to an induced subgraph of G. Since there are homomorphisms in two directions, G and H have the same clique number, chromatic number and odd girth. Also, there is a retraction from G to K_k if and only if $\chi(G) = \omega(G) = k$.

There is a homomorphism $G \to H$ if and only if the union of G and H retracts to H. When H is a fixed pseudoforest, i.e., each connected component has at most one cycle, given as an induced subgraph of G, the complexity of the retract problem has been classified in [9]. If H is bipartite or contains a loop, then checking if there is a homomorphism $G \to H$ is polynomial and otherwise it is NP-complete when H is fixed [20]. It follows that, for any graph H, checking if a graph H is a retract of a graph G is NP-complete, unless H is bipartite. The problem remains NP-complete, even when H is an even cycle of length at least six, given as an induced subgraph of G. The question whether a graph G has a homomorphism to itself which is not the identity is also NP-complete.

Remark 1. No connected graph has a disconnected retract since the homomorphic image of a connected graph is connected. To see that, notice that a homomorphism $\phi : G \to H$ is a vertex coloring of G, where the vertices of H represent colors. By that we mean that, for each $v \in V(H)$, the pre-image $\phi^{-1}(v)$ is an independent set in G or $\varnothing$. One obtains the image $\phi(G)$ by identifying vertices in G that receive the same color. When G is connected, this 'quotient graph' on the color classes is also connected, which is easy to prove by means of contradiction.

Definition 3 ([5]). *A graph is a cograph if it has no induced path of four vertices, denoted P_4.*

Definition 4 ([13,25]). *A graph G is trivially perfect if for all induced subgraphs H of G, $\alpha(H)$ is equal to the number of maximal cliques in H, where $\alpha(H)$ is the stability number of H. Trivially perfect graphs are those graphs without induced C_4 and P_4, where C_4 is a cycle of four vertices.*

Definition 5 ([4]). *Threshold graphs are the graphs without induced $2K_2$, C_4 and P_4, where $2K_2$ is a pair of K_2.*

By above definitions, the class of threshold graphs is a subclass of the class of trivially perfect graphs which is a subclass of the class of cographs. Since the complement of a P_4 is a P_4, cographs are closed under complementation. Actually, the class of cographs is the smallest class of graphs which is closed under complementation and taking unions.

A similar characterization of cographs reads as follows. A graph G is a cograph if and only if one of the following holds [5].

(1) G has only one vertex, or
(2) G is disconnected and every component is a cograph, or
(3) the complement of G, $\bar{G}$ is disconnected and every component of $\bar{G}$ is a cograph.

It follows that cographs have a decomposition tree, called a cotree, defined as follows. The decomposition tree is a rooted tree T. There is a bijection from the leaves of T to the vertices of G. When G has at least two vertices then each internal node of T, including the root, is labeled as $\otimes$ or $\oplus$. The $\oplus$ label at a node takes the union of the graphs that correspond with the children of the node. The $\otimes$ label takes the join of the graphs that correspond with the children.

Remark 2. When defined as above, the labels of the internal nodes in any path from the root to a leaf alternate between $\oplus$ and $\otimes$. Alternatively, one frequently defines a cotree as a rooted *binary* tree, in which each internal node is labeled as $\oplus$ and $\otimes$. In this paper, when talking about cotrees, we always assume the first type of cotree. Thus, each child of the root corresponds with one component or, with one cocomponent of the graph.

Remark 3. There is a linear-time cograph recognition algorithm which also produces the cotree on recognizing a cograph [5]. A cotree has $O(n)$ nodes, where $n = |V(G)|$.

This paper is organized as follows. In Sections 2 and 3 we show that the retract problem is polynomial when restricted to the classes of threshold and trivially perfect graphs. In Section 4 we show that the problem is NP-complete for cographs. In Section 5 we show that, when H is given as an induced subgraph of G, it can be determined in polynomial time whether H is a retract of G. In Section 6 we show that the retract problem for cographs is fixed-parameter tractable. In Section 7 we show that computing the folding number is NP-complete for trivially perfect graphs. For threshold graphs the folding number equals the chromatic and achromatic number. Absolute retracts for the class of cographs are discussed in Section 8.

2 Retracts in Threshold Graphs

We use the following characterization of threshold graphs.

Theorem 1. *A graph is a threshold graph if and only if every induced subgraph has a universal vertex or an isolated vertex, where a universal vertex is a vertex of degree $n-1$.*

Theorem 2. *Let G and H be threshold graphs. There exists a linear-time algorithm to check if H is a retract of G.*

Proof. Assume that H is a retract of G and let ρ and γ be the retraction and co-retraction. Assume that G has a universal vertex, say x_1. Then H must have a universal vertex as well, since a retract of a connected graph is connected. Let y_1 be a universal vertex of H. Let $y_i = \rho(x_1)$. Since ρ is a homomorphism it preserves edges, and since x_1 is universal in G, ρ maps no other vertex of G to y_i. Notice also that $\gamma(y_i) = x_1$ since $\rho \circ \gamma = id_H$ and ρ maps no other vertex to y_i. Assume that $y_i \neq y_1$. Let $\gamma(y_1) = x_\ell$. Then $x_\ell \neq x_1$ since γ preserves edges. Furthermore, since y_1 is universal, γ maps no other vertex of H to x_ℓ. Of course, since $\rho \circ \gamma = id_H$, $\rho(x_\ell) = y_1$.

We claim that y_i is universal in H, and therefore exchangeable with y_1. Assume not and let $y_s \in V(H)$ be another vertex of H not adjacent to y_i. Let $\gamma(y_s) = x_p$. Then $x_p \neq x_1$ since $\rho \circ \gamma = id_H$ and $\rho(x_1) = y_i \neq y_s$. Now, since ρ is a homomorphism and since x_1 is universal,

$$\{x_1, x_p\} \in E(G) \quad \Rightarrow \quad \{\rho(x_1), \rho(x_p)\} = \{y_i, y_s\} \in E(H),$$

which is a contradiction. Therefore, we may assume that $y_i = y_1$. That is, from now on we assume that

$$\rho(x_1) = y_1 \quad \text{and} \quad \gamma(y_1) = x_1.$$

This proves that, when G is connected then H is a retract of G if and only if $H - y_1$ is a retract of $G - x_1$. By the way, notice that if $|V(H)| = 1$ then H can be a retract of G only if G is an independent set, so this case is easy to check.

Finally, assume that G is not connected. Since G has no induced $2K_2$, all components, except possibly one, have only one vertex. The number of components of H can be at most equal to the number of components of G, since ρ maps components in G to components of H, and $\rho \circ \gamma = id_H$, and so any two components of H are mapped by γ to different components of G.

First assume that H is also disconnected. Without loss of generality, let $x_1, \ldots, x_a$ be the isolated vertices of G and let $y_1, \ldots, y_b$ be the isolated vertices of H. If H is a retract of G, then H is an induced subgraph of G which implies $a \geq b$; otherwise, H is not a retract of G. Now, H is a retract of G if and only if $H - \{y_1, \ldots, y_b\}$ is a retract of $G - \{x_1, \ldots, x_a\}$.

If H is connected, with at least two vertices, then let y_1 be a universal vertex. If H is a retract of G then G must have exactly one component with at least two vertices, since G is a threshold graph and ρ is a homomorphism. Let x_u be the universal vertex of the component and $x_1, \ldots, x_a$ be the isolated vertices in other components. In this case, H is a retract if and only if $H - y_1$ is a retract of $G - \{x_1, \ldots, x_a, x_u\}$.

An elimination ordering, which eliminates successive isolated and universal vertices in a threshold graph, can be obtained in linear time. This proves the theorem. □

Remark 4. Co-trivially perfect graphs are the graphs without induced $2K_2$ and P_4. The retract problem is polynomial for this class via matching since a co-trivially perfect graph is the join of a bunch of threshold graphs.

3 Retracts in Trivially Perfect Graphs

Theorem 3 ([25]). *A graph is trivially perfect if and only if every connected induced subgraph has a universal vertex.*

Theorem 4. *Let G and H be trivially perfect graphs. There exists an $O(N^{5/2})$ algorithm which checks if H is a retract of G, where $N = |V(G)| \cdot |V(H)|$.*

Proof. Assume that H is a retract of G. Let $C_1, \ldots, C_t$ be the components of G and let $D_1, \ldots, D_s$ be the components of H. Then $s \leq t$; otherwise, H is not a retract of G. Without loss of generality, let D_i be a retract of C_i for $i \in \{1, \ldots, s\}$. For the components C_i with $i > s$, there must be a $j \leq s$ such that there is a homomorphism from C_i to D_j.
First assume that G and H are connected. Let $g_1, \ldots, g_k$ be the universal vertices of G and let $h_1, \ldots, h_\ell$ be the universal vertices of H. As in the proof of Theorem 2 it follows that H is a retract of G if and only if

(i) $k \leq \ell$, and
(ii) either H is a clique and $\omega(G) = \omega(H)$ or $H - \{h_1, \ldots, h_k\}$ is a retract of $G - \{g_1, \ldots, g_k\}$.

For the general case, consider the following bipartite graph B. The vertices of B are the components of G and H. There is an edge $\{C_i, D_j\} \in E(B)$ if and only if C_i retracts to D_j. Then G retracts to H if and only if

(a) B has a matching which exhausts all components of H, and
(b) for every component C_i which is not an endpoint of an edge in the matching there is a D_j such that there is a homomorphism from $G[C_i]$ to $H[D_j]$.

To check if a component $G[C_i]$ retracts to some $H[D_j]$ the algorithm greedily matches the universal vertices of $G[C_i]$ and $H[D_j]$ and checks if the remaining graph G', i.e., after removal of the matched universal vertices, retracts to the remaining graph H'. Let $C_i^1, \ldots, C_i^p$ and $D_j^1, \ldots, D_j^q$ be the components of G' and H', respectively. After constructing the bipartite graph B_{ij} on the components C_i^k and D_j^ℓ, where $k \in \{1, \ldots, p\}$ and $\ell \in \{1, \ldots, q\}$, the algorithm checks if there is an edge $(C_i^k, D_j^\ell) \in E(B_{ij})$ in $O(1)$ time by table look-up, and so the bipartite graph B_{ij} is constructed in

$$O(pq) = O(|C_i| \cdot |D_j|).$$

Edmonds' algorithm [8] computes a maximum matching in B_{ij} in time

$$O((p+q)^{5/2}) = O((|C_i| + |D_j|)^{5/2}).$$

Summing over the components C_i and D_j, for $i \in \{1, \ldots, t\}$ and $j \in \{1, \ldots, s\}$, we obtain

$$\sum_{i=1}^{t} \sum_{j=1}^{s} (|C_i| \cdot |D_j| + (|C_i| + |D_j|)^{5/2}) = O(|V(G)|^{5/2} \cdot |V(H)|^{5/2}).$$

This proves the claim. □

4 NP-Completeness of Retracts in Cographs

Recall that a graph G is perfect when $\omega(G') = \chi(G')$ for every induced subgraph G' of G. By the perfect graph theorem a graph is perfect if and only if it has no odd hole or odd antihole. This implies that cographs are perfect. Perfect graphs are recognizable in polynomial time. For a graph G, when $\omega(G) = \chi(G)$ one can compute this value in polynomial time via Lovász theta function.

Lemma 1 ([10]). *Assume that $\omega(H) = \chi(H)$. There is a homomorphism $G \to H$ if and only if $\chi(G) \leq \omega(H)$.*

Corollary 1. *When G and H are perfect one can check in polynomial time whether there is a homomorphism $G \to H$.*

It is well-known that retracts, like general homomorphisms, constitute a transitive relation. We provide a short proof for completeness sake.

Lemma 2. *Let A be a retract of G and let B be a retract of A. Then B is a retract of G.*

Proof. Let ρ_1 and γ_1 be a retraction and co-retraction from G to A and let ρ_2 and γ_2 be a retraction and co-retraction from A to B. Since all four maps ρ_1, ρ_2, γ_1 and γ_2 are homomorphisms, the following two maps are homomorphisms as well.

$$\rho_2 \circ \rho_1 : G \to B \quad \text{and} \quad \gamma_1 \circ \gamma_2 : B \to G.$$

Furthermore,

$$(\rho_2 \circ \rho_1) \circ (\gamma_1 \circ \gamma_2) = \rho_2 \circ id_A \circ \gamma_2 = \rho_2 \circ \gamma_2 = id_B.$$

This proves that B is a retract of G. □

Throughout the remainder of this section it is assumed that G and H are cographs. Note that, using the cotree, $\omega(G)$ and $\chi(G)$ can be computed in linear time when G is a cograph.

Lemma 3. *Assume that H is a retract of a graph G, where H is disconnected with components $H_1, \ldots, H_t$. Then there is an ordering of the components of G, say $G_1, \ldots, G_s$ such that*

(a) *$s \geq t$, and*
(b) *G_i retracts to H_i, for every $i \in \{1, \ldots, t\}$, and*
(c) *for every $j \in \{t+1, \ldots, s\}$, there is a homomorphism $G_j \to H$.*

Proof. Assume that G retracts to H. Then we may assume that $H_1, \ldots, H_t$ are induced subgraphs of components $G_1, \ldots, G_t$ of G and that each G_i retracts to H_i. For the remaining components G_j, where $j > t$, there is a homomorphism $G_j \to H$.

Notice that, for $j > t$, we can check if there is a homomorphism $G_j \to H$ by checking if $G_j \oplus H_k$ retracts to H_k, for some $1 \leq k \leq t$ or, equivalently (since cographs are perfect), if $\omega(G_j) \leq \omega(H_k)$ for some $1 \leq k \leq t$. □

Remark 5. Assume that we are given, for each pair G_i and H_j whether G_i retracts to H_j or not. Then, to check if G retracts to H, we may consider a bipartite graph B defined as follows. One color class of B has the components of G as vertices and the other color class has the components of H as vertices. There is an edge between G_i and H_j whenever G_i retracts to H_j. To check if G retracts to H, we can let an algorithm compute a maximum matching in B. There is a retraction only if the matching exhausts all components of H *and* if $\omega(G) = \omega(H)$.

A cocomponent of a graph G is a subset of vertices which induces a component of the complement $\bar{G}$.

Lemma 4. *Assume G is connected and assume that G retracts to H. Then H is also connected. Let $G_1, \ldots, G_t$ be the subgraphs of G induced by the cocomponents of G. Then there is a partition of the cocomponents of H such that the subgraphs of H induced by the parts of the partition, can be ordered $H_1, \ldots, H_t$ such that G_i retracts to H_i for $i \in \{1, \ldots, t\}$.*

Proof. Every subgraph G_i of G, induced by a cocomponent, retracts to some induced subgraph. These retracts are pairwise joined, so each part is the join of some subgraphs induced by cocomponents of H. Thus the parts of $V(H)$ that are the images of the subgraphs induced by cocomponents of G form a partition of the cocomponents of H. □

Theorem 5. *Let G and H be cographs. The problem to decide whether H is a retract of G is NP-complete.*

Proof. We reduce the 3-partition problem to the retract problem on cotrees. The 3-partition problem is the following. Let m and B be integers. Let S be a multiset of $3m$ positive integers, $a_1, \ldots, a_{3m}$. Determine if there is a partition of S into m subsets $S_1, \ldots, S_m$, such that the sum of the numbers in each subset is B. Without loss of generality we assume that $B/4 \leq a_i \leq B/2$ for $1 \leq i \leq 3m$, which guarantees that in a solution each subset contains exactly three numbers that add up to B.

The 3-partition problem is strongly NP-complete [11], that is, the problem remains NP-complete when all the numbers in the input are represented in unary. In our reduction, the cotree for the graph H has a root which is labeled as a join-node $\otimes$ (see Figure 1 for an illustration). The root has $3m$ children, one for each number a_i. For simplicity we refer to the children as a_i, $i \in \{1, \ldots, 3m\}$. Each child a_i has a union node $\oplus$ as the root. The root of each a_i-child has two children, one is a single leaf and the other is a join-node $\otimes$ with a_i leaves. This ends the description of H.

The cotree for the graph G has a join-node $\otimes$ as a root which has m children. The idea is that each child corresponds with one set of a 3-partition of S. The subtrees for all the children are identical. Each subtree has a union-node $\oplus$ as the root. Consider all triples $\{i, j, k\}$ for which $a_i + a_j + a_k = B$. For each such triple create one child, which is the join of three cotrees for a_i, a_j and a_k in the triple. The subtree for a_i is a union of two subtrees. As in the cotree for the pattern H, one subtree is a single leaf, and the other subtree is the join of a_i leaves. The other two subtrees, for the numbers a_j and a_k in the triple are constructed similarly.

Let T_H and T_G be the cotrees for H and G as constructed above. Say T_H and T_G have roots r_H and r_G. When the graph H is a retract of G then the a_i-children of r_H are partitioned into triples, such that there is a bijection between these triples, say $\{a_i, a_j, a_k\}$ and a branch in the cotree of G. Each $\oplus$-node which is the root of a child of r_G must have exactly one $\{a_i, a_j, a_k\}$-child that corresponds with the triple. Notice that, by the construction, all subgraphs induced by remaining components of the $\oplus$-node have maximal cliques of size B. Therefore, all other children of the $\oplus$-node are homomorphic to the one child which corresponds to the triple $\{a_i, a_j, a_k\}$.

It now follows from Lemma 4 that there is a 3-partition if and only if the graph H is a retract of G. This completes the proof. □

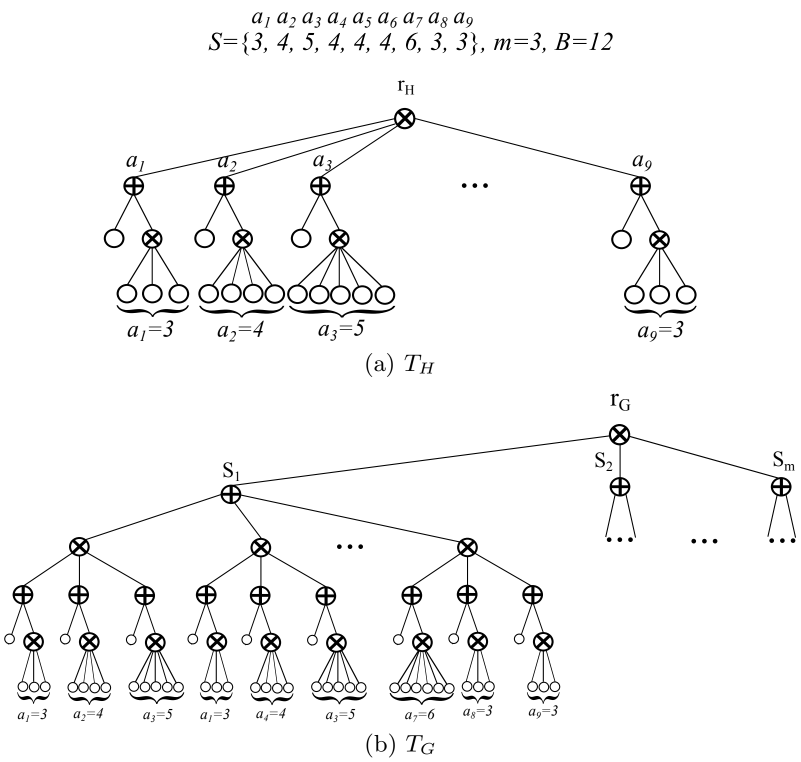


Fig. 1. The cotrees for G and H used in the proof of Theorem 5

5 The Partitioned Case for Retracts in Cographs

Theorem 6. *Let G and H be cographs and assume that H is given as an induced subgraph of G. There exists a linear-time algorithm to test if G retracts to H.*

Proof. We describe the algorithm. Construct a cotree for the graph G. Repeatedly, remove children of $\oplus$-nodes for which

(a) the branch has no leaves corresponding with vertices in H, and
(b) the subgraph induced by the branch has clique number at most equal to the clique number of a sibling.

When the algorithm ends such that all remaining vertices are in H then G retracts to H and otherwise it does not.

For brevity we omit the proof of correctness (which is straightforward). □

6 A Fixed-Parameter Solution for Retracts in Cographs

In this section we look at a parameterized solution for the retract problem. The reader is referred to [23] for the details of fixed-parameter algorithms. Let G and

H be cographs. We consider the parameterization by the number of vertices in H. Let

$$k = |V(H)|.$$

Proposition 1. *When H is a retract of G then $\omega(G) = \omega(H) \leq k$. Let T_G be a cotree for G. Then every join-node in T_G has at most k children, and the height of the cotree is $O(k)$.*

Lemma 5. *The retract problem, which asks if a cograph H is a retract of G, is fixed-parameter tractable when parameterized by the number of vertices in H.*

Proof. See arXiv:1301.3979.

□

Proposition 2. *For every H, the H-retract problem can be formulated in monadic second-order logic (without quantification over subsets of edges).*

By Courcelle's theorem [6] we may also conclude the following.

Corollary 2. *There exists a function $f : \mathbb{N} \to \mathbb{N}$ such that, for every H, say with $k = |V(H)|$, there is an $O(f(k) \cdot n)$ algorithm which checks if H is a retract of a cograph G.*

7 Foldings

Definition 6. *Let $G = (V, E)$ be a graph and let x and y be two vertices in G that are at distance two. A simple fold with respect to x and y is the operation which identifies x and y and eliminates duplicate edges. A folding is a homomorphism which is a sequence of simple folds.*

When $G \to H$ is a folding then we say that G folds onto H. It is well-known that any retraction of a connected graph is a folding, see e.g., [17, Proposition 2.19].

Definition 7. *The folding number $\Sigma(G)$ of a connected graph G is the largest number s such that G folds onto K_s. When G is disconnected the folding number is the maximal folding number of the graphs induced by the components of G.*

Recall that the achromatic number $\Psi(G)$ of a graph G is the largest number of colors with which one can properly color the vertices of G such that for any two colors there are two adjacent vertices that have those colors.

Lemma 6. *Assume that G has a universal vertex u. Then*

$$\Sigma(G) = 1 + \Psi(G - u) = \Psi(G).$$

Proof. Any two nonadjacent vertices of $G - u$ are at distance two in G. Thus any achromatic coloring of G is a folding. The universal vertex must be in a color class by itself. Harary and Hedetniemi [18] show that, when G is the join of two graphs G_1 and G_2 then $\Psi(G) = \Psi(G_1) + \Psi(G_2)$. This proves the lemma. □

Notice that the achromatic number problem is NP-complete, even for trees. However, the problem is fixed-parameter tractable [22]. The image of a tree after a simple fold is a tree. Therefore, the folding number of a tree is at most two.

Theorem 7. *The problem to compute the folding number is NP-complete, even when restricted to trivially perfect graphs.*

Proof. Bodlaender shows in [3] that computing the achromatic number is NP-complete, even when restricted to trivially perfect graphs. Since the class of trivially perfect graphs is closed under adding universal vertices, by Lemma 6 computing the folding number is NP-complete for trivially perfect graphs. □

Theorem 8. *When G is a threshold graph then*

$$\chi(G) = \Sigma(G) = \Psi(G).$$

Proof. When G is the join of two graphs G_1 and G_2 then

$$\Psi(G) = \Psi(G_1) + \Psi(G_2).$$

Assume that G has an isolated vertex x. In any achromatic coloring, the vertex must have a color that is used by another vertex also. Therefore,

$$\Psi(G) = \max \{ 1, \Psi(G - x) \}.$$

This proves the theorem. □

8 Absolute Retracts for Cographs

Definition 8. *Let $\mathcal{G}$ be a class of graphs. A graph H is an absolute retract for $\mathcal{G}$ if H is a retract of a graph $G \in \mathcal{G}$ whenever G is an isometric embedding of H and $\chi(H) = \chi(G)$.*

Hell, in his Ph.D. thesis, characterized absolute retracts for the class of bipartite graphs as the retracts of components of categorical products of paths [19]. Pesch and Poguntke characterized absolute retracts of k-chromatic graphs [24]. Their characterization can be strengthened for the case of bipartite graphs such that it leads to a polynomial recognition algorithm for absolute retracts of bipartite graphs [2]. Examples of absolute retracts of bipartite graphs are the chordal bipartite graphs [14]. Median graphs are exactly the absolute retracts of hypercubes [1]. For reasons of brevity we leave out the mention of all results on reflexive graphs.

Theorem 9. *Let H be a connected cograph. Then H is an absolute retract for the class of cographs if and only if every vertex of H is in a maximal clique of cardinality $\omega(H)$.*

Proof. See arXiv:1301.3979. □

References

1. Bandelt, H.: Retracts of hypercubes. Journal of Graph Theory 8, 501–510 (1984)
2. Bandelt, H., Dählmann, A., Schütte, H.: Absolute retracts of bipartite graphs. Discrete Applied Mathematics 16, 191–215 (1987)
3. Bodlaender, H.: Achromatic number is NP-complete for cographs and interval graphs. Information Processing Letters 31, 135–138 (1989)
4. Chvátal, V., Hammer, P.: Aggregation of inequalities in integer programming. Technical Report STAN-CS-75-518, Stanford University, California (1975)
5. Corneil, D., Perl, Y., Stewart, L.: A linear recognition algorithm for cographs. SIAM Journal on Computing 14, 926–934 (1985)
6. Courcelle, B.: The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Information and Computation 85, 12–75 (1990)
7. Damaschke, P.: Induced subgraph isomorphism for cographs is NP-complete. In: Möhring, R.H. (ed.) WG 1990. LNCS, vol. 484, pp. 72–78. Springer, Heidelberg (1991)
8. Edmonds, J.: Paths, trees, and flowers. Canadian Journal of Mathematics 17, 449–467 (1965)
9. Feder, T., Hell, P., Jonsson, P., Krokhin, A., Nordh, G.: Retractions to pseudoforests. SIAM Journal on Discrete Mathematics 24, 101–112 (2010)
10. Fomin, F., Heggernes, P., Kratsch, D.: Exact algorithms for graph homomorphisms. Theory of Computing Systems 41, 381–393 (2007)
11. Garey, M., Johnson, D.: Computers and intractability: a guide to the theory of NP-completeness. Freeman (1979)
12. Golovach, P., Lidický, B., Martin, B., Paulusma, D.: Finding vertex-surjective graph homomorphisms. Acta Informatica 49, 381–394 (2012)
13. Golumbic, M.: Trivially perfect graphs. Discrete Mathematics 24, 105–107 (1978)
14. Golumbic, M., Goss, C.: Perfect elimination and chordal bipartite graphs. Journal of Graph Theory 2, 155–163 (1978)
15. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. Journal of the ACM 54 (2007)
16. Grohe, M.: Personal communication
17. Hahn, G., Tardif, C.: Graph homomorphisms: structure and symmetry. In: Hahn, G., Sabidussi, G. (eds.) Graph Symmetry – Algebraic Methods and Applications. NATO ASI Series C: Mathematical and Physical Sciences, vol. 497, pp. 107–166. Kluwer (1997)
18. Harary, F., Hedetniemi, S.: The achromatic number of a graph. Journal of Combinatorial Theory 8, 154–161 (1970)
19. Hell, P.: Rétractions de graphes. PhD Thesis, Université de Montréal (1972)
20. Hell, P., Nešetřil, J.: Graphs and homomorphisms. Oxford University Press (2004)
21. Howorka, E.: A characterization of distance-hereditary graphs. The Quarterly Journal of Mathematics 28, 417–420 (1977)
22. Máté, A.: A lower estimate for the achromatic number of irreducible graphs. Discrete Mathematics 33, 171–183 (1981)
23. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
24. Pesch, E., Poguntke, W.: A characterization of absolute retracts of n-chromatic graphs. Discrete Mathematics 57, 99–104 (1985)
25. Wolk, E.: A note on "The comparability graph of a tree". Proceedings of the American Mathematical Society 16, 17–20 (1965)

Coloring Triangle-Free Rectangular Frame Intersection Graphs with $O(\log\log n)$ Colors*

Tomasz Krawczyk[1], Arkadiusz Pawlik[1], and Bartosz Walczak[1,2]

[1] Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University
{krawczyk,pawlik,walczak}@tcs.uj.edu.pl
[2] École Polytechnique Fédérale de Lausanne
bartosz.walczak@epfl.ch

Abstract. Recently, Pawlik et al. have shown that triangle-free intersection graphs of line segments in the plane can have arbitrarily large chromatic number. Specifically, they construct triangle-free segment intersection graphs with chromatic number $\Theta(\log\log n)$. Essentially the same construction produces $\Theta(\log\log n)$-chromatic triangle-free intersection graphs of a variety of other geometric shapes—those belonging to any class of compact arc-connected subsets of $\mathbb{R}^2$ closed under horizontal scaling, vertical scaling, and translation, except for axis-aligned rectangles. We show that this construction is asymptotically optimal for the class of rectangular frames (boundaries of axis-aligned rectangles). Namely, we prove that triangle-free intersection graphs of rectangular frames in the plane have chromatic number $O(\log\log n)$, improving on the previous bound of $O(\log n)$. To this end, we exploit a relationship between off-line coloring of rectangular frame intersection graphs and on-line coloring of interval overlap graphs. Our coloring method decomposes the graph into a bounded number of subgraphs with a tree-like structure that "encodes" strategies of the adversary in the on-line coloring problem, and colors these subgraphs with $O(\log\log n)$ colors using a combination of techniques from on-line algorithms (first-fit) and data structure design (heavy-light decomposition).

1 Introduction

A *proper coloring* of a graph is an assignment of colors to the vertices of the graph such that no two adjacent ones are in the same color. The minimum number of colors sufficient to color a graph G properly is called the *chromatic number* of G and denoted by $\chi(G)$. The maximum size of a clique (a set of pairwise adjacent vertices) in a graph G is called the *clique number* of G and denoted by $\omega(G)$. It is clear that $\chi(G) \geqslant \omega(G)$. The classes for which there is a function $f : \mathbb{N} \to \mathbb{N}$ such that $\chi(G) \leqslant f(\omega(G))$ holds for any graph G from the class are called

* T. Krawczyk and A. Pawlik were supported and B. Walczak was partially supported by the MNiSW grant no. 884/N-ESF-EuroGIGA/10/2011/0 as part of ESF EuroGIGA project GraDR. B. Walczak was partially supported by Swiss National Science Foundation Grant no. 200020-144531.

χ-bounded. A *triangle* is a clique of size 3. A graph is *triangle-free* if it does not contain any triangle.

It has been observed in the 1940s that the existence of large cliques is not necessary for the chromatic number to grow. Various classical constructions show that it can be arbitrarily large even for triangle-free graphs. The first such construction was given by Zykov [1], and the one perhaps best known is due to Mycielski [2]. Kim [3] constructed triangle-free graphs with chromatic number $\Theta(\sqrt{n/\log n})$, which is tight as shown by Ajtai, Komlós and Szemerédi [4].

In this paper, we focus on the relation between the chromatic number and the number of vertices of a graph for classes of triangle-free graphs arising from geometry. The *intersection graph* of a family of sets $\mathcal{F}$ is the graph with vertex set $\mathcal{F}$ and edge set consisting of pairs of intersecting elements of $\mathcal{F}$. For simplicity, we identify the family $\mathcal{F}$ with its intersection graph.

The study of the chromatic number of graphs with geometric representation was initiated in the seminal paper of Asplund and Grünbaum [5], where they proved that the families of axis-aligned rectangles are χ-bounded. In particular, they proved a tight bound of 6 on the chromatic number of triangle-free families of axis-aligned rectangles. Gyárfás [6,7] proved that the class of *overlap graphs*, that is, graphs represented by closed intervals on the line so that edges are pairs of intervals that intersect but are not nested, is χ-bounded. In contrast, Burling [8] showed that triangle-free intersection graphs of axis-aligned boxes in $\mathbb{R}^3$ can have arbitrarily large chromatic number. Pawlik et al. [9,10] provided a construction of triangle-free families of segments and, more generally, triangle-free families of vertically and horizontally scaled translates of any fixed arc-connected compact set in $\mathbb{R}^2$ that is not an axis-aligned rectangle, with arbitrarily large chromatic number. These families require $\Omega(\log\log n)$ colors, where n is the size of the family. One of the problems posed in [9] is to determine (asymptotically) the maximum chromatic number that a triangle-free family of n segments can have.

We solve the analogous problem for triangle-free families of *frames*, that is, boundaries of axis-aligned rectangles in the plane, showing that the construction of Pawlik et al. is asymptotically best possible.

Theorem 1. *Every triangle-free family of n frames can be properly colored with $O(\log\log n)$ colors.*

Note that the intersection graph of a family of frames is not the same as the intersection graph of the rectangles enclosed by these frames. Specifically, two frames intersect if and only if their corresponding rectangles intersect but are not nested. Frame intersection graphs can thus be considered as two-dimensional generalizations of overlap graphs.

Theorem 1 provides the first asymptotically tight bound on the chromatic number for a natural class of geometric intersection graphs that does not allow a constant bound. So far, best upper bounds were of order $O(\log n)$, following from the results of McGuinness [11] or Suk [12] for families of shapes including segments and frames, or polylogarithmic on n, obtained by Fox and Pach [13] for families of curves with bounded clique number and bounded number of pairwise intersections. Recently, Fox and Pach [14] established polylogarithmic upper bounds for arbitrary families of curves with bounded clique number. The only known lower bounds follow from the above-mentioned constructions of Burling and Pawlik et al. We hope that our ideas will lead to improving the bounds for other important classes, in particular, for segment intersection graphs.

On-line coloring is an intensively studied variant of the coloring problem. The difference between regular and on-line coloring is that in the on-line setting the vertices appear one by one and the coloring algorithm must assign colors to them immediately, knowing only the edges between vertices shown thus far. Our proof exploits a correspondence between on-line coloring of overlap graphs and usual (off-line) coloring of frame intersection graphs. We obtain a structural decomposition of an arbitrary frame graph, yielding a constant number of so-called *directed frame families*. Their intersection graphs turn out to be so-called *overlap game graphs*, which may be viewed as encodings of adversary strategies in the on-line overlap graph coloring problem. We succeed in coloring overlap game graphs with $O(\log \log n)$ colors using a mixture of two strategies: heavy-light decomposition of trees (first introduced by Sleator and Tarjan [15]) and first-fit coloring. For frame intersection graphs, we use a result due to McGuinness [11] that simple triangle-free families of arc-connected compact sets in the plane pierced by a common line have bounded chromatic number, as well as coloring techniques introduced by Gyárfás in his proof that overlap graphs are χ-bounded.

2 Basic Ideas

A *frame* is the boundary of an axis-aligned rectangle. The *filling rectangle* of a frame F, denoted by $\mathrm{rect}(F)$, is the rectangle whose boundary is F. The x-coordinates of the left and right sides of F and the y-coordinates of the bottom and top sides of F are denoted by $\ell(F)$, $r(F)$, $b(F)$, $t(F)$, respectively. Thus $\mathrm{rect}(F) = [\ell(F), r(F)] \times [b(F), t(F)]$.

Throughout the paper we assume that all frames are in general position, that is, no corner of any frame lies on another frame. We can easily adjust any family of frames to satisfy this condition without introducing or losing any intersection, just by expanding each frame in every direction by a tiny amount inversely proportional to the area enclosed by the frame. We distinguish the following types of frame intersections, illustrated in Fig. 1: *crossings*, *leftward-*, *rightward-*, *downward-* and *upward-directed intersections*, and *diagonal intersections*. A family of frames $\mathcal{F}$ is *leftward-*, *rightward-*, *downward-* or *upward-directed* if the following two conditions are satisfied:

(D1) the intersection of any two frames in $\mathcal{F}$ is leftward-, rightward-, downward- or upward-directed, respectively,
(D2) no frame in $\mathcal{F}$ is enclosed by two intersecting frames in $\mathcal{F}$ (see Fig. 1).

The first condition explains the term "directed", while the second one is for technical reasons. A family of frames $\mathcal{F}$ is *directed* if it is leftward-, rightward-, downward- or upward-directed. Note that in a directed family, we still allow only one of the four types of directed intersections, we just do not specify which one.

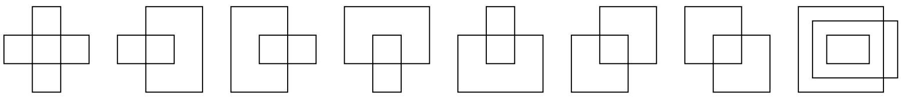


Fig. 1. From left to right: a crossing; a leftward-, rightward-, downward- and upward-directed intersection; two diagonal intersections; forbidden configuration in a rightward-directed family

The first step in our proof of Theorem 1, explained in Section 5, is to reduce to the case of directed triangle-free families of frames:

Lemma 1. *Every triangle-free family of frames $\mathcal{F}$ can be partitioned into a bounded number of directed subfamilies, where the bound is independent of $\mathcal{F}$.*

The next step is a more abstract description of the structure of intersection graphs of directed families of frames in terms of intervals on $\mathbb{R}$. We denote the family of all closed intervals on $\mathbb{R}$ by $\mathcal{I}$. The left and right endpoints of an interval I are denoted by $\ell(I)$ and $r(I)$, respectively. Again, we assume that we are dealing with intervals in general position, that is, the endpoints of all intervals are distinct. Intervals I and J *overlap* if $\ell(I) < \ell(J) < r(I) < r(J)$ or $\ell(J) < \ell(I) < r(J) < r(I)$. The *overlap graph* defined on a family of intervals has an edge for each pair of overlapping intervals.

Let G be a triangle-free graph, M be a rooted forest with $V(M) = V(G)$, and $\mu : V(G) \to \mathcal{I}$. For $u, v \in V(G)$, we write $u \prec v$ if $u \neq v$ and u is an ancestor of v in M. The graph G is an *overlap game graph* with *meta-forest* M and *representation* μ if the following conditions are satisfied:

(G1) $\ell(\mu(x)) < \ell(\mu(y))$ whenever $x \prec y$,
(G2) x and y are adjacent in G if and only if $x \prec y$ and $\mu(x)$ and $\mu(y)$ overlap,
(G3) there are no x, y, z such that $x \prec y \prec z$, $\mu(x)$ and $\mu(y)$ overlap, and $\mu(z) \subset \mu(x) \cap \mu(y)$.[1]

Lemma 2. *The intersection graph of a directed family of frames is an overlap game graph.*

It should be noted that the converse of Lemma 2 also holds, that is, every overlap game graph is the intersection graph of a directed family of frames. To see this, for each $u \in V(G)$ define $\eta(u) \in \mathcal{I}$ so that $\eta(v) \subset \eta(u)$ whenever $u \prec v$, and $\eta(u) \cap \eta(v) = \emptyset$ whenever u and v are $\prec$-incomparable. The boundaries of $\mu(u) \times \eta(u)$ for $u \in V(G)$ form the requested family.

Lemma 3. *Overlap game graphs have chromatic number $O(\log \log n)$.*

Now, Theorem 1 follows from Lemmas 1, 2, and 3. We prove Lemmas 2 and 3 in Sections 4 and 3, respectively.

[1] Lemmas 2 and 3 remain valid if we drop the conditions (D2) in the definition of a directed intersection and (G3) in the definition of an overlap game graph. The reason why we impose them is that we get (D2) for free in the proof of Lemma 1, while (G3) simplifies the proof of Lemma 3.

The construction from [10] shows that the bound in Theorem 1 is tight. By Lemmas 1 and 2, the bound in Lemma 3 is also tight.

We complete this outline by explaining the meaning of the word "game" in the notion of overlap game graphs. Let $k \in \mathbb{N}$. Consider the following *overlap coloring game* between two players: Presenter, who presents intervals one by one, and Painter, who colors them on-line, that is, each interval is colored right after it is presented and without possibility of changing the color later. Presenter's moves are restricted by the following rules:

(i) if an interval I_2 is presented after I_1, then $\ell(I_1) < \ell(I_2)$;
(ii) no three intervals I_1, I_2, I_3 such that $\ell(I_1) < \ell(I_2) < \ell(I_3) < r(I_1) < r(I_2) < r(I_3)$ are presented, that is, the overlap graph defined on the intervals presented is triangle-free;
(iii) no three intervals I_1, I_2, I_3 such that $\ell(I_1) < \ell(I_2) < \ell(I_3) < r(I_3) < r(I_1) < r(I_2)$ are presented, that is, no two overlapping intervals contain a third one.

The coloring constructed by Painter has to be proper. Presenter aims to force Painter to use more than k colors, while Painter tries to do with at most k colors.

Every finite strategy of Presenter (not necessarily winning or deterministic) gives rise to an overlap game graph G with meta-forest M and representation μ such that the root-to-leaf paths in M correspond to the intervals presented on the possible branches of the strategy. Specifically, each root r of M corresponds to an interval $\mu(r)$ that can be played in Presenter's first move, and each child of a vertex x of M corresponds to an interval that Presenter can play right after $\mu(x)$ at the position represented by x. Conversely, an overlap game graph G with meta-forest M and representation μ represents a non-deterministic strategy of Presenter, as follows. Presenter starts with an arbitrarily chosen root r of M presenting $\mu(r)$, and then, in each move from position u in M, follows to an arbitrarily chosen child v of u and presents $\mu(v)$. Now, the crucial observation is that Painter has a strategy to use at most k colors against the considered strategy of Presenter if and only if $\chi(G) \leqslant k$. The proof of Lemma 3 essentially shows that each such strategy needs to have a double exponential number of branches.

3 Coloring Overlap Game Graphs

For the purpose of this entire section, let G be an n-vertex overlap game graph with meta-forest M and representation μ. Our goal is to prove that G has chromatic number $O(\log \log n)$. Since different components of M are not connected by edges of G, they can be colored independently using the same set of colors. Thus it is enough to consider each component of M separately, and therefore we can assume without loss of generality that M is a single tree.

The relation $\prec$ defines an orientation of the edges of G: we write $x \to y$ if x is adjacent to y and $x \prec y$. We classify each vertex v of G as either *primary* or *secondary* as follows:

- if v is the root of M, then v is primary;
- if $\mu(v) \subset \mu(u)$, where u is the parent of v, then v is secondary; otherwise v is primary.

Let $P(u) \subset V(G)$ be the set of vertices v for which u is the first primary vertex on the path from v to the root of M (including u itself). Clearly, $P(u)$ is independent in G and $\mu(v) \subset \mu(u)$ for $v \in P(u)$. Let $P(u) \to P(v)$ denote that $u \prec v$ and there are $x \in P(u)$ and $y \in P(v)$ such that $x \to y$.

We are going to show that any proper k-coloring of the primary vertices of G can be transformed into a proper $2k$-coloring of the whole graph G. This can be done with the help of the following lemma.

Lemma 4. *Let S be an independent set of primary vertices in G, and let $v \in S$. There is at most one vertex $u \in S$ such that $P(u) \to P(v)$.*

Proof. Omitted.

We now show how to color the vertices of G with $2k$ colors. Let S be a color class in a proper k-coloring of the primary vertices of G. Consider all the sets $P(u)$ for $u \in S$. Each of these sets is independent, and by Lemma 4, the edges between the sets form a bipartite graph. Therefore, we need just two colors for the vertices in $\bigcup_{u \in C} P(u)$ and $2k$ colors in total for the whole G.

It suffices to show that the chromatic number of the graph induced on the primary vertices is $O(\log\log n)$. The primary vertices induce an overlap game graph witnessed by the restriction of μ and $\prec$. From now on, we simply assume that there are no secondary vertices in G.

Let P be the set of (primary) vertices lying on a path from the root to a leaf of M. We say that two edges $x \to y$ and $z \to t$ with $x, y, z, t \in P$ *overlap* if either $x \prec z \prec y \prec t$ or $z \prec x \prec t \prec y$. The following lemma essentially means that the set P induces a forest in G with no overlapping edges.

Lemma 5. *The following statements hold:*

(1) *For every $v \in P$ there is at most one $w \in P$ such that $v \to w$.*
(2) *No two edges $v_1 \to v_3$ and $v_2 \to v_4$ with $v_1, v_2, v_3, v_4 \in P$ overlap.*

Proof. Omitted.

To continue the proof we need to introduce the idea of heavy-light decomposition due to Sleator and Tarjan [15]. Let T be a rooted tree, and let T_v denote the subtree of T rooted at a vertex v. For every internal vertex u of T, let $s(u)$ be a child v of u such that the number of vertices in T_v is as large as possible. The edges of T connecting u to $s(u)$ are called *heavy*. Clearly, there is a unique root-to-leaf path in T consisting of heavy edges only—call it the *heavy path* of T. Remove the heavy path from the tree, obtaining a forest. Continue by removing heavy paths from each tree in the forest, until there is nothing left to remove. The resulting vertex cover of T by paths is called the *heavy-light decomposition* of T. We will call each path in this cover a heavy path of T. The heavy-light decomposition has the following crucial property.

Lemma 6. *If a root-to-leaf path in T intersects k heavy paths, then T has at least $2^k - 1$ vertices.*

Proof. Straightforward induction. $\square$

Fix a heavy-light decomposition of M. Form an auxiliary graph G' by removing the edges of G that connect two vertices in different heavy paths. By Lemma 5(1), the vertices on each heavy path induce a forest in G, hence G' is a forest and can be properly colored with two colors. Let C_1 and C_2 be the coloring classes in a proper two-coloring of G', and fix $i \in \{1, 2\}$. For any $x, y \in C_i$, the following holds:

$$\text{If } x \to y, \text{ then } x \text{ and } y \text{ are in different heavy paths.} \qquad (*)$$

We color the vertices from C_i with positive integers one at a time, going from the root of M towards the leaves. We choose the color of a vertex $v \in C_i$ to be the least positive integer not occurring as a color of any vertex w with $w \in C_i$ and $w \to v$ (every such vertex has been colored before v). This coloring strategy is known as *first-fit.*

Lemma 7. *If first-fit assigns a color k to some vertex $v \in C_i$, then the path P in M from the root to v intersects at least 2^{k-2} heavy paths.*

Proof. Let $f(w)$ denote the color chosen by first-fit for each vertex $w \in C_i$. The colors $1, \ldots, f(v) - 1$ have been chosen for vertices $w \in P \cap C_i$ with $w \to v$, so there are at least $f(v) - 1$ such vertices. Let F be a minimal subset of $P \cap C_i$ that satisfies the following conditions:

- F contains v,
- for any $w \in F$, there are $w_1, \ldots, w_{f(w)-1} \in F$ such that $w_j \to w$ and $f(w_j) = j$ for $1 \leqslant j \leqslant f(w) - 1$.

By Lemma 5(1), the set F induces a directed tree T in G with the root v and edges directed towards v. By the minimality of F, each vertex w of T has exactly $f(w) - 1$ children (which are closer in M to the root of M). It follows that T has exactly 2^{k-1} vertices and 2^{k-2} internal vertices.

Let w be an internal vertex of T, and let $u \in F$ be the vertex that precedes w in the order $\prec$ on F. We claim that u is a child of w in T. Suppose it is not. Since u precedes w in the order $\prec$ and therefore is not the root of T, it has a parent p in T. Since w is an internal vertex of T, it has a child c in T. We know that $w \prec p$ and $c \prec u$, as u and w are consecutive in the order $\prec$ on F. It follows that $c \prec u \prec w \prec p$ and the edges $c \to w$ and $u \to p$ overlap, which contradicts Lemma 5(2). We have shown that there is an edge between u and w in T, so they must lie on different heavy paths. Consequently, any two $\prec$-consecutive internal vertices of T lie on different heavy paths, which shows that P intersects at least 2^{k-2} heavy paths. $\square$

Proof (Lemma 3). As noted previously, we may assume that G consists of primary vertices only and is connected. Suppose that k is the maximal color used by first-fit on a vertex $v \in C_i$. By Lemma 7, the path in M from the root to v intersects at least 2^{k-2} heavy paths. This implies, by Lemma 6, that $n \geqslant 2^{2^{k-2}} - 1$. Therefore, first-fit uses at most $O(\log\log n)$ colors on C_i. We color C_1 and C_2 by first-fit using two separate sets of colors, obtaining a proper coloring of G with $O(\log\log n)$ colors. □

4 Reduction to Overlap Game Graphs

Proof (Lemma 2). Let $\mathcal{F}$ be a directed family of rectangular frames. We assume without loss of generality that $\mathcal{F}$ is rightward-directed. Define a map $\mu : \mathcal{F} \to \mathcal{I}$ so that $\mu(F)$ is the interval obtained by projecting F on the x-axis. Thus we have $\ell(\mu(F)) = \ell(F)$ and $r(\mu(F)) = r(F)$.

For $F \in \mathcal{F}$, let $\mathcal{L}(F)$ be the subfamily of $\mathcal{F}$ consisting of such F' that $\ell(F') < \ell(F) < r(F')$ and $b(F') < b(F) < t(F) < t(F')$. We define a rooted forest M on $\mathcal{F}$ as follows. If $\mathcal{L}(F)$ is empty, then F is a root of M. Otherwise, the parent of F in M is the member F' of $\mathcal{L}(F)$ with greatest $\ell(F')$. We show that the intersection graph of $\mathcal{F}$ is an overlap game graph with meta-forest M and representation μ. To this end, we argue that the conditions (G1)–(G3) from the definition of an overlap game graph are satisfied by the intersection graph of $\mathcal{F}$.

It follows directly from the definition of parent that $F_1 \prec F_2$ implies $\ell(F_1) < \ell(F_2)$ and $b(F_1) < b(F_2) < t(F_2) < t(F_1)$. This already shows (G1) and the right-to-left implication in (G2). Let $F_1, F_2, F_3 \in \mathcal{F}$ be such that $F_1 \prec F_2 \prec F_3$, $\mu(F_1)$ and $\mu(F_2)$ overlap, and $\mu(F_3)$ is contained in both $\mu(F_1)$ and $\mu(F_2)$. We have $\ell(F_1) < \ell(F_2) < \ell(F_3) < r(F_3) < r(F_1) < r(F_2)$ and $b(F_1) < b(F_2) < b(F_3) < t(F_3) < t(F_2) < t(F_1)$. However, such a configuration is forbidden in a directed family of rectangular frames. This contradiction shows (G3). Now, let F_1 and F_2 be two intersecting members of $\mathcal{F}$. By the assumption that $\mathcal{F}$ is rightward-directed, $\mu(F_1)$ and $\mu(F_2)$ overlap, and we have $F_1 \in \mathcal{L}(F_2)$ or $F_2 \in \mathcal{L}(F_1)$. Therefore, in order to prove the left-to-right implication in (G2), it remains to show that $F_1 \in \mathcal{L}(F_2)$ implies $F_1 \prec F_2$. To this end, we use induction on the increasing order of $\ell(F_2)$. There is nothing to prove when F_1 is the parent of F_2, so assume the other case. Let F_2' be the parent of F_2. We have $\ell(F_1) < \ell(F_2') < \ell(F_2) < r(F_1)$ and, since $\mathcal{F}$ is rightward-directed, $b(F_1) < b(F_2') < t(F_2') < t(F_1)$. Thus $F_1 \in \mathcal{L}(F_2')$. This and the induction hypothesis yield $F_1 \prec F_2'$ and thus $F_1 \prec F_2$. □

5 Reduction to Directed Families of Frames

The goal of this section is to prove Lemma 1. This is achieved via a combination of techniques introduced by Asplund and Grünbaum [5] and Gyárfás [6].

Lemma 8. *Every triangle-free family of frames $\mathcal{F}$ can be partitioned into two subfamilies each containing no pair of crossing frames.*

Proof. Omitted.

Two families of frames $\mathcal{F}_1$ and $\mathcal{F}_2$ are *mutually independent* if any pair of frames $F_1 \in \mathcal{F}_1$ and $F_2 \in \mathcal{F}_2$ is non-intersecting. Let $\mathcal{F}$ be a family of frames. For $\mathcal{L} \subset \mathcal{F}$, a frame $F' \in \mathcal{F} \setminus \mathcal{L}$ is *external* to $\mathcal{L}$ if $F' \not\subset \bigcup_{F \in \mathcal{L}} \operatorname{rect}(F)$. A subfamily $\mathcal{L}$ of $\mathcal{F}$ is a *layer* with respect to $\mathcal{F}$ if $|\mathcal{L}| = 1$ or every frame in $\mathcal{L}$ intersects some frame in $\mathcal{F} \setminus \mathcal{L}$ external to $\mathcal{L}$.

Lemma 9. *Every family of frames $\mathcal{F}$ has a partition $\mathcal{P}$ into layers. Moreover, there is a bipartition $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$ such that each $\mathcal{P}_i$ consists of mutually independent layers.*

Proof. Omitted.

Theorem 2 (Asplund, Grünbaum [5]). *Every triangle-free family of axis-aligned rectangles can be properly colored with* 6 *colors.*

Theorem 3 (McGuinness [11]). *Let L be a closed Jordan loop. Let $\mathcal{C}$ be a triangle-free family of curves such that $|L \cap C| = 1$ for any $C \in \mathcal{C}$ and $|C_1 \cap C_2| \leqslant 1$ for any distinct $C_1, C_2 \in \mathcal{C}$. It follows that $\chi(\mathcal{C}) \leqslant \beta$ for a constant β independent of L and $\mathcal{C}$.*

Proof (Lemma 1). By Lemma 8, it is enough to find the required partition for triangle-free families of frames containing no crossings. Thus, assume $\mathcal{F}$ is such a family of frames. Our goal is to color $\mathcal{F}$ with a bounded number of colors so that the frames of each color form a directed family. For simplicity, we construct a coloring of $\mathcal{F}$ such that every connected component of the intersection graph of frames of each color corresponds to a directed family. We call such a coloring *good*. Once we have a good coloring of $\mathcal{F}$, we can easily obtain a coloring required by the lemma using at most four times as many colors.

Let $\mathcal{P}$ be a partition of $\mathcal{F}$ into layers and $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$ be a bipartition claimed by Lemma 9. Each layer $\mathcal{L} \in \mathcal{P}$ consists of a single frame or is such that for every $F \in \mathcal{L}$ there is $F' \in \mathcal{F} \setminus \mathcal{L}$ external to $\mathcal{L}$ and intersecting F. This guarantees that each layer $\mathcal{L}$ satisfies the condition (D2) in the definition of directed family of frames: if $F, F_1, F_2 \in \mathcal{L}$ are such that F_1 and F_2 intersect and both enclose F, then F_1, F_2 and F' form a triangle. It is enough to find a good coloring of each layer $\mathcal{L} \in \mathcal{P}$ with a bounded number of colors. Then, we can color all the layers in $\mathcal{P}_1$ with one set of colors and all the layers in $\mathcal{P}_2$ with another, separate set of colors, obtaining a good coloring of the entire $\mathcal{F}$ with twice as many colors in total.

Fix a layer $\mathcal{L} \in \mathcal{P}$, and once again apply Lemma 9 to get a partition $\mathcal{Q}$ of $\mathcal{L}$ into layers. By the very same argument, it is enough to find a good coloring of each layer $\mathcal{G} \in \mathcal{Q}$ with a bounded number of colors. So fix a layer $\mathcal{G} \in \mathcal{Q}$, and assume $|\mathcal{G}| > 1$. It follows that for each $F \in \mathcal{G}$ there is a frame $F' \in \mathcal{L} \setminus \mathcal{G}$ external to $\mathcal{G}$ and intersecting F.

Let $\mathcal{E}$ be the family of those frames in $\mathcal{G}$ that are not enclosed by any other frame in $\mathcal{G}$. Two frames in $\mathcal{E}$ intersect if and only if their filling rectangles intersect. Therefore, by Theorem 2, $\mathcal{E}$ can be colored properly with 6 colors. Let $\xi : \mathcal{E} \to \{1, \ldots, 6\}$ be such a coloring. For each $F \in \mathcal{G} \setminus \mathcal{E}$, choose any frame

$e(F)$ enclosing F. For $E \in \mathcal{E}$, let $e^{-1}(E) = \{F \in \mathcal{G} : e(F) = E\}$. It is enough to obtain a good coloring of each $e^{-1}(E)$ with a bounded number of colors. Indeed, one can first partition $\mathcal{G}$ into at most 6 families according to $\xi(F)$ when $F \in \mathcal{E}$ and $\xi(e(F))$ otherwise. Each of these families has a good coloring with bounded number of colors, as when $E_1, E_2 \in \mathcal{E}$, $E_1 \neq E_2$ and $\xi(E_1) = \xi(E_2)$, the families $e^{-1}(E_1) \cup \{E_1\}$ and $e^{-1}(E_2) \cup \{E_2\}$ are mutually independent.

Fix $E \in \mathcal{E}$, and let $\mathcal{M} = e^{-1}(E)$. For each $M \in \mathcal{M}$, choose a frame $s(M) \in \mathcal{L} \setminus \mathcal{G}$ external to $\mathcal{G}$ and intersecting M. If there are more than one candidates for $s(M)$, choose one that is not enclosed by any other candidate. The frame $s(M)$ is the *support* of M. Each $s(M)$ is external to $\mathcal{M} \cup \{E\}$ and thus intersects E. It follows that the supports of frames in $\mathcal{M}$ have pairwise disjoint filling rectangles. Indeed, no two supports of frames in $\mathcal{M}$ intersect, as together with E they would form a triangle. Moreover, since each frame in $\mathcal{M}$ is enclosed by E and no frame in $\mathcal{L}$ can be enclosed by two intersecting frames in $\mathcal{L}$, no frame in $\mathcal{M}$ is enclosed by any support. Therefore, no $s(M_1)$ encloses any $s(M_2)$ with $M_1, M_2 \in \mathcal{M}$, as then $s(M_1)$ would either enclose or intersect M_2, the latter being excluded by the choice of $s(M_2)$.

When the intersection of two frames is leftward-, rightward-, downward- or upward-directed, we say that the frame whose two opposite sides intersect one side of the other frame *enters* that other frame *from the left*, *the right*, *below* or *above*, respectively.

At most four supports enclose corners of E. Since frames with a common support must be pairwise disjoint, the members of $\mathcal{M}$ with supports enclosing a corner of E can be colored properly with four colors. The remaining frames in $\mathcal{M}$ are partitioned into four classes according to whether their supports enter E from below, above, the left, or the right, and each of these classes is colored independently with a separate set of colors. We restrict our attention to the class $\mathcal{M}'$ of frames with supports entering E from below—the others are handled analogously.

Let $I = \operatorname{int}(\operatorname{rect}(E)) \setminus \bigcup_{M \in \mathcal{M}'} \operatorname{rect}(s(M))$. Each frame $M \in \mathcal{M}'$ intersects I, as it cannot be enclosed by any support. The frames in $\mathcal{M}'$ entering some support from the left are pairwise disjoint (see Fig. 2)—we use a separate color on them. Similarly for the frames in $\mathcal{M}'$ entering some support from the right. Let $\mathcal{M}''$ be the family of the remaining frames, that is, those frames in $\mathcal{M}'$ that enter their supports from above, are entered by their supports from below, or intersect their supports diagonally.

For each frame $M \in \mathcal{M}''$, define two curves, the *left* and *right trace* of M, as follows. The left (right) trace starts at the top right (left) vertex of M, follows to the top left (right) vertex of M along the top side of M, and then continues along M until it reaches the boundary of I on either the left (right) or the bottom side of M.

No frame $M_1 \in \mathcal{M}''$ can enter any $M_2 \in \mathcal{M}''$ from above, as M_1, M_2 and $s(M_1)$ would form a triangle or M_1 would be enclosed by $s(M_2)$ (see Fig. 3). Therefore, only the following types of intersections can occur in $\mathcal{M}''$:

(i) one frame enters another from below,
(ii) one frame enters another from the left,

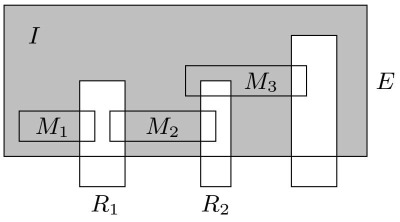


Fig. 2. Frames M_1 and M_2 (M_2 and M_3) entering some support from the left are disjoint, as otherwise they would form a triangle with R_1 (R_2, respectively)

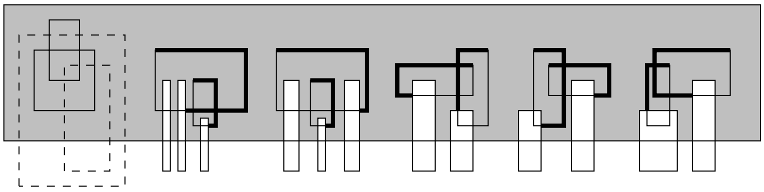


Fig. 3. From left to right: no frame in $\mathcal{M}''$ can enter another from above; case (i), right traces intersect; case (i), traces do not intersect; case (ii), left traces intersect; case (iii), right traces intersect; case (iv), left traces intersect

(iii) one frame enters another from the right,
(iv) two frames intersect diagonally.

It is easy to see that in cases (ii)–(iv) either the left or the right traces of the two frames intersect, and in every case the left or right traces intersect at at most one point (see Fig. 3).

The intersection graph of the left (right) traces is triangle-free, because it is a subgraph of the intersection graph of $\mathcal{M}''$. Since every trace meets the boundary of I at exactly one point, we can apply Theorem 3 twice to obtain two colorings $\xi_L, \xi_R : \mathcal{M}'' \to \{1, \ldots, \beta\}$, one proper on the left traces and the other proper on the right traces. The coloring by pairs (ξ_L, ξ_R) distinguishes all pairs of frames in $\mathcal{M}''$ with intersections of types (ii)–(iv), and thus only intersections of type (i) remain in each color class. Additionally, no frame is enclosed by two intersecting frames within one color class, as this has been excluded earlier in the argument. Therefore, each color class is a directed family of frames and the coloring that we obtained is good as required. □

6 Open Problems

The authors of [9] asked for the maximum possible chromatic number of a triangle-free intersection graph of n segments. In this paper, we resolved a similar question for frames. The following problems ask whether segment graphs behave similarly to frame graphs with respect to proper coloring.

Problem 1. Can every triangle-free segment intersection graph be decomposed into a bounded number of overlap game graphs?

Problem 2. Does every triangle-free segment intersection graph with chromatic number k contain an overlap game graph with chromatic number at least ck as an induced subgraph, for some absolute constant $c > 0$?

The positive answer to the question in Problem 1 would yield the answer $\Theta(\log\log n)$ bound for triangle-free segment intersection graphs, while the negative answer to the question in Problem 1 or 2 would help us understand the limitations of our methods. The questions can be generalized to K_k-free graphs.

Problem 3. What is the maximum possible chromatic number of a K_k-free intersection graph of n frames?

References

1. Zykov, A.A.: On some properties of linear complexes. Mat. Sb. (N.S.) 24(66)(2), 163–188 (1949) (in Russian)
2. Mycielski, J.: Sur le coloriage des graphes. Colloq. Math. 3, 161–162 (1955)
3. Kim, J.H.: The Ramsey number $R(3,t)$ has order of magnitude $t^2/\log t$. Random Struct. Algor. 7(3), 173–208 (1995)
4. Ajtai, M., Komlós, J., Szemerédi, E.: A note on Ramsey numbers. J. Combin. Theory Ser. A 29(3), 354–360 (1980)
5. Asplund, E., Grünbaum, B.: On a colouring problem. Math. Scand. 8, 181–188 (1960)
6. Gyárfás, A.: On the chromatic number of multiple interval graphs and overlap graphs. Discrete Math. 55(2), 161–166 (1985)
7. Gyárfás, A.: Corrigendum: On the chromatic number of multiple interval graphs and overlap graphs. Discrete Math. 62(3), 333 (1986)
8. Burling, J.P.: On coloring problems of families of prototypes. PhD thesis, University of Colorado, Boulder (1965)
9. Pawlik, A., Kozik, J., Krawczyk, T., Lasoń, M., Micek, P., Trotter, W.T., Walczak, B.: Triangle-free intersection graphs of line segments with large chromatic number. arXiv:1209.1595 (submitted)
10. Pawlik, A., Kozik, J., Krawczyk, T., Lasoń, M., Micek, P., Trotter, W.T., Walczak, B.: Triangle-free geometric intersection graphs with large chromatic number. Discrete Comput. Geom. 50(3), 714–726 (2013)
11. McGuinness, S.: Colouring arcwise connected sets in the plane I. Graph. Combin. 16(4), 429–439 (2000)
12. Suk, A.: Coloring intersection graphs of x-monotone curves in the plane. To appear in Combinatorica, arXiv:1201.0887
13. Fox, J., Pach, J.: Coloring K_k-free intersection graphs of geometric objects in the plane. European J. Combin. 33(5), 853–866 (2012)
14. Fox, J., Pach, J.: Applications of a new separator theorem for string graphs. To appear in Combin. Prob. Comput., arXiv:1302.7228
15. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. J. Comput. System Sci. 26(3), 362–391 (1983)

On Finding Tucker Submatrices and Lekkerkerker-Boland Subgraphs

Nathan Lindzey and Ross M. McConnell

Computer Science Department, Colorado State University, Fort Collins, CO, 80523-1873 U.S.A.
{lindzey,rmm}@cs.colostate.edu

Abstract. Lekkerkerker and Boland characterized the minimal forbidden induced subgraphs for the class of interval graphs. We give a linear-time algorithm to find one in any graph that is not an interval graph. Tucker characterized the minimal forbidden submatrices of matrices that do not have the consecutive-ones property. We give a linear-time algorithm to find one in any matrix that does not have the consecutive-ones property.

1 Introduction

A graph is an *interval graph* if it is the intersection graph of a set of intervals on a line. Such a set of intervals is known as an *interval model* of the graph. They are an important subclass of perfect graphs [4], they have been written extensively about and they model constraints in various combinatorial optimization and decision problems [13, 15]. They have a rich structure and history, and interesting relationships to other graph classes. For a survey, see [2].

If M is a 0-1 (binary) matrix, we let $size(M)$ denote the number of rows, columns and 1's. Such a matrix has the *consecutive-ones property* if there exists a reordering of its columns such that, in every row, the 1's are consecutive. A *consecutive-ones matrix* is a matrix that has the consecutive-ones property, and a *consecutive-ones-ordered matrix* is a matrix where the 1's are consecutive in every row. A *clique matrix* of a graph is a matrix that has a row for each vertex, a column for each clique, and a 1 in row i, column j if vertex i is contained in clique j. A graph is an interval graph if and only if its clique matrices have the consecutive-ones property, see, for example [4].

In 1962, Lekkerkerker and Boland described the minimal induced forbidden subgraphs for the class of interval graphs [7], known as the *LB graphs* (Figure 2). Ten years later, Tucker described the minimal forbidden submatrices for consecutive-ones matrices [19]. These are depicted in Figure 1. Not surprisingly, there is a relationship between the intersection graphs of rows of Tucker matrices and the LB graphs, depicted in Figure 2.

In this paper, we give a linear time bound for finding one of the LB subgraphs when a graph is not an interval graph. As part of our algorithm, we also give a linear-time ($O(size(M))$) bound for finding one of Tucker's submatrices in a

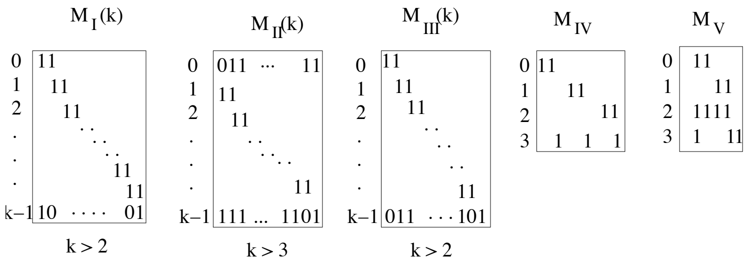


Fig. 1. The minimal forbidden submatrices for consecutive-ones matrices. For M_I, $k \geq 3$, and for M_{II} and M_{II}, $k \geq 4$. M_{IV} and M_I have fixed size.

matrix M that does not have the consecutive-ones property. This latter problem was solved previously in $O(n * size(M))$ time in [17], where n is the number of rows of the matrix. An $O(\Delta^3 m^2 n(m+n^2))$ bound for finding a Tucker submatrix of minimum size is given in [9], where Δ is the maximum number of 1's in any row.

A graph is *chordal* if it has no *chordless cycle* (an induced cycle on four or more vertices). A vertex is *simplicial* if it and its neighbors induce a complete subgraph. Every chordal graph has a simplicial vertex, and every interval graph is chordal [4].

An interval graph is *proper* if there exists an interval model where no interval is a subset of another. It is a *unit* interval graph if there exists an interval model where all intervals have the same length. These graph classes are the same, and Wegner showed that a graph is a proper interval graph if and only if it does not have a chordless cycle, the special case of G_{IV} or G_V for $n = 6$ or the *claw* ($K_{1,3}$) as an induced subgraph [20]. Hell and Huang give an algorithm that produces one of them in linear time [5] . The problem of finding a forbidden subgraph reduces easily to finding an LB subgraph. Each of the LB graphs is either one of Wegner's forbidden subgraphs or contains an obvious claw, and finding a claw in linear time, given an interval model, is elementary. By itself, this approach has no obvious advantages over Hell and Huang's elegant algorithm, but such reductions are useful when studying or programming a collection of related algorithms.

A *certifying algorithm* is an algorithm that provides, with each output, a simple-to-check proof that it has answered correctly [6, 11]. An interval model gives a certificate that a graph is an interval graph, and an LB subgraph gives one if the graph is not an interval graph. However, a certifying algorithm was given previously in [6]. The ability to give a consecutive-ones ordering or a Tucker submatrix in linear time gives a linear-time certifying algorithm for consecutive-ones matrices, but one was given previously in [10]. The previous certificates are easier to check, which is a desirable property for certifying algorithms. However, they are neither minimal nor uniquely characterized. Aside from the theoretical interest in LB subgraphs, it is easy to obtain a minimal certificate of the form given in [6] from an LB subgraph found by the algorithm we describe below. Tucker submatrices may be useful in heuristics for finding large submatrices that have the consecutive-ones property, small Tucker matrices, or identifying errors in biological data [16, 3]. Our techniques provide new tools for such heuristics.

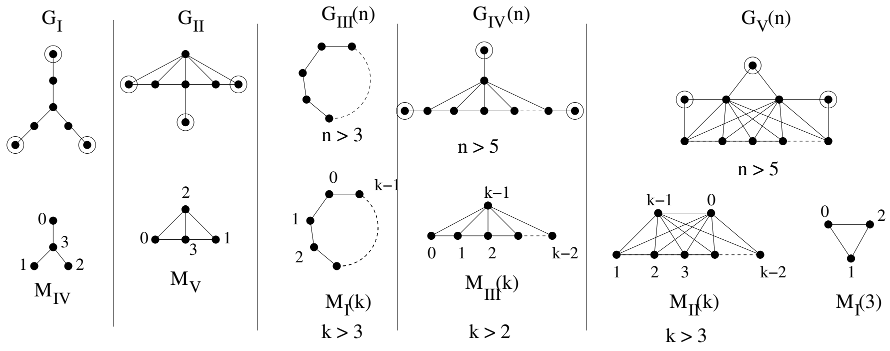


Fig. 2. G_I through G_V are the minimal non-interval graphs discovered by Lekkerkerker and Boland. The circled vertices are the simplicial vertices in the graphs. Below them are the intersection graphs of the rows of the corresponding Tucker matrices, numbered as they are in Figure 1. Removing the rows belonging to the simplicial vertices in the clique matrix of each Lekkerkerker-Boland graph gives the Tucker matrix below it in the figure.

2 Preliminaries

Given a graph G, let V denote the number of vertices and E denote the number of edges. If $\emptyset \subset X \subseteq V$, let $G[X]$ denote the subgraph induced by X. Standard sparse representations of 0-1 matrices take $O(size(M))$ space to represent M. We treat the rows and columns as *sets*, where each row R is the set of columns in which the row has a 1 and each column C is the set of rows in which the column has a 1. Suppose $\mathcal{R}$ is the set of rows of a consecutive-ones ordered matrix and $(C_1, C_2, \ldots, C_m)$ is the ordering of the columns. In linear time, we can find, for each row, the leftmost and rightmost column in the row. Let us call these the *left endpoint* and *right endpoint* of the row.

That interval graphs are a subclass of the class of chordal graphs follows from inclusion of the G_{III}'s among the LB subgraphs. Rose, Tarjan and Lueker give an $O(V + E)$ algorithm that recognizes whether a graph is a chordal graph, and, if so, produces its maximal cliques [14]. Otherwise, the algorithm of [18] produces a chordless cycle (G_{III}) in linear time.

When a graph is chordal, the problem of deciding whether it is an interval graph reduces to the problem of deciding whether its clique matrix has the consecutive-ones property. Booth and Lueker further reduced this problem to that of finding a maximal prefix $\mathcal{R}' = \{R_1, R_2, \ldots, R_r\}$ of the rows of a binary matrix M that has the consecutive-ones property, and give an algorithm that solves this in $O(size(M))$ time [1].

Assigning a left-to-right order to children of each internal node of a rooted tree results in a unique left-to-right order of the leaves. Booth and Lueker's algorithm produces a *PQ tree*, for $\mathcal{R}'$. The PQ tree represents all possible consecutive-ones orderings of $\mathcal{R}'$. There is one leaf $\{c\}$ for each column c. The internal nodes of the PQ tree consist of *P nodes* and *Q nodes*. The consecutive-ones ordering of columns are given by the leaf orders obtainable by assigning an arbitrary left-to-right order to children of each P node, and for each Q node, assigning the given left-to-right order or its reverse.

Though the PQ tree can be represented using $O(1)$ space per node, conceptually, we will consider each node of the PQ tree to be a set given by the disjoint union of its children; equivalently, it is the union of its leaf descendants.

Definition 1. *Let $\mathcal{S}$ be a collection of subsets of a set U. Two elements of U are in the same* Venn class *if they are elements of the same set of members of $\mathcal{S}$. The* unconstrained Venn class *consists of those elements of U that are not in any member of $\mathcal{S}$; all others are* constrained. *Two sets R_1, R_2* overlap *if their intersection is nonempty, but neither is a subset of the other. The* overlap graph *of $\mathcal{S}$ is the undirected graph whose vertices are the members of $\mathcal{S}$, and $R_1, R_2 \in \mathcal{S}$ are adjacent if and only if R_1 and R_2 overlap.*

Lemma 1. *[12] A set of columns is a Q node of a consecutive-ones matrix M if and only if it is the union of rows of a connected component of the overlap graph of rows of M. The Venn classes of rows in this component are its children.*

3 Breadth-First Search on the Overlap Graph of a Collection of Sets, Given a Consecutive-Ones Ordering

In linear time, we may label each row of a consecutive-ones ordered matrix with its left and right endpoints. We may then label each column c_i of a consecutive-ones ordered matrix with the set of rows that have their left endpoints in c_i. In linear time, we can then radix sort the list of sets that have their left endpoint at c_i in descending order of index of right endpoint, yielding a list $\mathcal{R}_i$. This is accomplished with a single radix sort that has the index of the left endpoint as its primary sort key and index of the right endpoint as the secondary sort key. By symmetry, we can construct a list $\mathcal{L}_i$ of rows that have their right endpoint in each column c_i, sorted in ascending order of index of left endpoint.

This allows us to perform a breadth-first search on the overlap graph of the rows in time linear in the size of the matrix, as follows. The lists $\mathcal{L}_i$ and $\mathcal{R}_i$ are represented with doubly-linked lists. We maintain the invariant that elements that have been placed in the BFS queue have been removed from these lists. When a consecutive-ones ordered set R comes to the front of the queue, we traverse its list $(c_j, c_{j+1}, \ldots, c_k)$ of columns. For each c_h in the list (excluding the first column), we remove elements from $\mathcal{R}_h$ and place them in the queue, until we reach an element in $\mathcal{R}_h$ whose right endpoint is no farther to the right than c_k. All of the removed elements overlap R. Since $\mathcal{R}_h$ is sorted in descending order of right endpoint, all these elements are a prefix of $\mathcal{R}_h$, and any remaining elements in the list do not overlap R. When we remove an element from $\mathcal{R}_h$, we remove it from any list $\mathcal{L}_{h'}$ that it is a member of, to maintain the invariant. This takes $O(1)$ time for each element moved to the BFS queue, plus $O(1)$ time for each column of R. The lists L_h are handled symmetrically. Summing over all rows R, the time is $O(size(M))$.

4 Finding Tucker Submatrices

4.1 Tucker Matrices with at Most Four Rows

Lemma 2. *If a set $\mathcal{R}'$ of rows has the consecutive-ones property and Z is a row such that $\mathcal{R} = \mathcal{R}' \cup \{Z\}$ does not, then Z is one of the rows of every Tucker submatrix in $\mathcal{R}$.*

Algorithm 1. initialRows(M',k)

```
**Precondition:** $M'$ does not have the consecutive-ones property
**Postcondition:** Given by Lemma 3
$M \longleftarrow M'$;
$i \longleftarrow 1$;
**while** $i \leq k+1$ *and* $M$ *has at least* $i-1$ *rows* **do**
    Using Booth and Lueker's algorithm [1], find the minimal prefix
    $(R_1, R_2, \ldots, R_r, Z)$ of rows of $M$ that does not have the consecutive-ones
    property;
    $M \longleftarrow (Z, R_1, R_2, \ldots, R_r)$; //set $Z$ as first row of $M$
    $i \longleftarrow i+1$;
```

Lemma 3. *Suppose Algorithm 1 is run with parameter k and a matrix M' that does not have the consecutive-ones property. If the returned matrix M has at most k rows, then these are the rows of every Tucker matrix of M. Otherwise, M fails to have the consecutive-ones property and every Tucker submatrix in M has at least $k+1$ rows.*

Proof. By induction on i, M does not have the consecutive-ones property at the end of iteration i. Also, by induction on i, using Lemma 2, at the end of iteration i, for every Tucker submatrix M_T of M, the rows of M_T include the first i rows of M. If M_T has only i rows, then the first i rows of M do not have the consecutive-ones property, so at the end of iteration $i+1$, M will have i rows.

We run Algorithm 1 for $k = 4$. If it returns a matrix with j rows, where $j \leq 4$, it is easy to get a linear time bound to get the columns. (One way is to generate all $j! \leq 24$ orderings of rows and for each, to check for the columns of each Tucker matrix of size j.) Otherwise, Algorithm 1 returns a matrix M of more than 4 rows. By Lemma 3, M fails to have the consecutive-ones property and every Tucker submatrix of M has at least five rows. This excludes any instances of M_{IV}, M_V or the anomalous case of M_I on three rows that does not correspond to a chordless cycle.

4.2 Matrices in Which All Tucker Submatrices Have More Than Four Rows

Lemma 4. *The overlap graphs of M_I, M_{II}, and M_{III} are simple cycles.*

Definition 2. *Suppose $\mathcal{R}'$ is a set of rows with the consecutive-ones property, Q is a Q node of its PQ tree, $(X_1, X_2, \ldots, X_k)$ is the ordering of Q's children and Z is a row not in $\mathcal{R}'$. Let X_h, X_i, X_j be three children of Q such that $h < i < j$. They are a* 1-0-1 configuration *for Z if X_h and X_j each contain a 1 of row Z and X_i contains a 0 of row Z. They are a* 0-1-0 configuration *for Z if X_h and X_j each contain a 0 of row Z and X_i contains a 1 of row Z.*

Lemma 5. *If $\mathcal{R}'$ is a set of rows that has the consecutive-ones property and Z is a row not in $\mathcal{R}'$, then $\mathcal{R}' \cup \{Z\}$ does not have the consecutive-ones property if the PQ tree of $\mathcal{R}'$ has a Q node Q such that either:*

1. *Q has a 1-0-1 configuration for Z;*
2. *Q has a 0-1-0 configuration for Z and Z is not a subset of Q.*

This test is implicit in Booth and Lueker's algorithm, where it is a sufficient condition, but not a necessary one. The following is a consequence of Lemma 1.

Lemma 6. *[10] The conditions of Lemma 5 are necessary and sufficient if the overlap graph of $\mathcal{R}'$ is connected.*

Lemma 7. *If a matrix fails to have the consecutive-ones property and has no Tucker submatrix with fewer than five rows, then when Algorithm 1 is run on it with $k = 4$, at the end of one of the five iterations of its loop, the PQ tree of $\mathcal{R}' = \{R_1, R_2, \ldots, R_r\}$ will have a Q node Q with the following properties for the row Z of the iteration:*

- *Q has a 1-0-1 configuration (X_h, X_i, X_j) for Z;*
- *There exist $A, B \in \mathcal{R}'$ that are members of the component of the overlap graph on $\mathcal{R}'$ whose union is Q, and such that A contains X_h and is disjoint from X_i and X_j, and B contains X_j and is disjoint from X_h and X_i.*

Proof. If $\mathcal{T}$ is the rows of M that contain a Tucker submatrix M_T, at the end of an iteration of the loop of Algorithm 1, $Z \in \mathcal{T}$ by Lemma 2. By Lemma 4, the overlap graph of $\mathcal{T}' = \mathcal{T} \setminus \{Z\}$ is connected, so $\mathcal{T}'$ is a subset of a component of the overlap graph of $\mathcal{R}'$, which gives rise to a Q node Q of the PQ tree of $\mathcal{R}'$, by Lemma 1. Since the children of Q are the Venn classes of the component, no two Venn classes of $\mathcal{T}'$, hence no two columns of M_T, can lie in the same child of the Q node.

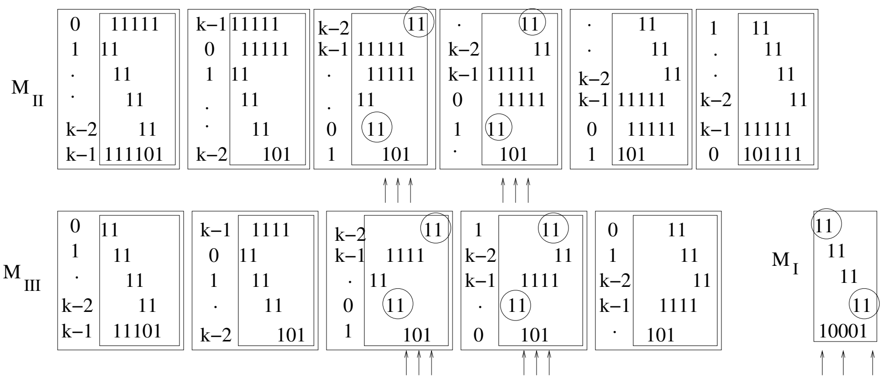


Fig. 3. Consecutive-ones orderings of all but the last row of M_I, M_{II} and M_{III} for different choices of the last row. For all but at most four choices of the last row, there exists a 1-0-1 configuration and rows A and B satisfying Lemma 7.

For each choice of a last row of a Tucker matrix on at least five rows, Figure 7 gives the possible orderings imposed on the last row by a consecutive-ones ordering of $\mathcal{T}'$, which is unique up to reversal, by Lemmas 4 and 1. In each case, if row $i \notin \{0, 1, k-2, k-1\}$ is chosen to go last, rows $i-1$ and $i+1$ satisfy the requirements of A and B. M_T has at least five rows and no row of M_T is contained in Z more than once in the five iterations, so in at least one of the iterations a row $i \notin \{0, 1, k-2, k-1\}$ will go last.

Algorithm 2. findRows(M)

```
**Preconditions:** $M$ does not have the consecutive-ones property or any Tucker
matrix with fewer than five rows.
**Postconditions:** The rows of a Tucker matrix of $M$ have been returned.
Run initialRows($M$, 4) (Algorithm 1) to find $\mathcal{R}', Z, A, B$ that satisfy Lemma 7;
$P \longleftarrow$ shortest path in overlap graph of $\mathcal{R}'$ from $A$ to $B$;
$P_1 \longleftarrow$ minimal prefix of $P$ such that the union of $\{Z\}$ and the set $\mathcal{P}_1$ of rows
of $P_1$ does not have the consecutive-ones property;
$P_2 \longleftarrow$ minimal suffix of $P_1$ such that the union of $\{Z\}$ and the set $\mathcal{P}_2$ of rows
of $P_2$ does not have the consecutive-ones property;
**return** $\mathcal{P}_2 \cup \{Z\}$;
```

In $O(size(M))$ time, all nodes can be labeled as having no descendants in Z (*empty*), all descendants in Z (*full*), or some descendants in Z and some not (*partial*), working from leaves toward the root. A procedure is given in Booth and Lueker's paper. Given this labeling, checking for the conditions of Lemma 7 in $O(size(M))$ time is trivial; details are omitted.

The correctness and linear time bound for the following are the key results of this section:

Lemma 8. *If M does not have the consecutive-ones property and every Tucker matrix of M has at least five rows, then Algorithm 2 returns the set of rows of a Tucker matrix of M.*

Proof. Since A and B lie in the same component of the overlap graph of $\mathcal{R}'$, P exists. Let $\mathcal{P}$ be the set of rows of P. Since $\mathcal{R}'$ has the consecutive-ones property, so does $\mathcal{P}$. Because $\mathcal{P}$ has a connected overlap graph, $\bigcup \mathcal{P}$ is a single Q node of the PQ tree of $\mathcal{P}$, by Lemma 1. Because of A and B, X_h, X_i, and X_j are contained in distinct Venn classes of $\mathcal{P}$, and the ones containing X_h and X_j are constrained. Since $\bigcup \mathcal{P}$ is consecutive, it must have a row that contains X_i, hence the Venn class of $\mathcal{P}$ containing X_i is also constrained. Therefore, $\mathcal{P} \cup \{Z\}$ does not have the consecutive-ones property by Lemma 5, and P_1 and P_2 exist. By Lemma 2, all Tucker matrices in $\mathcal{R}' \cup \{Z\}$ contain Z, so this applies also to $\mathcal{P}_2 \cup \{Z\}$.

Suppose there is a proper subset $\mathcal{R}''$ of the rows on P_2 such that $\mathcal{R}'' \cup \{Z\}$ contains a Tucker matrix. The overlap graph of $\mathcal{R}''$ is connected, by Lemma 4. Since P_2 is a shortest path, it is a chordless path, so $\mathcal{R}''$ is a subpath of P_2 by Lemma 4. Let $\mathcal{R}'_1$ be the rows on P_1, excluding the last row on P_1. Let $\mathcal{R}'_2$ be the rows of P_2, excluding the first row on P_2. By the minimality of P_1 and P_2, $\mathcal{R}'_1 \cup \{Z\}$ and $\mathcal{R}'_2 \cup \{Z\}$ have the consecutive-ones property. Since $\mathcal{R}''$ is a subpath of P_2, $\mathcal{R}'' \subseteq \mathcal{R}'_1$ or $\mathcal{R}'' \subseteq \mathcal{R}'_2$, so $\mathcal{R}'' \cup \{Z\}$ has the consecutive-ones property, contradicting our assumption that it does not. Therefore, $\mathcal{P}_2 \cup \{Z\}$ is the set of rows of a Tucker matrix.

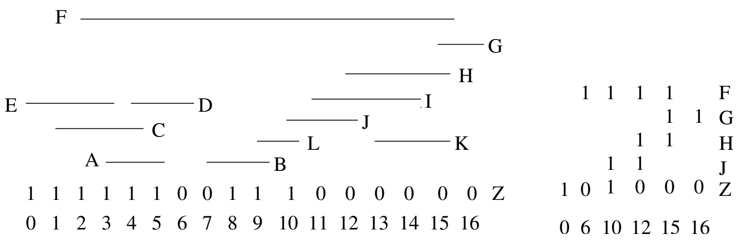


Fig. 4. Example of finding a minimal set of rows that does not have the consecutive-ones property

Figure 4 gives an example on which we illustrate some implementation details. Z is given by the 1's and 0's above the column numbers in the figure on the left, and $\mathcal{R}'$ is depicted by the intervals. The rows labeled A and B satisfy the requirements of A and B for Lemma 7, and $P = (A, E, F, G, H, J, L, B)$ is a shortest path from A to B in the overlap graph of $\mathcal{R}'$, found using the BFS algorithm of Section 3.

Using Booth and Lueker's terminology, we maintain labels on each class indicating whether it is *full* (contains only 1's of Z), *empty* (contains only 0's of Z), or *partial* (contains both 1's and 0's of Z). The minimal prefix P_1 of P whose rows, together with Z, do not have the consecutive-ones property, is (A, E, F, G, H, J). This is detected as follows. It is easy to verify that its sequence of constrained Venn classes is $(\{0,1\}$,$\{2\}$,$\{3\}$, $\{4,5\}$, $\{6,7,\ldots,9\}$, $\{10,11\}$, $\{12\}$, $\{13,14\}$, $\{15\}$, $\{16\})$, and their full/partial/empty labels $(F, F, F, F, P, P, E, E, E, E)$, respectively. Selecting a 1 from a full class, a 0 from the first partial class and a 1 from the second partial class gives a 1-0-1 configuration satisfying Lemma 5. It is the minimal such prefix. A smaller prefix, (A, E, F) has a 0-1-0 configuration, but it does not satisfy Lemma 5 because $Z \subset A \cup E \cup F$.

The minimal suffix P_2 of P_1 that satisfies Lemma 5 is (F, G, H, J), which is found in the same way by working on the reverse of P_1. Its sequence of constrained Venn classes are $(\{2,3,\ldots,9\}$, $\{10,11\}$, $\{12-14\}$, $\{15\}$, $\{16\})$, labeled (P, P, E, E, E), respectively. Selecting a 0 from the first partial class, a 1 from the next, and a 0 from an empty class gives a 0-1-0 configuration. It satisfies condition 2 of the lemma, because the unconstrained class, $\{0,1\}$ is partial, hence $Z \not\subseteq F \cup G \cup H \cup J$.

Therefore, $\{F, G, H, J, Z\}$ is the set of rows of a Tucker submatrix. A minimal set of columns that illustrates that it satisfies the lemma is $\{0, 6, 10, 12, 15, 16\}$. On the righthand side of Figure 4 is the resulting Tucker matrix, which matches the final configuration in the sequence for M_{III} in Figure 7.

This example shows that the key to finding P_1 and P_2 is maintaining the sequence of constrained Venn classes and their full/partial/empty labels as rows are added in the order in which they occur on P or on the reverse of P_1. Since they are added in an order such that every prefix of the order has a connected overlap graph, the sequence is uniquely constrained after each row is added, by Lemma 1. When a row R_i is added, if it overlaps a constrained Venn class X, X must be replaced in the sequence with two Venn classes, $(X \setminus R_i, X \cap R_i)$ or with $(X \cap R_i, X \setminus R_i)$, whichever is required to maintain consecutiveness of R_i. If R_i intersects the unconstrained class, S, then $R_i \cap S$ must be added at one extreme end of the sequence, whichever maintains consecutiveness of R_i. Details are given in [10].

The difference between this algorithm and that of [10] (and Booth and Lueker) is that, instead of testing at each iteration whether the next row R_i can be added to those considered so far without undermining the consecutive-ones property, it must repeatedly perform this test on the fixed row Z after each row R_i is added. We already know that R_i can be added, since $\mathcal{R}'$ has the consecutive-ones property. Like Booth and Lueker, the previous algorithm of [10] applies the full/partial/empty labels for R_i to facilitate the test, in $O(|R_i|)$ time, and then removes them before considering the next row R_{i+1}. Though we must perform the test on the fixed row Z at each iteration, instead of on R_i, we must do it $O(|R_i|)$ time, not $O(|Z|)$ time, in order to retain the linear time bound. To do this, we leave the full/partial/empty labelings for Z from one iteration to the next, so that we only have to update them, using R_i, rather than re-creating them each time a new row is considered.

To facilitate this, we keep updated labels $c(X)$ and $n(X)$ on each Venn class X, where $c(X)$ denotes the cardinality of X and $n(X)$ is the number of elements of Z in X. Labels only need to be updated when a Venn class is split. It is split into $X \cap R_i$ and $X \setminus R_i$. We may find $c(X \cap R_i)$ and $n(X \cap R_i)$ by counting them directly, since there are $O(|R_i|)$ of these elements. The classes are implemented with doubly-linked lists, and these sets are removed from the list for X, leaving it to represent $X \setminus R_i$. Subtracting $c(X \cap R_i)$ and $n(X \cap R_i)$ from the old labels $c(X)$ and $n(X)$ gives the updated labels for $c(X \setminus R_i)$ and $n(X \setminus R_i)$ in $O(1)$ time. Each of the new classes is full if its $c()$ and $n()$ labels are equal, empty if its $n()$ label is 0, and partial otherwise.

To evaluate whether one of the conditions of Lemma 5 holds, it is easy to see that it suffices to keep track of *transition pairs*, which are consecutive pairs such that one contains a 0 and one contains a 1. This happens when their full/partial/empty labels are unequal, or else both partial. When a new transition pair forms, we have touched at least one member of the pair within our $O(|R_i|)$ operations, so keeping track of these does not affect this time bound.

Since finding P takes linear time by the BFS of Section 3, it remains only to bound the time required to find the first step, finding the elements A and B of Lemma 7. This is much more straightforward, since we apply it once for each iteration of the loop of Algorithm 1 hence we can afford to take $\Theta(size(M))$ time for the test. We can apply the entire set of full/partial/empty labels for Z to the PQ tree within this bound, by working from the leaves to the root.

For each Q node Q, the members of the overlap component whose union is Q are unions of more than one and fewer than all of its children. How to find them in linear time for all Q nodes has been described previously, for example in [8]. We find the rows of the overlap component that contain a Venn child of Q that is labeled full or partial (a "1"). Out of all such rows, let A' be the one with a leftmost right endpoint, and let B' be the one with the rightmost left endpoint. By a simple greedy swapping argument, the overlap component giving rise to Q contains an A and a B satisfying Lemma 7 if and only if A' and B' satisfy it, which happens if and only if there is a child between the right endpoint of A' and the left endpoint of B' that is labeled partial or empty (a "0"). This can also clearly be implemented so that the time bound over all Q nodes takes $O(size(M))$ time.

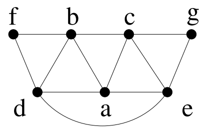
Once the set $\mathcal{P}_2 \cup \{Z\}$ of rows of a Tucker matrix have been found, it remains to find the columns. This is a set of columns, the removal of any one of which would undermine the conditions of Lemma 5, which are satisfied initially by $\mathcal{P}_2 \cup \{Z\}$. Deletion of a column undermines the lemma if and only if:

1. It disconnects the path in overlap graph on $\mathcal{R}$; or
2. It undermines the only remaining 1-0-1 configuration, or 0-1-0 configuration with a 1 in the unconstrained class.

The second test is elementary and omitted because of space constraints. For the first test, recall that $P_2 = (R_1, R_2, \ldots, R_k)$ is a chordless path in the overlap graph. Let $\mathcal{A} = \{R_1 \setminus R_2\} \cup \{X | X = R_i \cap R_{i+1} \text{ for } i \in \{1, 2, \ldots, k-1\}\}$ $\cup \{Y | Y = R_{i+1} \setminus R_i \text{ for } i \in \{1, 2, \ldots, k-1\}\}$. Each element of $\mathcal{A}$ is consecutive-ones ordered, the sum of cardinalities of sets in $\mathcal{A}$ is $O(size(M))$. The overlap graph remains connected if and only if every member of $\mathcal{A}$ contains at least one retained column. We give each column a list of members of $\mathcal{A}$ it is contained in and keep a counter on each element of $\mathcal{A}$ indicating the number of remaining columns it contains. When removing a column C, the counters can be updated by decrementing the counters of members of $\mathcal{A}$ in its list. A column cannot be removed if removing it would decrement a counter to 0.

5 Finding a Lekkerkerker-Boland Subgraph

Tucker observed that the smallest graphs whose clique matrices contain a Tucker matrix must be exactly the LB graphs [19]. Deletion of rows for the three simplicial vertices in the clique matrix of G_I, G_{II}, G_{IV} of G_V gives a Tucker matrix. However, it does not follow that every such Tucker submatrix in a clique matrix of G can be extended to a submatrix giving the clique matrix of one of these LB graphs. This is illustrated by Figure 5. Fortunately, it is true for clique matrices of chordal graphs, which we show next.



$$\begin{array}{c|cccccc|}
\hline
a & 1 & 1 & & & 1 & 1 \\
b & & 1 & 1 & & 1 & \\
c & & 1 & & 1 & & 1 \\
d & 1 & & 1 & & 1 & \\
e & 1 & & & 1 & & 1 \\
f & & & 1 & & & \\
g & & & & 1 & & \\
\hline
\end{array}$$

Fig. 5. A graph G and its clique matrix. In the upper left of the matrix is a Tucker matrix, M_{III}. The only LB subgraph of G is the chordless cycle (b, c, e, d), which does not contain row a of the M_{III}. This illustrates that not every Tucker submatrix in a clique matrix can be extended to the clique matrix of an LB subgraph.

If G is not chordal, we may return a G_{III} by the algorithm of [18]. Henceforth, we may assume that the graph is chordal. A *clique tree* of a chordal graph is a tree that has one node for each maximal clique, and with the property that for each vertex v of G, the cliques that contain v induce a connected subtree. Every chordal graph has a clique tree, see for example [4]. The following Lemma is immediate from results that appear in [4].

Lemma 9. *Let T be a clique tree for a chordal graph G, and let K be a leaf. Then K contains a simplicial vertex of G. Let S be the simplicial vertices of K, and let T' be the result of deleting leaf K from T. Deleting S from G yields an induced subgraph that has T' as a clique tree.*

Definition 3. *By* shrinking a clique tree T*, let us denote the operation of deleting the set S of simplicial vertices in a leaf K of T, yielding a smaller graph G' with the smaller clique tree described by Lemma 9.*

Lemma 10. *Let G be a chordal graph and let M_T be a submatrix of a clique matrix of G that is an instance of M_I on three vertices or an instance of M_{II} – M_V. Then the clique matrix of an LB graph occurs in the submatrix of K induced by the columns of M_T, the rows of M_T, and three additional rows, one for each of the simplicial vertices depicted in Figure 2.*

Proof. Let T be the column numbers occupied by M_T in the clique matrix of G, let $i \in T$, let $\mathcal{K}$ be the cliques of G corresponding to columns of T, and K_i be the clique of G corresponding to the column i. Let V' be vertices corresponding to rows occupied by M_T, let $\mathcal{C} = \{K \cap V' | K \in \mathcal{K}\}$ and let $C_i = K_i \cap V'$. Let $\mathcal{C}' = \{C' \setminus C_i | C' \in \mathcal{C}$ and $C' \neq C_i\}$. If the intersection graph of $\mathcal{C}'$ is connected, then let us say that C_i is an *outsider* in M_T.

If C_i is an outsider, then in any clique tree of G, K_i does not lie on the path between any pair of members of $\mathcal{K}$ in the clique tree of G. This is seen as follows. Suppose K_i lies on the path P between two members of $\mathcal{K}$. Then removal of K_i from the clique tree separates the clique tree into two or more trees, at least two of which contain members of $\mathcal{K}$. Because the intersection graph of $\mathcal{C}'$ is connected, there exists $v \in V' \setminus C_i$ that resides in cliques in two of these trees. Since $v \notin C_i$, the subtree of the clique tree induced by cliques containing v is not connected, a contradiction.

Therefore, suppose we iteratively shrink the clique tree of G subject to the constraint that we do not shink any member of $\mathcal{K}$ when it becomes a leaf. The procedure halts when all leaves of the clique tree of the resulting graph G' are members of $\mathcal{K}$. If C_i is an outsider, K_i is a leaf in the clique tree of G', which means that it has a simplicial vertex s, by Lemma 9. The only column of T where s has a 1 is in column i, hence C_i is the set of neighbors of s in G'.

It is easily verified that in $M_{II}(k)$, the first and last two columns of $M_{II}(k)$ ($\{1, k-1\}$, $\{0, k-2\}$, $\{k-2, k-1\}$) are outsiders. These are the neighbor sets of $G_V(k+3)$. Therefore, the $M_{II}(k)$ can be extended to a submatrix that is the clique matrix of $G_V(k+3)$. The rows of this submatrix is an induced $G_V(k+3)$ in G, as illustrated on the righthand side of Figure 2. Similarly, the first, third and fifth column of M_{IV} and the first, third and fourth column of M_V the first column and last two columns of $M_{III}(k)$, and all columns of $M_I(3)$ are outsiders. Any instance of these submatrices in the clique matrix of a chordal graph can be extended to an instance of G_I, G_{II}, $G_{IV}(k+3)$, and $G_V(6)$, respectively, identifying an LB subgraph of G.

References

[1] Booth, S., Lueker, S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J. Comput. Syst. Sci. 13, 335–379 (1976)

[2] Brandstaedt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. SIAM Monographs on Discrete Mathematics, Philadelphia (1999)

[3] Chauve, C., Haus, U.-U., Stephen, T., You, V.P.: Minimal conflicting sets for the consecutive ones property in ancestral genome reconstruction. Journal of Computational Biology 17, 1167–1181 (2010)

[4] Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York (1980)

[5] Hell, P., Huang, J.: Certifying LexBFS recognition algorithms for proper inteval graphs and proper interval bigraphs. SIAM J. Discrete Math. 18, 554–570 (2004)

[6] Kratsch, D., McConnell, R.M., Mehlhorn, K., Spinrad, J.P.: Certifying algorithms. SIAM Journal on Computing 36, 236–353 (2006)

[7] Lekkerker, C., Boland, D.: Representation of finite graphs by a set of intervals on the real line. Fund. Math. 51, 45–64 (1962)

[8] Lueker, G.S., Booth, K.S.: A linear time algorithm for deciding interval graph isomorphism. J. ACM 26, 183–195 (1979)

[9] Neidermeier, R., Dom, M., Guo, J.: Approximation and fixed-parameter algorithms for consecutive ones submatrix problems. J. Comput. Syst. Sci. 76, 204–221 (2010)

[10] McConnell, R.M.: A certifying algorithm for the consecutive-ones property. In: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004), vol. 15, pp. 761–770 (2004)

[11] McConnell, R.M., Mehlhorn, K., Näher, S., Schweitzer, P.: Certifying algorithms. Computer Science Reviews 5, 119–161 (2011)

[12] Meidanis, J., Porto, O., Telles, G.P.: On the consecutive ones property. Discrete Applied Mathematics 88, 325–354 (1998)

[13] Roberts, F.S.: Graph Theory and Its Applications to Problems of Society. Society for Industrial and Applied Mathematics, Philadelphia (1978)

[14] Rose, D., Tarjan, R.E., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. 5, 266–283 (1976)

[15] Spinrad, J.: Efficient Graph Representations. American Mathematical Society, Providence (2003)

[16] Stoye, J., Wittler, R.: A unified approach for reconstructing ancience gene clusters. IEEE/ACM Transactions on Computational Biology and Bioinformatics 6, 387–400 (2009)

[17] Tamayo, M.: Algorithms for Finding Tucker Patterns. PhD thesis, Simon Fraser University (2013)

[18] Tarjan, E.E., Yannakakis, M.: Addendum: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM Journal on Computing 14, 254–255 (1985)

[19] Tucker, A.: A structure theorem for the consecutive 1's property. Journal of Combinatorial Theory, Series B 12, 153–162 (1972)

[20] Wegner, G.: Eigenschaften der Nerven Homologishe-Einfactor Familien in R^n. PhD thesis, Universität Göttingen (1967)

Certifying 3-Edge-Connectivity

Kurt Mehlhorn, Adrian Neumann, and Jens M. Schmidt

Max Planck Institute for Informatics, Saarbrücken, Germany

Abstract. We present a linear-time certifying algorithm that tests graphs for 3-edge-connectivity. If the input graph G is not 3-edge-connected, the algorithm returns a 2-edge-cut. If G is 3-edge-connected, the algorithm returns a construction sequence that constructs G from the graph with two nodes and three parallel edges using only operations that (obviously) preserve 3-edge-connectivity.

1 Introduction

Advanced graph algorithms answer complex yes-no questions such as "Is this graph planar?" or "Is this graph k-vertex-connected?". They are not only nontrivial to implement, it is also difficult to test their implementations, as usually only small test sets are available. It is hence possible that bugs persist unrecognized for a long time. An example is the linear time planarity test of Hopcroft and Tarjan [7] in LEDA [13]. A bug was discovered only after two years of intensive use.

Certifying algorithms [12] approach this problem by computing an additional *certificate* that proves the correctness of the answer. This may, e.g., be either a 2-coloring or an odd cycle for testing bipartiteness, or either a planar embedding or a Kuratowski subgraph for testing planarity. Certifying algorithms are designed such that checking the correctness of the certificate is substantially simpler than solving the original problem. Ideally, checking the correctness is so simple that the implementation of the checking routine allows for a formal verification. In that case, the solution of *every instance* is *correct by a formal proof* [1].

Our main result is a linear time certifying algorithm for 3-edge-connectivity based on a result of Mader [11]. He showed that every 3-edge-connected graph can be obtained from K_2^3, the graph consisting of two vertices and three parallel edges, by a sequence of three simple operations that each introduce one edge and, trivially, preserve 3-edge-connectivity. We show how to compute such a sequence in linear time for 3-edge-connected graphs. If the input graph is not 3-edge-connected, a 2-edge-cut is computed. The previous algorithms [6, 15, 22–24] for deciding 3-edge-connectivity are not certifying; they deliver a 2-edge-cut for graphs that are not 3-edge-connected but no certificate in the yes-case.

Our algorithm uses the concept of a *chain decomposition* of a graph introduced in [19]. A chain decomposition is an ear decomposition [10]. It is used in [21] as a common and simple framework for certifying 1- and 2-vertex, as well as 2-edge-connectivity. Further, [20] uses them for certifying 3-vertex-connectivity. Chain decompositions are an example of *path-based* algorithms (see, e.g., Gabow [5]), which use only the simple structure of certain paths in a DFS-tree to compute connectivity information about the graph.

We use chain decompositions to certify 3-edge-connectivity in linear time. Thus, chain decompositions form a common framework for certifying k-vertex- and k-edge-connectivity for $k \leq 3$ in linear time. We use many techniques from [20], but in a simpler form. Hence our paper may also be used as a gentle introduction to the 3-vertex-connectivity algorithm in [20].

Related Work. Deciding 3-edge-connectivity is a well researched problem, with applications in fields such as bioinformatics [4] and quantum chemistry [3]. Consequently, there are many linear time solutions known [6, 15, 22–24]. None of them is certifying.

The paper [12] is a recent survey on certifying algorithms. For a linear time certifying algorithm for 3-vertex-connectivity, see [20] (implemented in [16]). For general k, there is a randomized certifying algorithm for k-vertex connectivity in [9] with expected running time $O(kn^{2.5} + nk^{3.5})$. There is a non-certifying algorithm [8] for deciding k-edge-connectivity in time $O(m \log^3 n)$ w.h.p..

In [6], a linear time algorithm is described that transforms a graph G into a graph G' such that G is 3-edge-connected if and only if G' is 3-vertex-connected. Combined with this transformation, the certifying 3-vertex-connectivity algorithm from [20] certifies 3-edge-connectivity in linear time. However, that algorithm is much more complex than the algorithm given here. Moreover, we were unable to find an elegant method for transforming the certificate obtained for the 3-vertex-connectivity of G' into a certificate for 3-edge-connectivity of G.

2 Preliminaries

We consider finite undirected graphs G with n vertices, m edges, no self-loops, and minimum degree three, and use standard graph-theoretic terminology from [2], unless stated otherwise. We use uv to denote an edge with endpoints u and v.

A set of edges that leaves a disconnected graph upon deletion is called *edge cut*. For $k \geq 1$, let a graph G be *k-edge-connected* if $n \geq 2$ and there is no edge cut $X \subseteq E(G)$ with $|X| < k$. Let $v \rightarrow_G w$ denote a path P between two vertices v and w in G and let $s(P) = v$ and $t(P) = w$ be the source and target vertex of P, respectively. Every vertex in $P \setminus \{s(P), t(P)\}$ is called an *inner vertex* of P and every vertex in P is said to *lie on* P.

Let T be an undirected tree rooted at vertex r. For two vertices x and y in T, let x be an *ancestor* of y and y be a *descendant* of x if $x \in V(r \rightarrow_T y)$. If additionally $x \neq y$, x is a *proper* ancestor and y is a *proper* descendant. We write $x \leq y$ ($x < y$) if x is an ancestor (proper ancestor) of y. The parent $p(v)$ of a vertex v is its immediate proper ancestor. The parent function is undefined for r. Let K_2^m be the graph on 2 vertices that contains exactly m parallel edges.

Let *subdividing an edge* uv of a graph G be the operation that replaces uv with a path uzv, where z was not previously in G. All 3-edge-connected graphs can be constructed using a small set of operations starting from a K_2^3.

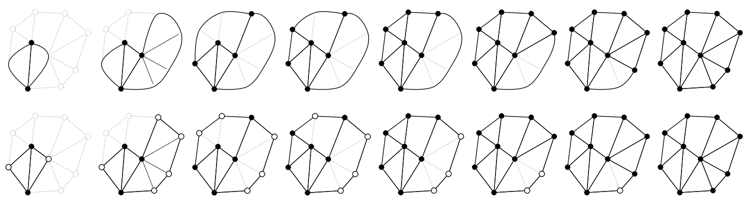


Fig. 1. Two ways of constructing the 3-edge-connected graph shown in the rightmost column. The upper row shows the construction according to Theorem 1. The lower row shows the construction according to Corollary 1. Branch (non-branch) vertices are depicted as filled (non-filled) circles. The black edges exist already, while dotted gray vertices and edges do not exist yet.

Theorem 1 (Mader [11]). *Every* 3*-edge-connected graph (and no other graph) can be constructed from a* K_2^3 *using the following three operations:*
- *Adding an edge (possibly parallel or a loop).*
- *Subdividing an edge* xy *and connecting the new vertex to any existing vertex.*
- *Subdividing two distinct edges* wx, yz *and connecting the two new vertices.*

A subdivision G' of a graph G is a graph obtained by subdividing edges zero or more times. The *branch vertices* of a subdivision are the vertices with degree at least three (we call the other vertices *non-branch*-vertices) and the *links* of a subdivision are the maximal paths whose inner vertices have degree two. If G has no vertex of degree two, the links of G' are in one-to-one correspondence to the edges of G. Theorem 1 readily generalizes to subdivisions of 3-edge-connected graphs.

Corollary 1. *Every subdivision of a* 3*-edge-connected graph (and no other graph) can be constructed from a subdivision of a* K_2^3 *using the following three operations:*
- *Adding a path connecting two branch vertices.*
- *Adding a path connecting a branch vertex and a non-branch vertex.*
- *Adding a path connecting two non-branch vertices lying on distinct links.*

In all three cases, the inner vertices of the path added are new vertices.

Each path that is added to a graph H in the process of Corollary 1 is called a *Mader-path* (*with respect to* H). Note that an ear is always a Mader-path unless both endpoints lie on the same link.

Figure 1 shows two constructions of a 3-edge-connected graph, one according to Theorem 1 and one according to Corollary 1. In this paper, we show how to find the Mader construction sequence according to Corollary 1 for a 3-edge-connected graph in linear time. Such a construction is readily turned into one according to Theorem 1.

3 Chain Decompositions

We use a very simple decomposition of graphs into cycles and paths. The decomposition was previously used for linear-time tests of 2-vertex- and 2-edge-connectivity [21] and 3-vertex-connectivity [20]. In this paper we show that it can also be used to find

Mader's construction for a 3-edge-connected graph. We define the decomposition algorithmically; a similar procedure that serves for the computation of low-points can be found in [18].

Let G be a connected graph without self-loops and let T be a depth-first search tree of G. Let r be the root of T. We orient tree-edges uv up, i.e., such that $v < u$, and back-edges xy down, that is, such that $x < y$.

We decompose[1] G into a set $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$ of cycles and paths, called *chains*, by applying the following procedure for each vertex v in the order in which they were discovered during the DFS.

First, we declare v visited[2]. Then, for every back-edge vw with $s(vw) = v$, we traverse $w \rightarrow_T r$ until a vertex x is encountered that was visited before; x is a descendant of v. The traversed subgraph $vw \cup (w \rightarrow_T x)$ forms a new *chain* C with $s(C) = v$ and $t(C) = x$. All inner vertices of C are declared visited. Observe that $s(C)$ and $t(C)$ are already visited when the construction of the chain starts.

Figure 2 illustrates these definitions. Since every back-edge defines one chain, there are precisely $m - n + 1$ chains. We number the chains in the order of their construction.

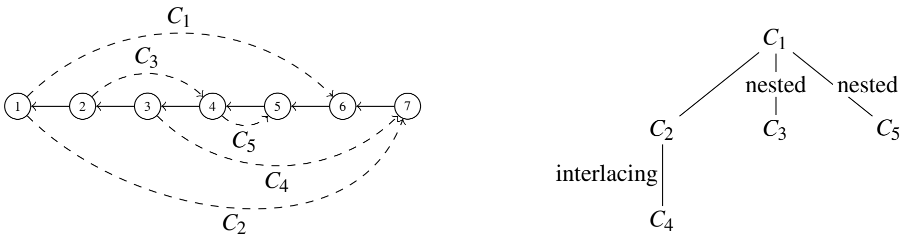


Fig. 2. The left side of the figure shows a DFS tree with a chain decomposition; tree-edges are solid and back-edges are dashed. C_1 is (16,65,54,43,32,21), C_2 is (17,76), C_3 is (24), C_4 is (37), and C_5 is (45). C_3 and C_5 are nested children of C_1 and C_4 is an interlacing child of C_2. Also, $s(C_4)$ s-belongs to C_1.

In the algorithm of Sect. 6, we start with $G_c = C_1 \cup C_2$. In the first phase, we form three segments, namely $\{C_4\}$, $\{C_3\}$, and $\{C_5\}$. The first segment can be added according to Lemma 4. Then C_3 can be added and then C_5.

We call $\mathcal{C}$ a *chain decomposition*. It can be computed in time $O(n+m)$. For 2-edge-connected graphs the term decomposition is justified by Lemma 1.

Lemma 1 ([21]). *Let $\mathcal{C}$ be a chain decomposition of a graph G. Then G is 2-edge-connected if and only if G is connected and the chains in $\mathcal{C}$ partition $E(G)$.*

Since the condition of Lemma 1 is easily checked, we assume from now on that G is 2-edge-connected. Then $\mathcal{C}$ partitions $E(G)$ and the first chain C_1 is a cycle containing r (since there is a back-edge incident to r). We say that r *strongly belongs (s-belongs)* to the first chain and any vertex $v \neq r$ *s-belongs* to the chain containing the edge $vp(v)$.

[1] If G is not 2-edge-connected, there will be edges and maybe vertices not belonging to any chain.

[2] Initially, no vertex is visited.

We use s-belongs instead of belongs since a vertex can belong to many chains when chains are viewed as sets of vertices.

We can now define a parent-tree on chains. The first chain C_1 has no parent. For any chain $C \neq C_1$, let the *parent* $p(C)$ of C be the chain to which $t(C)$ s-belongs. We write $C \leq D$ ($C < D$) for chains C and D if C is an ancestor (proper ancestor) of D in the parent-tree on chains.

The following lemma summarizes important properties of chain decompositions.

Lemma 2. *Let $\{C_1, \dots, C_{m-n+1}\}$ be a chain decomposition of a 2-edge-connected graph G and let r be the root of the DFS-tree. Then*
(1) For every chain C_i, $s(C_i) \leq t(C_i)$.
(2) Every chain C_i, $i \geq 2$, has a parent chain $p(C_i)$. We have $s(p(C_i)) \leq s(C_i)$ and $p(C_i) = C_j$ for some $j < i$.
(3) For $i \geq 2$: If $t(C_i) \neq r$, $t(p(C_i)) < t(C_i)$. If $t(C_i) = r$, $t(p(C_i)) = t(C_i)$.
(4) If $u \leq v$, u s-belongs to C, and v s-belongs to D then $C \leq D$.
(5) If $u \leq t(D)$ and u s-belongs to C, then $C \leq D$.
(6) For $i \geq 2$: $s(C_i)$ s-belongs to a chain C_j with $j < i$.

4 Chains as Mader-Paths

We show that, assuming that the input graph is 3-edge-connected, there are two chains that form a subdivision of a K_2^3, and that the other chains of the chain decomposition can be added one by one such that each chain is a Mader-path with respect to the union of the previously added chains. We will also show that chains can be added parent-first, i.e., when a chain is added, its parent was already added. In this way the current graph G_c consisting of the already added chains is always *parent-closed*. We will later show how to compute this ordering efficiently.

We assume that the input graph is 2-edge-connected. This is easily checked using Lemma 1. This guarantees that the chain decomposition is a partition of the edge set of the input graph. We will also use the following necessary condition for 3-edge-connectivity.

Proposition 1. *Let T be a DFS tree starting at r for a graph G. If G is 3-edge-connected, then the subtree of every child of r must be connected to r by at least two back-edges.*

Using the chain decomposition, we can identify a K_2^3 subdivision in the graph as follows. We may assume that the first two back-edges explored from r in the DFS have their other endpoint in the same subtree T' rooted at some child of r. The first chain C_1 forms a cycle. The vertices in $C_1 \setminus r$ are then contained in T'. By assumption, the second chain is constructed by another back-edge that connects r with a vertex in T'. If there is no such back-edge, Proposition 1 exhibits a 2-edge-cut, namely the tree-edge and the back-edge connecting r and T'. Let $x = t(C_2)$. Then $C_1 \cup C_2$ forms a K_2^3 subdivision with branch vertices r and x. The next lemma derives properties of parent-closed unions of chains.

Lemma 3. *Let G_c be a parent-closed union of chains that contains C_1 and C_2. Then*
(1) For any vertex $v \neq r$ of G_c, the edge $v\,p(v)$ is contained in G_c, i.e., the set of vertices of G_c is a parent-closed subset of the DFS-tree.

(2) $s(C)$ and $t(C)$ are branch vertices of G_c for every chain C contained in G_c.
(3) Let C be a chain that is not in G_c but a child of some chain in G_c. Then C is an ear with respect to G_c and the path $t(C) \to_T s(C)$ is contained in G_c. C is a Mader-path (i.e., the endpoints of C are not inner vertices of the same link of G_c) with respect to G_c if and only if there is a branch vertex on $t(C) \to_T s(C)$.

We can now prove that chains can always be added in parent-first order.

Theorem 2. *Let G graph and let G_c be a parent-closed union of chains such that no child of a chain $C \subset G_c$ is a Mader-path with respect to G_c and there is at least one such chain. Then the extremal edges of every link of length at least two in G_c are a 2-cut in G.*

Proof. Assume otherwise. Then there is a parent-closed union G_c of chains such that no child of a chain in G_c is a Mader-path with respect to G_c and there is at least one such child outside of G_c, but for every link in G_c the extremal edges are not a cut in G.

Consider any link L of G_c. Since the extremal edges of L do not form a 2-cut, there is a path connecting an inner vertex on L with a vertex that is either a branch vertex of G_c or a vertex on a link of G_c different from L. Let P be such a path of minimum length. By minimality, no inner vertex of P belongs to G_c. Note that P is a Mader-path with respect to G_c. We will show that at least one edge of P belongs to a chain C with $p(C) \in G_c$ and that C can be added, contradicting our choice of G_c.

Let a and b be the endpoints of P, let z be the lowest common ancestor of all points in P. Since a DFS generates only tree- and back-edges, z lies on P. Since $z \leq x$ for all $x \in P$, no inner vertex of P belongs to G_c, and the vertex set of G_c is a parent-closed subset of the DFS-tree, z is equal to a or b. Assume w.l.o.g. that $z = a$. All vertices of P are descendants of a. We view P as oriented from a to b.

Since b is a vertex of G_c, the path $b \to_T a$ is part of G_c by Lemma 2 and hence no inner vertex of P lies on this path. Let av be the first edge on P. The vertex v must be a descendant of b as otherwise the path $v \to_P b$ would contain a cross-edge, i.e. an edge between different subtrees. Hence av is a back-edge. Let D be the chain that starts with the edge av. D does not belong to G_c, as no edge of P belongs to G_c.

We claim that $t(D)$ is a proper descendant of b or D is a Mader-path with respect to G_c. Since v is a descendant of b and $t(D)$ is an ancestor of v, $t(D)$ is either a proper descendant of b, equal to b, or a proper ancestor of b. We consider each case separately.

If $t(D)$ were a proper ancestor of b the edge $b\,p(b)$ would belong to D and hence D would be part of G_c, contradicting our choice of P. If $t(D)$ is equal to b as then D is a Mader-path with respect to G_c. This leaves the case that $t(D)$ is a proper descendant of b.

Let yb be the last edge on the path $t(D) \to_T b$. We claim that yb is also the last edge of P. This holds since the last edge of P must come from a descendant of b (as ancestors of b belong to G_c) and since it cannot come from a child different from y as otherwise P would have to contain a cross-edge.

Let D^* be the chain containing yb. Then $D^* \leq D$ by Lemma 2.(5) (applied with $C = D^*$ and $u = y$) and hence $s(D^*) \leq s(D) \leq a$ by part (4) of the same lemma. Also $t(D^*) = b$. Since $b = t(D^*) \in G_c$, $p(D^*) \in G_c$.

As a and b are not inner vertices of the same link, the path $t(D^*) \to_T s(D^*)$ contains a branch vertex. Thus D^* is a Mader-path by Lemma 3. □

Corollary 2. *If G is 3-edge-connected, chains can be greedily added in parent-first order.*

Theorem 2 gives rise to an $O((n+m)\log(n+m))$ algorithm, the Greedy-Chain-Addition Algorithm. Details can be found in the full version of this paper.

5 A Classification of Chains

When we add a chain in the Greedy-Chain-Addition algorithm, we also process its children. Children that do not have both endpoints as inner nodes of the chain can be added to the list of addable chains immediately. However, children that have both endpoints as inner nodes of the chain cannot be added immediately and need to be observed further until they become addable. We now make this distinction explicit by classifying chains into two types, interlacing and nested.

We classify the chains $\{C_3,\dots C_{m-n+1}\}$ into two types. Let C be a chain with parent $\widehat{C}=p(C)$. We distinguish two cases[3] for C.

- If $s(C)$ is an ancestor of $t(\widehat{C})$ and a descendant of $s(\widehat{C})$, C is *interlacing*. We have $s(\widehat{C})\le s(C)\le t(\widehat{C})\le t(C)$.
- If $s(C)$ is a proper descendant of $t(\widehat{C})$, C is *nested*. We have $s(\widehat{C})\le t(\widehat{C})<s(C)\le t(C)$ and $t(C)\to_T s(C)$ is contained in $\widehat{C}$.

These cases are exhaustive as the following argument shows. Let $s(\widehat{C})v$ be the first edge on $\widehat{C}$. By Lemma 2, $s(\widehat{C})\le s(C)\le v$. We split the path $v\to_T s(\widehat{C})$ into two parts corresponding to the two cases above, namely $t(\widehat{C})\to_T s(\widehat{C})$, and $(v\to_T t(\widehat{C}))\backslash t(\widehat{C})$. Depending on which of these paths $s(C)$ lies, it is classified as interlacing or nested.

The following simple observations are useful. For any chain $C\ne C_1$, $t(C)$ s-belongs to $\widehat{C}$. If C is nested, $s(C)$ and $t(C)$ s-belong to $\widehat{C}$. If C is interlacing, $s(C)$ s-belongs to a chain which is a proper ancestor of $\widehat{C}$ or $\widehat{C}=C_1$. The next lemma confirms that interlacing chains can be added once their parent belongs to G_c.

Lemma 4. *Let G_c be a parent-closed union of chains that contains C_1 and C_2, let C be any chain contained in G_c, and let D be an interlacing child of C not contained in G_c. Then D is a Mader-path with respect to G_c.*

6 A Linear Time Algorithm

According to Lemma 4, interlacing chains whose parent already belongs to the current graph are always Mader-paths and can be added. Adding a chain may create new branching vertices which in turn can turn other chains into Mader-paths. This observation suggests adding interlacing chains as early as possible. Only when there is no interlacing chain to add, we need to consider nested chains. In that case and if the graph is 3-edge-connected, some nested chain must be addable (because a previously added

[3] In [20], three types of chains are distinguished. What we call nested is called Type 1 there and what we call interlacing is split into Types 2 and 3 there. We do not need this finer distinction.

Algorithm 1. Certifying linear-time algorithm for 3-edge connectivity.

```
procedure CONNECTIVITY(G=(V,E))
    Let $\{C_1, C_2, \dots, C_{m-n+1}\}$ be a chain decomposition of $G$ as described in Sect. 3;
    Initialize $G_c$ to $C_1 \cup C_2$;
    for $i$ from 1 to $m-n+1$ do
                                   ▷ Phase i: add all chains whose source s-belongs to $C_i$
        Group the chains $C$ for which $s(C)$ s-belongs to $C_i$ into segments;
                                   ▷ Part I of Phase i: add segments with interlacing root
        Add all segments whose minimal chain is interlacing to $G_c$;
                                   ▷ Part II of Phase i: add segments with nested root
        Either find an insertion order $S_1, \dots, S_k$ on the segments having a nested minimal chain
or exhibit a 2-edge-cut and stop;
        for $j$ from 1 to $k$ do
            Add the chains contained in $S_j$ parent-first;
        end for
    end for
end procedure
```

chain created a branching vertex on the tree-path from the sink to the source of the chain). The question is how to find this nested chain efficiently.

The following observation paves the way. Once we add a nested chain, its interlacing children and then their interlacing children etc. become addable. This suggests considering nested chains not in isolation, but to consider them together with their interlacing offspring. We formalize this intuition in the concept of segment below.

Nested chains have both endpoints on their parent chain. Consider the chains nested in chain C_i. Which chains can help their addition by creating branching points on C_i? First, chains nested in C_i and their interlacing offspring, and second, interlacing chains having their source on some C_j with $j < i$. Chains having their source on some C_j with $j > i$ cannot help because they have no endpoint on C_j. These observations suggest an algorithm operating in phases. In the i-th phase, we try to add all chains having their source vertex on C_i.

The overall structure of the linear-time algorithm is given in Algorithm 1. An implementation in Python is available at`https://github.com/adrianN/edge-connectivity`. The algorithm operates in phases and maintains a current graph G_c. Let C_1, C_2, ..., C_{m-n+1} the chains of the chain decomposition in the order of creation. We initialize G_c to $C_1 \cup C_2$. In phase i, $i \in [1, m-n+1]$, we consider the i-th chain C_i and either add all chains C to G_c for which the source vertex $s(C)$ s-belongs to C_i to G_c or exhibit a 2-edge-cut. As already mentioned, chains are added parent-first and hence G_c is always parent-closed. We maintain the following invariant:

Invariant: After phase i, G_c consists of all chains for which the source vertex s-belongs to one of the chains C_1 to C_i.

Lemma 5. *For all i, the current chain C_i is part of the current graph G_c at the beginning of phase i or the algorithm has exhibited a 2-edge-cut before phase i.*

The next lemma gives information about the chains for which the source vertex s-belongs to C_i. None of them belongs to G_c at the beginning of phase i (except for chain

C_2 that belongs to G_c at the beginning of phase 1) and they form subtrees of the chain tree. Only the roots of these subtrees can be nested. All other chains are interlacing.

Lemma 6. *Assume that the algorithm reaches phase i without exhibiting a 2-edge-cut. Let $C \neq C_2$ be a chain for which $s(C)$ s-belongs to C_i. Then C is not part of G_c at the beginning of phase i. Let D be any ancestor of C that is not in G_c. Then:*
(1) $s(D)$ s-belongs to C_i.
(2) If D is nested, it is a child of C_i.
(3) If $p(D)$ is not part of the current graph, D is interlacing.

We can now define the segments with respect to C_i. Consider the set $\mathscr{S}$ of chains whose source vertex s-belongs to C_i. For a chain $C \in \mathscr{S}$, let C^* be the minimal ancestor of C that does not belong to G_c. Two chains C and D in $\mathscr{S}$ belong to the same segment if and only if $C^* = D^*$, see Figure 2 for an illustration.

Consider any $C \in \mathscr{S}$. By part (1) of the preceding lemma either $p(C) \in \mathscr{S}$ or $p(C)$ is part of G_c. Moreover, C and $p(C)$ belong to the same segment in the first case. Thus segments correspond to subtrees in the chain tree. In any segment only the minimal chain can be nested by Lemma 6. If it is nested, it is a child of C_i (parts (2) and (3) of the preceding lemma). Since only the root of a segment may be a nested chain, once it is added to the current graph all other chains in the segment can be added in parent-first order by Lemma 4. All that remains is to find the proper ordering of the segments . We do so in Lemma 10. If no proper ordering exists, we exhibit a 2-edge-cut.

Lemma 7. *All chains in a segment S can be added in parent-first order if its minimal chain can be added.*

It is easy to determine the segments with respect to C_i. We iterate over all chains C whose source $s(C)$ s-belongs to C_i. For each such chain, we traverse the path C, $p(C)$, $p(p(C))$, ...until we reach a chain that belongs to G_c or is already marked[4]. In the former case, we distinguish cases. If the last chain on the path is nested we mark all chains on the path with the nested chain. If we hit a marked chain we copy the marker to all chains in the path. Otherwise, i.e., all chains are interlacing and unmarked, we add all chains in the path to G_c in parent-first order, as this segment can be added according to Corollary 7. We have now completed part I of phase i, namely the addition of all segments whose minimal chain is interlacing. We have also determined the segments with nested minimal chain.

It remains to compute a proper ordering of the segments in which the minimal chain is nested or to exhibit a 2-edge-cut. We do so in part II of phase i. For simplicity, we will say ‘segment’ instead of ‘segment containing a nested chain’ from now on.

For a segment S let the *attachment points* of S be all vertices in S that are in G_c. Note that the attachment points must necessarily be endpoints of chains in S and hence adding the chains of S makes the attachment points branch vertices. Nested children C of C_i can be added if there are branch vertices on $t(C) \rightarrow_T s(C)$, therefore adding a segment can make it possible to add further segments.

Lemma 8. *Let C be a nested child of C_i and let S be the segment containing C. Then all attachment points of S lie on the path $t(C) \rightarrow_T s(C)$ and hence on C_i.*

[4] Initially, all chains are unmarked

For a set of segments $S_1, \ldots, S_k$, let the *overlap graph* be the graph on the segments and a special vertex R for the branch vertices on C_i. In the overlap graph, there is an edge between R and a vertex S_i, if there are attachment points $a_1 \leq a_2$ of S_i such that there is a branch vertex on the tree path $a_2 \rightarrow_T a_1$. Further, between two vertices S_i and S_j there is an edge if there are attachment points a_1, a_2 in S_i and b_1, b_2 in S_j, such that $a_1 \leq b_1 \leq a_2 \leq b_2$ or $b_1 \leq a_1 \leq b_2 \leq a_2$. We say that S_i and S_j *overlap*.

Lemma 9. *Let $\mathscr{C}$ be a connected component of the overlap graph H and let S be any segment with respect to C_i whose minimal chain C is nested. Then $S \in \mathscr{C}$ if and only if*
(i) $R \in \mathscr{C}$ and there is a branch vertex on $t(C) \rightarrow_T s(C)$ or
(ii) there are attachments a_1 and a_2 of S and attachments b_1 and b_2 of segments in $\mathscr{C}$ with $a_1 \leq b_1 \leq a_2 \leq b_2$ or $b_1 \leq a_1 \leq b_2 \leq a_2$.

Lemma 10. *Assume the algorithm reaches phase i. If the overlap graph H induced by the segments with respect to C_i is connected, we can add all segments of C_i. If H is not connected, we can exhibit a 2-edge-cut for any component of H that does not contain R.*

It remains to show that we can find an order as required in Lemma 10, or a 2-edge-cut, in linear time. We reduce the problem of finding an order on the segments to a problem on intervals. W.l.o.g. assume that the vertices of C_i are numbered consecutively from 1 to $|C_i|$. Consider any segment S, and let $a_0 \leq a_1 \leq \ldots \leq a_k$ be the set of attachment points of S, i.e., the set of vertices that S has in common with C_i. We associate the intervals $\{[a_0, a_\ell] | 1 \leq \ell \leq k\} \cup \{[a_\ell, a_k] | 1 \leq \ell < k\}$, with S and for every branch vertex v on C_i we define an interval $[0, v]$. See Figure 3 for an example.

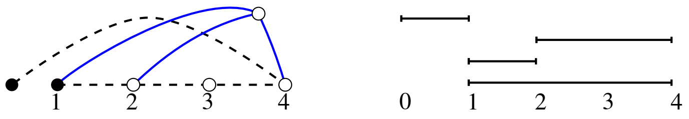


Fig. 3. Intervals for the solid segment with attachment points 1,2,4

We say two intervals $[a, a'], [b, b']$ *overlap* if $a \leq b \leq a' \leq b'$. Note that overlapping is different from intersecting; an interval does not overlap intervals in which it is properly contained or which it properly contains. This relation naturally induces a graph H' on the intervals. Contracting all intervals that belong to the same segment makes H' isomorphic to the overlap graph as required for Lemma 10. Hence we can use H' to find the order on the segments.

A naive approach that constructs H', contracts intervals, and runs a DFS will fail, since the overlap graph can have a quadratic number of edges. However, using a method developed by Olariu and Zomaya [17], we can compute a spanning forest of H' in time linear in the number of intervals. The presentation in [17] is for the PRAM and thus needlessly complicated for our purposes. A simpler explanation can be found in the full version of this paper [14].

Since the number of intervals created for a chain C_i is bounded by $|\mathrm{NC}(C_i)| + 2|\mathrm{In}(C_i)| + |V_{\mathrm{branch}}(C_i)|$, where $\mathrm{NC}(C_i)$ are the nested children of C_i, $\mathrm{In}(C_i)$ are the interlacing chains that start on C_i, and $V_{\mathrm{branch}}(C_i)$ is the set of branch vertices on C_i, the total time spent on this procedure for all chains is $O(m)$. From the above discussion follows:

Theorem 3. *For a* 3*-edge-connected graph, a Mader construction sequence can be found in time* $O(n+m)$.

7 Verifying the Certificate

The certificate is either a 2-edge-cut, or a sequence of Mader-paths. For a 2-edge-cut, we simply remove the two edges and verify that G is no longer connected.

Checking the Mader sequence is slightly more involved. We assume that each edge in a Mader-path is doubly linked to the corresponding edge in G. Let G' be a copy of G. We remove the Mader-paths again, in reverse order, suppressing vertices of degree two as they occur. This can create multiple edges and loops. Let G'_i be the multi-graph before we remove the i-th path P_i. There are several things that we need to verify:

- G must have minimum degree three.
- The union of Mader-paths must be isomorphic to G and the Mader-paths must partition the edges of G. This is easy to check using the links between the edges of the paths and the edges of G.
- The paths we remove must be ears. More precisely, at step i, P_i must have been reduced to a single edge in G'_i, as inner vertices of P_i must have been suppressed if P_i is an ear for G'_i.
- The P_i must not subdivide the same link twice. That is, after deleting the edge corresponding to P_i, it must not be the case that both endpoints are still adjacent (or equal, i.e. P_i is a loop) but have degree two.
- When only two paths are left, the graph must be a K_2^3.

8 Conclusion

We presented a certifying linear time algorithm for 3-edge-connectivity based on chain decompositions of graphs. It is simple enough for use in a classroom setting and can serve as a gentle introduction to the certifying 3-vertex-connectivity algorithm of [20]. We also provide an implementation in Python, available at `https://github.com/adrianN/edge-connectivity`.

There remain some open problems. Foremost, our algorithm only computes one 2-edge-cut. Is it possible to compute the 3-edge-connected components easily?

Mader's construction sequence is general enough to construct k-edge-connected graphs for any $k \geq 3$, and can thus be used in certifying algorithms for larger k. So far, though, it is unclear how to compute these more complicated construction sequences. We hope that the chain decomposition framework can be adapted to work in these cases too.

References

1. Alkassar, E., Böhme, S., Mehlhorn, K., Rizkallah, C.: Verification of certifying computations. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 67–82. Springer, Heidelberg (2011)
2. Bondy, J.A., Murty, U.S.R.: Graph Theory. Springer (2008)

3. Corcoran, J.N., Schneider, U., Schüttler, H.-B.: Perfect stochastic summation in high order feynman graph expansions. International Journal of Modern Physics C 17(11), 1527–1549 (2006)
4. Dehne, F., Langston, M.A., Luo, X., Pitre, S., Shaw, P., Zhang, Y.: The cluster editing problem: Implementations and experiments. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 13–24. Springer, Heidelberg (2006)
5. Gabow, H.N.: Path-based depth-first search for strong and biconnected components. Inf. Process. Lett. 74(3-4), 107–114 (2000)
6. Galil, Z., Italiano, G.F.: Reducing edge connectivity to vertex connectivity. SIGACT News 22(1), 57–61 (1991)
7. Hopcroft, J., Tarjan, R.: Efficient planarity testing. Journal of the ACM (JACM) 21(4), 549–568 (1974)
8. Karger, D.R.: Minimum cuts in near-linear time. J. ACM 47(1), 46–76 (2000)
9. Linial, N., Lovász, L., Wigderson, A.: Rubber bands, convex embeddings and graph connectivity. Combinatorica 8(1), 91–102 (1988)
10. Lovász, L.: Computing ears and branchings in parallel. In: Proceedings of the 26th Annual Symposium on Foundations of Computer Science, FOCS 1985 (1985)
11. Mader, W.: A reduction method for edge-connectivity in graphs. In: Bollobás, B. (ed.) Advances in Graph Theory. Annals of Discrete Mathematics, vol. 3, pp. 145–164 (1978)
12. McConnell, R.M., Mehlhorn, K., Näher, S., Schweitzer, P.: Certifying algorithms. Computer Science Review 5(2), 119–161 (2011)
13. Mehlhorn, K., Näher, S., Uhrig, C.: The LEDA Platform of Combinatorial and Geometric Computing. Cambridge University Press (1999)
14. Mehlhorn, K., Neumann, A., Schmidt, J.M.: Certifying 3-edge-connectivity. CoRR, abs/1211.6553 (2012)
15. Nagamochi, H., Ibaraki, T.: A linear time algorithm for computing 3-edge-connected components in a multigraph. Japan Journal of Industrial and Applied Mathematics 9, 163–180 (1992)
16. Neumann, A.: Implementation of Schmidt's algorithm for certifying triconnectivity testing. Master's thesis, Universität des Saarlandes and Graduate School of CS, Germany (2011)
17. Olariu, S., Zomaya, A.Y.: A time- and cost-optimal algorithm for interlocking sets – With applications. IEEE Trans. Parallel Distrib. Syst. 7(10), 1009–1025 (1996)
18. Ramachandran, V.: Parallel open ear decomposition with applications to graph biconnectivity and triconnectivity. In: Synthesis of Parallel Algorithms, pp. 275–340 (1993)
19. Schmidt, J.M.: Contractions, removals and certifying 3-connectivity in linear time. Tech. Report B 10-04, Freie Universität Berlin, Germany (May 2010)
20. Schmidt, J.M.: Contractions, removals and certifying 3-connectivity in linear time. SIAM Journal on Computing 42(2), 494–535 (2013)
21. Schmidt, J.M.: A simple test on 2-vertex- and 2-edge-connectivity. Information Processing Letters 113(7), 241–244 (2013)
22. Taoka, S., Watanabe, T., Onaga, K.: A linear time algorithm for computing all 3-edge-connected components of a multigraph. IEICE Trans. Fundamentals E75(3), 410–424 (1992)
23. Tsin, Y.H.: A simple 3-edge-connected component algorithm. Theor. Comp. Sys. 40(2), 125–142 (2007)
24. Tsin, Y.H.: Yet another optimal algorithm for 3-edge-connectivity. J. of Discrete Algorithms 7(1), 130–146 (2009)

Parameterized Algorithms for MAX COLORABLE INDUCED SUBGRAPH Problem on Perfect Graphs

Neeldhara Misra[1], Fahad Panolan[2], Ashutosh Rai[2], Venkatesh Raman[2], and Saket Saurabh[2]

[1] Indian Institute of Science, Bangalore, India
neeldhara@csa.iisc.ernet.in
[2] Institute of Mathematical Sciences, Chennai, India
{fahad,ashutosh,vraman,saket}@imsc.res.in

Abstract. We address the parameterized complexity of MAX COLORABLE INDUCED SUBGRAPH on perfect graphs. The problem asks for a maximum sized q-colorable induced subgraph of an input graph G. Yannakakis and Gavril [*IPL 1987*] showed that this problem is NP-complete even on split graphs if q is part of input, but gave a $n^{O(q)}$ algorithm on chordal graphs. We first observe that the problem is W[2]-hard parameterized by q, even on split graphs. However, when parameterized by ℓ, the number of vertices in the solution, we give two fixed-parameter tractable algorithms.

– The first algorithm runs in time $5.44^{\ell}(n + \#\alpha(G))^{O(1)}$ where $\#\alpha(G)$ is the number of maximal independent sets of the input graph.
– The second algorithm runs in time $q^{\ell+o(\ell)}n^{O(1)}T_{\alpha}$ where T_{α} is the time required to find a maximum independent set in any induced subgraph of G.

The first algorithm is efficient when the input graph contains only polynomially many maximal independent sets; for example split graphs and co-chordal graphs. The running time of the second algorithm is FPT in ℓ alone (whenever T_{α} is a polynomial in n), since $q \leq \ell$ for all non-trivial situations. Finally, we show that (under standard complexity-theoretic assumptions) the problem does not admit a polynomial kernel on split and perfect graphs in the following sense:

(a) On split graphs, we do not expect a polynomial kernel if q is a part of the input.
(b) On perfect graphs, we do not expect a polynomial kernel even for fixed values of $q \geq 2$.

1 Introduction

A fundamental class of graph optimization problems involve finding a maximum induced subgraph satisfying specific properties, such as being edgeless (maximum independent set) [4,5,6,19], acyclic [9], bipartite [4,5], regular [12] or q-colorable [1,20] (equivalent to finding a maximum independent set when $q = 1$, and a maximum induced bipartite subgraph when $q = 2$). Several of these problems are NP-hard on general undirected graphs. Therefore, studies of these problems have involved algorithmic paradigms designed to cope with NP-hardness, like approximation and parameterization [4,5,9,6,19,20]. The focus of this paper is the Max q-Colorable Induced Subgraph problem, with a special focus on co-chordal graphs and perfect graphs. Our results are of a parameterized flavor, involving both FPT algorithms and lower bounds for polynomial kernels.

Before we can describe our results, we establish some basic notions. A graph $G = (V, E)$ is called *q-colorable* if there is a coloring function $f : V \to [q]$ such that $f(u) \neq f(v)$ for any $(u, v) \in E$. Equivalently, a graph is q-colorable if its vertex set can be partitioned into q independent sets. The Max q-Colorable Induced Subgraph asks for a maximum induced subgraph that is q-colorable, and the decision version, p-mcis, may be stated as follows:

p-Max Colorable Induced Subgraph (p-mcis) **Parameter:** ℓ
Input: An undirected graph $G = (V, E)$ and positive integers q and ℓ.
Question: Does there exist $Z \subseteq V$, $|Z| \geq \ell$, such that $G[Z]$ is q-colorable?

We will sometimes be concerned with the problem above for *fixed values of* q, and to distinguish this from the case when q is a part of the input, we use p-q-mcis to refer to the version where q is fixed. The problem is clearly NP-complete on general graphs as for $q = 1$ this corresponds to Independent Set problem. Yannakakis and Gavril [20] showed that this problem is NP-complete even on split graphs (which is a proper subset of perfect graphs, chordal graphs and co-chordal graphs, see Section 2 for definitions). However, they showed that p-q-mcis is solvable in time $n^{O(q)}$ on chordal graphs. A natural question, therefore, is whether the problem admits an algorithm with running time $f(q) \cdot n^{O(1)}$ on chordal graphs, or even on split graphs. This question was our main motivation for looking at p-mcis on special graph classes like co-chordal and perfect graphs.

Our study of p-mcis involves determining the parameterized complexity of the problem. The goal of parameterized complexity is to find ways of solving NP-hard problems more efficiently than brute force: here the aim is to restrict the combinatorial explosion to a parameter that is hopefully much smaller than the input size. Formally, a *parametrization* of a problem is assigning an integer k to each input instance and we say that a parameterized problem is *fixed-parameter tractable (*FPT*)* if there is an algorithm that solves the problem in time $f(k) \cdot |I|^{O(1)}$, where $|I|$ is the size of the input and f is an arbitrary computable function depending on the parameter k only. Just as NP-hardness is used as evidence that a problem probably is not polynomial time solvable, there exists a hierarchy of complexity classes above FPT, and showing that a parameterized problem is hard for one of these classes gives evidence that the problem is unlikely to be fixed-parameter tractable. The principal analogue of the classical intractability class NP is W[1]. A convenient source of W[1]-hardness reductions is provided by the result that Independent Set parameterized by solution size is complete for W[1]. Other highlights of the theory include that Dominating Set, by contrast, is complete for W[2]. For more background, the reader is referred to the monographs [8]. A parameterized problem is said to admit a *polynomial kernel* if every instance (I, k) can be reduced in polynomial time to an equivalent instance with both size and parameter value bounded by a polynomial in k. The study of kernelization is a major research frontier of parameterized complexity and many important recent advances in the area are on kernelization. The recent development of a framework for ruling out polynomial kernels under certain complexity-theoretic assumptions [3,7,10] has added a new dimension to the field and strengthened its connections to classical complexity. For overviews of kernelization we refer to surveys [2,11] and to the corresponding chapters in books on parameterized complexity [8,18].

Our results and related work. Most of the "induced subgraph problems" are known to be W-hard parameterized by the solution size on general graphs by a generic result of Khot and Raman [14]. In particular this also implies that p-MCIS is W[1]-hard parameterized by the solution size on general graphs. Observe that INDEPENDENT SET is essentially p-MCIS with $q = 1$. There has been also some study of parameterized complexity of INDEPENDENT SET on special graph classes [6,19]. Yannakakis and Gavril [20] showed that p-MCIS is NP-complete on split graphs and Addario-Berry et al. [1] showed that the problem is NP-complete on perfect graphs for every fixed $q \geq 2$. We observe in passing that the known NP-completeness reduction given in [20] implies that p-MCIS when parameterized by q alone is W[2]-hard even on split graphs. Our main contributions in this paper are two randomized FPT algorithms for p-MCIS and a complementary lower bound, which establishes the non-existence of a polynomial kernel under standard complexity-theoretic assumptions.

Our first algorithm runs in time $(2e)^\ell(n + \#\alpha(G))^{O(1)}$ where $\#\alpha(G)$ is the number of maximal independent sets of the input graph and the second algorithm runs in time $q^\ell \cdot T_\alpha \cdot n^{O(1)}$, where T_α is the time required to compute the largest independent set in any subgraph of the given graph. Observe that since $q \leq \ell$ for all non-trivial situations, we have that the second algorithm is FPT in ℓ alone, provided T_α is a polynomial in n. The first algorithm is efficient when the input graph contains only polynomially many maximal independent sets; for example on split graphs and co-chordal graphs. The second algorithm is efficient for a larger class of graphs, because it only relies on an efficient procedure for finding a maximum independent set (although this comes at the cost of the running time depending on q in the base of the exponent). In particular, the second algorithm runs in time $q^\ell n^{O(1)}$ on the class of perfect graphs. We also describe de-randomization procedures. While the derandomization technique for the first algorithm is standard, to derandomize the second algorithm we need a notion which generalizes the idea of "universal sets", introduced by Naor et al. [16]. We believe that our construction, though simple, could be of independent interest. Further, we show that unless CO-NP $\subseteq$ NP/poly, the problem does not admit polynomial kernel even on split graphs. Also, on perfect graphs, we show that the problem does not admit a polynomial kernel even for fixed $q \geq 2$, unless CO-NP $\subseteq$ NP/poly.

2 Preliminaries and Definitions

For a finite set V, a pair $G = (V, E)$ such that $E \subseteq V^2$ is a graph on V. The elements of V are called *vertices*, while pairs of vertices (u, v) such that $(u, v) \in E$

are called *edges*. We also use $V(G)$ and $E(G)$ to denote the vertex set and the edge set of G, respectively. In the following, let $G = (V, E)$ and $G' = (V', E')$ be graphs, and let $U \subseteq V$ be some subset of vertices of G. Let G' be a subgraph of G. If E' contains all the edges $\{u, v\} \in E$ with $u, v \in V'$, then G' is an *induced subgraph* of G, *induced by* V', denoted by $G[V']$. For any $U \subseteq V$, we denote $G[V \setminus U]$ by $G \setminus U$. For $v \in V$, $N_G(v) = \{u \mid (u, v) \in E\}$. The complement of a graph $G = (V, E)$, denoted by $\bar{G}$, is the graph with vertex set V and edge set $V \times V \setminus (E \cup \{(v, v) \mid v \in V\})$. A set $X \subseteq V$ is called a clique (resp., independent set) if every pair of vertices in X is adjacent (resp., non-adjacent) in G. X is called a *maximal* clique (resp., independent set), if no proper super set of X is clique (resp., independent set). We denote the size of the maximum clique in graph G by $w(G)$. A graph G is q-colorable if we can partition the vertex set in to q independent sets. The chromatic number $\chi(G)$ of a graph G is the minimum q such that G is q-colorable.

A graph G is called *perfect*, if $\forall\ U \subseteq V(G)$, $w(G[U]) = \chi(G[U])$. A graph $G = (V, E)$ is called *chordal* if every simple cycle of with more than three vertices has an edge connecting two nonconsecutive vertices on the cycle. A graph is *co-chordal* if its complement is a chordal graph. All chordal graphs and co-chordal graphs are perfect graphs. A *split graph* is a graph whose vertex set can be partitioned into two subsets I and Q such that I is an independent set and Q is a clique. Split graphs are closed under complementation. We denote the set $\{1, 2, \ldots, n\}$ by $[n]$ and all possible subsets of size k of $[n]$ by $\binom{[n]}{k}$.

Definition 2.1. *Let $G = (V, E)$ and $H_x = (V_x, E_x)$ for $x \in V$ be graphs. We define the graph $G' = Embed(G; (H_x)_{x \in V})$ as the graph obtained from G by replacing each vertex x with the graph H_x . Formally, $V(G') = \{u_x | x \in V,\ u \in V_x\}$ and $E(G') = \{(u_x, v_x) | (u, v) \in E_x\} \cup \{(u_x, v_y) | (x, y) \in E,\ u \in V_x,\ v \in V_y\}$.*

We say that the graph $Embed(G;\ (H_x)_{x \in V})$ is obtained by embedding $(H_x)_{x \in V}$ into G. We say that a graph class Π is closed under embedding if whenever $G \in \Pi$ and $H_x \in \Pi$, $\forall x \in V(G)$, then the graph $Embed(G;\ (H_x)_{x \in V(G)})$ belongs to Π. It is known that perfect graphs are closed under embedding [15]. Let $G = (V, E)$ be a graph and $E' \subseteq E$. We define the graph $\Delta(G; E')$ as adding vertices x_e and edges (x_e, u), (x_e, v) for all $(u, v) = e \in E'$.

Lemma 2.1 ($\star$). *If $G = (V, E)$ is a perfect graph and $E' \subseteq E$, then $\Delta(G; E')$ is also a perfect graph.*

Due to space constraints, some proofs have been deferred to a full version of the paper. Results whose proofs are omitted are marked with a $\star$.

3 Generalized Universal Sets

In this section we generalize a derandomization tool, *universal sets* given by Naor et al. [16].

Definition 3.1. *An (n, k, q)-universal set is a set of vectors $V \subseteq [q]^n$ such that for any index set $S \in \binom{[n]}{k}$, the projection of V on S contains all possible q^k configurations.*

Theorem 3.1 ($\star$). *An (n, k, q)-universal set of cardinality $q^k k^{O(\log k)} \log^2 n$ can be constructed deterministically in time $O(q^k k^{O(\log k)} n \log^2 n)$.*

Definition 3.2 ([16]). *Let H be a family of functions from $[n]$ to $[l]$. H is an (n, k, l)-family of perfect hash functions if for all $S \in \binom{[n]}{k}$, there is an $h \in H$ which is one-to-one on S.*

Theorem 3.2 ([16]). *There is a deterministic algorithm with running time $O(e^k k^{O(\log k)} n \log n)$ that constructs an (n, k, k)-family of perfect hash functions $\mathcal{F}$ such that $|\mathcal{F}| = e^k k^{O(\log k)} \log n$.*

4 FPT Algorithms

In this section we design two randomized algorithms for p-MCIS. The first algorithm requires a subroutine that enumerates all maximal independent sets in the input graph and this algorithm is useful only when the input graph has polynomially many maximal independent sets. We can derandomize this algorithm using a (n, ℓ, ℓ)-family of perfect hash functions.

The second algorithm requires a subroutine which computes the *maximum* independent set of any induced subgraph of the input graph. Thus, this algorithm is FPT on all graph classes for which Independent Set is either polynomial time solvable or FPT parameterized by the solution size. We derandomize this algorithm using the (n, ℓ, q)-universal sets described in the previous section.

Notice that the second algorithm is less demanding than the first: we only need to find the largest independent set, rather than enumerating all maximal ones. Thus the second algorithm solves the problem for a larger class of graphs than the first, however, as we will see, the running time is compromised in that a dependence on q creeps into the base of the exponent. In particular, this is why the second algorithm does not render the first obsolete. The first can be thought of as a more efficient algorithm when the class of graphs was restricted further.

Algorithm based on enumerating Maximal Indepenent Sets. Let $\#\alpha(G)$ denote the number of maximal independent sets of G, and $T_{\#\alpha}(G)$ denote the time taken to enumerate the maximal independent sets of a graph G. In this section we give a randomized algorithm with one sided error for p-MCIS that uses all the maximal independent sets in the graph, runs in time $T_{\#\alpha}(G) + 2^\ell(n + \#\alpha)^{O(1)}$, and gives the correct answer with probability at least $e^{-\ell}$. The error is one-sided: if the input instance is No instance, then the algorithm will output No always. Thus, in any graph class where the maximal independent sets can be enumerated in polynomial time, we can solve p-MCIS with constant success probability in $O((2e)^\ell n^{O(1)})$ time.

Algorithm 1. An Algorithm for p-MCIS based on enumerating MIS.

```
Input:  A graph $G = (V, E)$ and positive integers $\ell, q$
Output: YES, if there exists $S \subseteq V$, $|S| = \ell$ and $G[S]$ is $q$-colorable, NO otherwise.

1. Enumerate all maximal independent sets in $G$. Let $M = \{m_1, m_2, \dots, m_t\}$ be the
   set of all maximal independent sets.
2. Construct a split graph $G' = (V \uplus M, E' = \{(v, m_i) | \; m_i \in M, \; v \in V \cap m_i\})$, where
   $G'[M]$ is a clique.
3. Color each vertex in $V$ with a color from an $\ell$-sized set of colors uniformly at
   random.
4. Merge all vertices in each color class into a single vertex. Formally, replace each
   color class $C_i$ by a single vertex $c_i$, and let $N(c_i) = \{u \mid \exists v \in C_i, (u, v) \in E'\}$. Let
   the graph after contraction be $G^* = (C \uplus M, E^*)$.
5. If there exists a partition of $C$ into $q$ sets $C_1, C_2, \dots, C_q$ such that for all $i$, $C_i$ has a
   common neighbor in $M$, then output YES, otherwise output NO. (This is based on
   a Steiner Tree computation with $C$ as terminals, see the proof for a description.)
```

Lemma 4.1. *Algorithm 1 runs in time $O(2^\ell n^{O(1)})$ on graphs where the maximal independent sets can be enumerated in polynomial time. Further, if (G, ℓ, q) is a* YES *instance of p-MCIS, then Algorithm 1 will output* YES *with probability at least e^{-l}, otherwise Algorithm 1 will output* NO *with probability* 1.

Proof. We first argue the running time bound. Since we assume that maximal independent sets are enumerable in polynomial time, Steps 1—4 are clearly polynomial time. To find the partition in Step 5, we run a Steiner Tree algorithm on the instance with C given as the set of terminals. We claim that a partition of the desired kind exists if and only if there exists a Steiner Tree using at most q additional vertices to connect the terminal set C. First, if the set C can be connected with at most q additional vertices $\{s_1, \dots, s_q\}$ from M, then notice that the non-terminal vertices in the Steiner Tree constitute a dominating set for C (indeed, any non-dominated vertex c_i is necessarily disconnected from $C \setminus \{c_i\}$). Therefore, $\{N(s_i) \setminus \bigcup_{1 \le j < i} N(s_j) \mid 1 \le i \le q\}$ gives the desired partition. On the other hand, suppose we have a partition of C into q sets $C_1, C_2, \dots, C_q$ such that for all i, C_i has a common neighbor s_i in M. Note that the set $S := \{s_1, \dots, s_q\}$ is a Steiner Tree for C: given $x \in C_i$ and $y \in C_j$, the path $(x, s_i), (s_i, s_j), (s_j, y)$ (where $s_i = s_j$ if $i = j$) lies in $C \cup S$. Since finding the optimal Steiner Tree on an instance with k terminals can be done in $O(2^k n^{O(1)})$ time [17], we have that the last step of the algorithm runs in time $O(2^\ell n^{O(1)})$.

We now show the correctness of the algorithm whenever the output is positive. Suppose Algorithm 1 outputs YES. Then there exist q vertices in M that dominates all vertices in C which implies at least one vertex in each color class that is dominated by one or more of these q vertices. In particular, there exists a subset $T \subseteq V$ with ℓ vertices and a subset $S \subseteq M$ with q vertices, such that S dominates T. We argue that $G[T]$ is the desired q-colorable subgraph. Let $T := \{v_1, v_2, \dots, v_\ell\}$. For each v_i, let $c(v_i)$ be the smallest j for which v_i is dominated by m_j. Notice that c defines a partition of T into q sets. For all

Algorithm 2. An Algorithm for p-MCIS based on finding maximum IS.

Input: A graph $G = (V, E)$ and a positive integers ℓ, q
Output: YES, if there exists $S \subseteq V$, $|S| = l$ and $G[S]$ is q-colorable, NO otherwise.

- Color the graph uniformly at random with q colors. Let C_i be the color classes for $1 \leq i \leq q$.
- Find the maximum independent sets H_i for each C_i.
- If $|\bigcup_{1 \leq i \leq q} H_i| \geq \ell$, say YES, otherwise say NO.

$1 \leq j \leq q$, it is clear that $c^{-1}(j)$ is a subset of some maximal independent set, and hence the proposed partition is a proper coloring. Therefore, (G, ℓ, q) is a YES instance of p-MCIS.

We now argue the probability that the algorithm finds a solution given that the input is a YES instance. Let (G, ℓ, q) be a YES instance of p-MCIS, and let $T \subseteq V$ with $|T| = \ell$, be a solution. When we randomly color the vertices, each vertex in T will get different colors with probability $\frac{\ell!}{\ell^\ell} \geq e^{-\ell}$. If T gets different colors then there exists q sets in M which dominate C because there exists a maximal independent set that contains each color class in $G[T]$ (since $G[T]$ is q-colorable). Hence Algorithm 1 will output YES with probability at least e^{-l}. □

We can boost the success probability to a constant by executing Algorithm 1 e^ℓ times, in which case the success probability will be at least $(1 - e^{-\ell})^{e^\ell} \geq \frac{1}{e}$. It is easy to see that we can derandomize the algorithm using a (n, ℓ, ℓ)-family of perfect hash functions (see Theorem 3.2) to obtain a deterministic algorithm with running time $(2e)^\ell \ell^{O(\log \ell)} n^{O(1)}$ for p-MCIS on graph classes for which maximal independent sets can be enumerated in polynomial time. Since the number of maximal cliques in chordal graphs with n vertices is bounded by n and all maximal cliques in chordal graphs can be enumerated in polynomial in n time, the number of independent sets in co-chordal graphs are bounded by linear in n and they can be enumerated in polynomial in n time as well. We therefore have the following corollary:

Corollary 4.1. *p-MCIS can be solved in time $(2e)^\ell \cdot \ell^{O(\log \ell)} n^{O(1)}$ on co-chordal graphs and split graphs.*

Algorithm based on finding a Maximum Independent Set. In Algorithm 2, we describe a randomized polynomial time algorithm which succeeds with probability $q^{-\ell}$ on graph classes where MAXIMUM INDEPENDENT SET can be solved in polynomial time.

Lemma 4.2 (⋆). *If (G, ℓ, q) is a YES instance of p-MCIS, then Algorithm 2 will output YES with probability $q^{-\ell}$, otherwise Algorithm 2 will output NO with probability 1. The algorithm runs in time $T_\alpha \cdot n^{O(1)}$, where T_α is the time required to find a maximum independent set up to size l in any induced subgraph of G.*

Corollary 4.2. *The problem of finding a ℓ-sized q-colorable subgraph on perfect graphs can be solved in time $q^\ell \ell^{O(\log \ell)} n^{O(1)}$.*

5 Kernelization Lower Bounds

In this section we show that MAX INDUCED BIPARTITE SUBGRAPH (i.e, q=2 in p-MCIS) on perfect graphs and p-MCIS on split graphs do not admit polynomial kernels unless CO-NP $\subseteq$ NP/poly.

Lower bound Machinery We begin by stating some of the known techniques developed for showing some problems do not admit polynomial kernels under standard complexity theoretic assumptions.

Definition 5.1 (Composition [3]). *A composition algorithm (also called OR-composition algorithm) for a parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that receives as input a sequence $((x_1, k), ..., (x_t, k))$, with $(x_i, k) \in \Sigma^* \times \mathbb{N}$ for each $1 \leq i \leq t$, uses time polynomial in $\sum_{i=1}^{t} |x_i|+k$, and outputs $(y, k') \in \Sigma^* \times \mathbb{N}$ with (a) $(y, k') \in \Pi \iff (x_i, k) \in \Pi$ for some $1 \leq i \leq t$ and (b) k' is polynomial in k. A parameterized problem is compositional (or OR-compositional) if there is a composition algorithm for it.*

We define the notion of the *unparameterized version* of a parameterized problem Π. The mapping of parameterized problems to unparameterized problems is done by mapping (x, k) to the string $x\#1^k$, where $\# \in \Sigma$ denotes the blank letter and 1 is an arbitrary letter in Σ. In this way, the unparameterized version of a parameterized problem Π is the language $\tilde{\Pi} = \{x\#1^k | (x, k) \in \Pi\}$. The following theorem yields the desired connection between the two notions.

Theorem 5.1 ([3,10]). *Let Π be a compositional parameterized problem whose unparameterized version $\tilde{\Pi}$ is NP-complete. Then, if Π has a polynomial kernel then* CO-NP $\subseteq$ NP/*poly.*

5.1 Max Induced Bipartite Subgraph on Perfect and Split Graphs

The MAX INDUCED BIPARTITE SUBGRAPH problem is formally given as follows:

MAX INDUCED BIPARTITE SUBGRAPH (p-MIBS) **Parameter:** k
Input: An undirected graph $G = (V, E)$ and a positive integer k.
Question: Does there exist $S \subseteq V$ such that $|S| = k$ and $G[S]$ is bipartite?

Here, we show that unless CO-NP $\subseteq$ NP/poly, p-MIBS does not have a polynomial kernel when restricted to perfect graphs. We note that we are dealing here with the case of finding a maximum induced bipartite subgraph in the interest of exposition; a more general result that shows the hardness of finding a maximum induced q-colorable subgraph for any fixed $q \geq 2$ on the class of perfect graphs is described in the full version of this work.

Our result here is established by demonstrating an OR-composition. Let $(G_0, k), (G_1, k), \ldots, (G_{t-1}, k)$ be t instances of p-MIBS, where every G_i is a perfect graph. Notice that we may assume that $t \leq 2^{k \log k+k}$. This is because, by Corollary 4.2, we may solve p-MIBS in time $2^{k \log k+k}$ (note that $q = 2$) on perfect graphs. Therefore, if $t > 2^{k \log k+k}$, then we may solve every instance in time

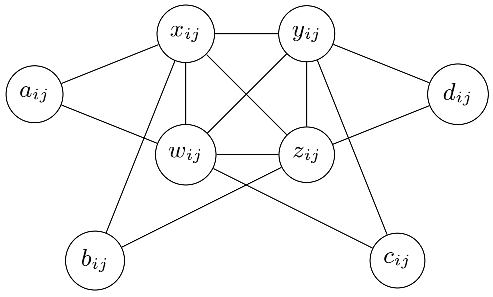


Fig. 1. Identity gadget H_{ij}

$t \cdot 2^{k \log k + k} < t^2$, and return a trivial YES or NO instance as the output of the composition, depending on whether there was at least one YES instance or not, respectively.

Thus, we assume that $t \leq 2^{k \log k + k}$, and therefore, $\log t \leq k^{O(1)}$. For convenience, we assume that t is a power of two (so that $\log t$ is an integral value). This can be done by padding the set of instances with trivial NO instances, and at most doubling the number of instances. We construct a composed instance (G, k^*) as follows. To begin with, let G be the disjoint union of all G_i, $0 \leq i \leq t-1$. For all $i \neq j$ add all possible edges between G_i and G_j.

Now add $2k \log t$ identity gadgets, named H_{ij} for $1 \leq i \leq 2k$, $1 \leq j \leq \log t$. The gadget H_{ij} consists of eight vertices $\{x_{ij}, y_{ij}, w_{ij}, z_{ij}, a_{ij}, b_{ij}, c_{ij}, d_{ij}\}$, where the vertices $\{x_{ij}, y_{ij}, w_{ij}, z_{ij}\}$ form a clique, and the vertex a_{ij} is adjacent to x_{ij} and w_{ij}; b_{ij} is adjacent to x_{ij} and z_{ij}; c_{ij} is adjacent to w_{ij} and y_{ij} and d_{ij} is adjacent to y_{ij} and z_{ij} (see Fig 1). For all $0 \leq l \leq t-1$, if the j^{th} bit of the $\log t$-bit binary representation of l is 0, then add edges from all vertices in G_l to x_{ij} and y_{ij}. Otherwise add edges from all vertices in G_l to w_{ij} and z_{ij}. This completes the description of the composed graph; we let $k^* = k + 12k \log t \leq k + 12k(k + k \log k) = O(k^2 \log k)$. Having shown that k^* is polynomially dependent on k, for simplicity, in the remaining discussion we continue refer to k^* in terms of t. We first show that this is indeed a valid OR-composition, and then demonstrate that G, as described, is a perfect graph.

Lemma 5.1. *The instance $(G, k + 12k \log t)$ is a* YES *instance of p-*MIBS *if, and only if, (G_l, k) is a* YES *instance of p-*MIBS *for some $0 \leq l \leq (t-1)$.*

Proof. ($\Rightarrow$) Assume $(G, k + 12k \log t)$ is a YES instance of p-MIBS and let $S \subseteq V(G)$ be a solution. We first claim that S will not contain vertices from more than two input instances. Indeed, suppose not. Then for $i_1 \neq i_2 \neq i_3$, let $v_{i_1} \in S \cap V(G_{i_1})$, $v_{i_2} \in S \cap V(G_{i_3})$ and $v_{i_3} \in S \cap V(G_{i_3})$. Note that $v_{i_1}, v_{i_2}, v_{i_3}$ will induce a triangle and contradict the fact that $G[S]$ is bipartite. We now assume that S contains vertices from two input graphs G_p and G_q. If one of them has at least k vertices in S, then we are done. Otherwise, $|S \cap V(G_p)| + |S \cap V(G_q)| < 2k$. Hence,

$$\sum_{i=1}^{2k}\sum_{j=1}^{\log t} |S \cap V(H_{ij})| > k + 12k\log t - 2k \geq 12k\log t - k$$

Therefore, by an averaging argument, there exists an i' such that $\sum_{j=1}^{\log t} |S \cap V(H_{i'j})| \geq 6\log t$. Since vertices $x_{ij}, y_{ij}, w_{ij}, z_{ij}$ from H_{ij} form a complete graph, S can contain at most 2 vertices from $\{x_{ij}, y_{ij}, w_{ij}, z_{ij}\}$. So $|S \cap V(H_{ij})| \leq 6$ and if $|S \cap V(H_{ij})| = 6$ then either $S \cap V(H_{ij}) = \{a_{ij}, b_{ij}, c_{ij}, d_{ij}, x_{ij}, y_{ij}\}$ or $S \cap V(H_{ij}) = \{a_{ij}, b_{ij}, c_{ij}, d_{ij}, w_{ij}, z_{ij}\}$. We know that to meet the budget, it must be the case that $\forall j,\ |S \cap V(H_{i'j})| = 6$.

Since $p \neq q$ there exists a j' such that j'^{th} bit of binary representation of p and q are different (say 0 and 1, respectively). Hence, all the vertices from G_p are connected to $x_{i'j'}, y_{i'j'}$ and all the vertices from G_q are connected to $w_{i'j'}, z_{i'j'}$. Hence there exists a triangle in $G[S \cap (V(G_p) \cup V(G_q) \cup V(H_{i'j'}))]$. This contradicts the fact that $G[S]$ is bipartite, showing that the case $|S \cap V(G_p)| + |S \cap V(G_q)| < 2k$ is infeasible. The remaining case is when S contains vertices from at most one input graph (say G_p). Since $|S \cap V(H_{ij})| \leq 6$, S will contain at least k vertices from $V(G_p)$. Hence $S \cap V(G_p)$ is a solution of (G_p, k).
($\Leftarrow$) Let (G_p, k) be a YES instance of p-MIBS, and let $S \subseteq V(G_p)$ be the solution. Let $b_1 b_2 \ldots b_{\log t}$ be the binary representation of p. Now consider the vertex set

$$\begin{aligned} T := \{x_{ij}, y_{i,j} \mid 1 \leq i \leq 2k \ \wedge\ b_j = 1\} \cup \{w_{ij}, z_{i,j} \mid 1 \leq i \leq 2k \ \wedge\ b_j = 0\} \\ \cup \{a_{ij}, b_{ij}, c_{ij}, d_{ij} \mid 1 \leq i \leq 2k \ \wedge\ 1 \leq j \leq \log t\}. \end{aligned} \tag{1}$$

It is easy to see that T involves exactly six vertices from each of the $2k\log t$ gadgets, and the vertices are chosen such that $G[T]$ induces a bipartite graph. Further, the vertices are chosen to ensure that there are no edges between vertices in S and vertices in T, and therefore, it is clear that $G[S \cup T]$ induces a bipartite subgraph of G of the desired size. Hence $(G, k + 12k\log t)$ is a YES instance of p-MIBS. □

Lemma 5.2. *The graph G constructed as the output of the OR-composition is a perfect graph.*

Proof. We begin by describing an auxiliary graph G', and show that G' is perfect. This graph is designed to be a graph from which G can be obtained by a series of operations that preserve perfectness, and this will lead us to establishing that G is perfect. The graph G' contains a clique on t vertices, K_t. We let $V(K_t) := \{v_0, v_1, \ldots v_{t-1}\}$. G' also contains $2k\log t$ small graphs, each of which consist of two vertices with an edge between them (i.e, each small graph is an edge). Let $\{n_{ij}, p_{ij}\}$ for all $1 \leq i \leq 2k$, $1 \leq j \leq \log t$ be the vertices of small graphs. For all $0 \leq l \leq t-1$, if the j^{th} bit of the $\log t$-bit binary representation of l is 0, then add edges from v_l to n_{ij} for all i. Otherwise add edges from v_l to p_{ij} for all i.

We claim that G' is perfect. Let H be an induced subgraph of G'. If $|V(H) \cap V(K_t)| \leq 1$, then H is a forest and so in this case $\omega(H) = \chi(H)$. Else, $r =$

$|V(H) \cap V(K_t)| \geq 2$. Since the neighborhoods of n_{ij} and p_{ij} do not intersect, and there are no edges between small graphs in G', at most one vertex from the entire set of small graphs can be part of the largest clique in H containing $V(H) \cap V(K_t)$ (note that there exists a largest clique that contains all the vertices in $V(H) \cap V(K_t)$). So $\omega(H) \leq r+1$. Let us denote by H^* the subgraph $H[V(H) \cap \{n_{ij}, p_{ij} \mid 1 \leq i \leq 2k, 1 \leq j \leq \log t\}]$.

If $\omega(H) = r+1$, then we define the following coloring. Color all r vertices in $V(H) \cap V(K_t)$ with colors $1, 2, \ldots, r$. For all $x \in V(H^*)$ such that x is adjacent to all vertices in $V(H) \cap V(K_t)$, we give a color $r+1$ (note that these vertices are independent by construction). If an $x \in V(H^*)$ is not adjacent to all vertices in $V(H) \cap V(K_t)$, then we can color it with a color that is already used on one of its non adjacent vertices in $V(H) \cap V(K_t)$. If $\omega(H) = r$, then there is no vertex in $V(H^*)$ which is adjacent to $V(H) \cap V(K_t)$. So we can color vertices in $V(H) \cap V(K_t)$ with r colors and for a vertex $x \in V(H^*)$ we can color x with a color same as (one of) its non adjacent vertex in $V(H) \cap V(K_t)$.Hence $\omega(H) = \chi(H)$.

Let be G^* be a graph obtained by embedding G_i on $v_i \in V(G')$ for all $0 \leq i \leq t-1$ and embedding an edge on each vertex in $\{n_{ij}, p_{ij} \mid 1 \leq i \leq 2k, 1 \leq j \leq \log t\}$. It can be observed that G^* is isomorphic to

$$G \setminus \bigcup_{1 \leq i \leq 2k, 1 \leq j \leq \log t} \{a_{ij}, b_{ij}, c_{ij}, d_{ij}\}.$$

It follows that G^* is perfect. Finally, observe that the graph G is $\Delta(G^*; E')$ for a suitable choice of $E' \subseteq E(G^*)$, and it follows that G is perfect. □

Lemmas 5.1,5.2 and Theorem 5.1, give us the following result.

Theorem 5.2. p-Max Induced Bipartite Subgraph *on perfect graphs does not admit a polynomial kernel unless* co-NP $\subseteq$ NP/*poly.*

We finally show that p-mcis does not admit a polynomial kernel on split graphs unless co-NP $\subseteq$ NP/poly by showing a "parameter-preserving reduction" from Small Universe Set Cover.

Theorem 5.3 ($\star$). p-mcis *on split graphs does not admit a polynomial kernel unless* co-NP $\subseteq$ NP/*poly.*

6 Conclusion

In this paper we studied the parameterized complexity of p-mcis on perfect graphs and showed that the problem is FPT when parameterized by the solution size. We also studied its kernelization complexity and showed that the problem does not admit polynomial kernel under certain complexity theory assumptions. An interesting direction of research that this paper opens up is the study of parameterized complexity of Induced Subgraph Isomorphism on special graph classes. As a first step it would be interesting to study the parameterized complexity of Induced Tree Isomorphism parameterized by the size of the tree on perfect graphs.

References

1. Addario-Berry, L., Kennedy, W.S., King, A.D., Li, Z., Reed, B.A.: Finding a maximum-weight induced k-partite subgraph of an i-triangulated graph. Discrete Applied Mathematics 158(7), 765–770 (2010)
2. Bodlaender, H.L.: Kernelization: New upper and lower bound techniques. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 17–37. Springer, Heidelberg (2009)
3. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. Syst. Sci. 75(8), 423–434 (2009)
4. Byskov, J.M.: Algorithms for k-colouring and finding maximal independent sets. In: SODA, pp. 456–457 (2003)
5. Byskov, J.M.: Enumerating maximal independent sets with applications to graph colouring. Oper. Res. Lett. 32(6), 547–556 (2004)
6. Dabrowski, K., Lozin, V.V., Müller, H., Rautenbach, D.: Parameterized algorithms for the independent set problem in some hereditary graph classes. In: Iliopoulos, C.S., Smyth, W.F. (eds.) IWOCA 2010. LNCS, vol. 6460, pp. 1–9. Springer, Heidelberg (2011)
7. Dell, H., van Melkebeek, D.: Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In: STOC, pp. 251–260 (2010)
8. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag New York, Inc., Secaucus (2006)
9. Fomin, F.V., Gaspers, S., Pyatkin, A.V., Razgon, I.: On the minimum feedback vertex set problem: Exact and enumeration algorithms. Algorithmica 52(2), 293–307 (2008)
10. Fortnow, L., Santhanam, R.: Infeasibility of instance compression and succinct PCPs for NP. In: STOC, pp. 133–142 (2008)
11. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. SIGACT News 38(1), 31–45 (2007)
12. Gupta, S., Raman, V., Saurabh, S.: Maximum r-regular induced subgraph problem: Fast exponential algorithms and combinatorial bounds. SIAM J. Discrete Math. 26(4), 1758–1780 (2012)
13. Johnson, D.S., Papadimitriou, C.H., Yannakakis, M.: On generating all maximal independent sets. Inf. Process. Lett. 27(3), 119–123 (1988)
14. Khot, S., Raman, V.: Parameterized complexity of finding subgraphs with hereditary properties. Theor. Comput. Sci. 289(2), 997–1008 (2002)
15. Lovasz, L.: Perfect graphs. In: Beineke, L.W., Wilson, R.J. (eds.) Selected Topics in Graph Theory, vol. 2, pp. 55–67. Academic Press, London (1983)
16. Naor, M., Schulman, L.J., Srinivasan, A.: Splitters and near-optimal derandomization. In: FOCS, pp. 182–191 (1995)
17. Nederlof, J.: Fast polynomial-space algorithms using möbius inversion: Improving on steiner tree and related problems. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 713–725. Springer, Heidelberg (2009)
18. Niedermeier, R.: Invitation to Fixed Parameter Algorithms. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, USA (2006)
19. Raman, V., Saurabh, S.: Short cycles make w-hard problems hard: Fpt algorithms for w-hard problems in graphs with no short cycles. Algorithmica 52(2), 203–225 (2008)
20. Yannakakis, M., Gavril, F.: The maximum k-colorable subgraph problem for chordal graphs. Inf. Process. Lett. 24(2), 133–137 (1987)

Characterizing and Computing the Structure of Clique Intersections in Strongly Chordal Graphs

Ragnar Nevries and Christian Rosenke

Department of Computer Science, University of Rostock
{ragnar.nevries,christian.rosenke}@uni-rostock.de

Abstract. In this paper, we present the clique arrangement $\mathcal{A}(G)$ for a chordal graph G to describe the intersections between the maximal cliques of G more precisely than in clique trees or related concepts. In particular, the node set of $\mathcal{A}(G)$ consists of all intersections of maximal cliques of G. In $\mathcal{A}(G)$, there is an arc from a node X to a node Z, if X is a subset of Z and there is no node Y, that is a superset of X and a subset of Z.

We provide a new characterization of strongly chordal graphs in terms of forbidden cyclic structures in the corresponding clique arrangements and we show how to compute the clique arrangement of a strongly chordal graph efficiently.

1 Introduction

Chordal graphs are an important generalization of trees that has been studied and characterized extensively. For a comprehensive survey refer to [2]. By definition, in a chordal graph, every cycle of length at least four has a chord, that is, an edge between two not consecutive vertices. Consequently, chordal graphs are also characterized by forbidden induced cycles on at least four vertices. This is a natural generalization of trees, where induced cycles of length three are forbidden as well. Moreover, chordal graphs are a generalization of trees in terms of decomposition. Whereas trees are the graphs that can be completely decomposed by repeatedly detaching leaves, this can be done on chordal graphs by detaching vertices that are adjacent to cliques. Such a decomposition of a chordal graph is called perfect elimination ordering.

The characterization of chordal graphs in terms of clique trees illustrates the close relation to trees most impressively. A clique tree T of a graph G is a tree with the maximal cliques of G as nodes, such that for every vertex v of G, the maximal cliques containing v induce a subtree in T. In [10], McKee shows that chordal graphs are exactly the graphs that admit a clique tree.

Clique trees have proven to represent the structure of chordal graphs fairly well by being utilized in many algorithms solving complex problems. In particular, the edges of a clique tree illustrate the structure of intersections between maximal cliques of the corresponding graph G and they identify all minimal vertex separators of G. In fact, Ho and Lee [8] show that a pair of maximal cliques

C_1 and C_2 forms an edge in some clique tree of G, if and only if $C_1 \cap C_2$ is a minimal vertex separator of G.

The structure of a chordal graph G, in particular the structure of its maximal clique intersections, is not entirely described by a corresponding clique tree. One can easily find quite different chordal graphs admitting isomorphic clique trees. For this reason, researchers developed a number of other descriptive graphs, using the maximal cliques as the node set, to capture their mutual intersection more accurately. In the clique graph developed by Shibata [12], two nodes are joined by an edge, if and only if the corresponding maximal cliques of G have a nonempty intersection. Bernstein and Goodman [1] describe a more general data structure called weighted clique intersection graph, which adds a weight to every edge specifying the cardinality of the intersection. Interestingly, McKee [10] shows that every maximum spanning tree of this graph is a clique tree for the original graph. Based on this, Galinier, Habib and Paul [7] give a graph that contains only the edges occurring in at least one maximum spanning tree, hence, in at least one clique tree, and thus, just the edges that represent minimal vertex separators.

All these concepts are not suited well to understand the intersections of more than two maximal cliques, in particular the intersections between minimal vertex separators. One promising attempt to better describe the clique intersections in a descriptive graph structure has been made by Ibarra in [9], introducing the clique-separator graph. This graph takes the union of all maximal cliques and all minimal vertex separators of G into its node set. Two nodes X and Z are connected by an edge XZ, if and only if X is a proper subset of Z and there is no other node Y with $X \subset Y \subset Z$. Ibarra shows that the clique-separator graph can be constructed in time $O(n^3)$ for chordal graphs on n vertices and improves this time to $O(n^2)$ for interval graphs and to $O(n \log n)$ for unit interval graphs.

In [9], Ibarra asks if subclasses of chordal graphs can be characterized in terms of the clique-separator graph, in particular strongly chordal graphs, the well-studied subclass introduced by Farber in [5]. Strongly chordal graphs are defined as the chordal graphs that have an odd chord in every cycle of even length. They have been characterized in many ways, for instance as the chordal graphs without induced k-suns for any $k \geq 3$ [5] and by the existence of strong elimination orderings [13], special kinds of perfect elimination orderings.

In fact, the clique-separator graph is not precise enough to distinguish between chordal graphs and strongly chordal graphs. For example, Figure 1 shows the 3-sun, which is not strongly chordal, and the net, which is strongly chordal. These two graphs have an isomorphic clique-separator graph. This happens, because the intersections between maximal cliques do not necessarily need to be minimal vertex separators and, consequently, such intersections and their relations are missed in the clique-separator graph.

To approach this problem, we propose the clique arrangement $\mathcal{A}(G)$ in Section 3. The node set of the clique arrangement consists of all intersections of a subset of maximal cliques of G. Like in the clique-separator graph, two nodes X

and Z are connected by an arc XZ if and only if $X \subset Z$ and there is no node Y with $X \subset Y \subset Z$.

Such a data structure has been used earlier in the analysis of relational database schemes [4] and was called Bachman diagram. But the structural properties of clique arrangements on chordal graphs have yet been analyzed only for ptolemaic graphs [14], a subclass of strongly chordal graphs. Uehara and Uno show that a graph is ptolemaic if and only if the clique arrangement is a tree. Therefore, the clique arrangement is called clique laminar tree in their paper. On the other hand, clique arrangements turn out to be infeasibly large for general chordal graphs, as their size may grow exponentially in the number of vertices in G, which is demonstrated in Section 3.

This paper generalizes the work of Uehara and Uno [14] and characterizes clique arrangements for strongly chordal graphs. In Section 4, we show that a graph G is strongly chordal if and only if the clique arrangement $\mathcal{A}(G)$ does not contain a specific cyclic structure. This is a generalization of the ptolemaic case, where the complete absence of cycles is known.

In Section 5, we give an algorithm that computes $\mathcal{A}(G)$ in linear time, if a strong elimination ordering of G is given, hence, in time $O(m \log n)$ for strongly chordal graphs on n vertices and m edges. This also implies that, in contrast to general chordal graphs, the clique arrangement of strongly chordal graphs cannot grow exponentially in the size of the input. Moreover, the algorithm marks all nodes in the arrangement that are maximal cliques and all nodes that are minimal vertex separators of the graph. With this information, we can construct the clique-separator graph in linear time from the clique arrangement. Hence, our algorithm can be used to construct the clique-separator graph for strongly chordal graphs in time $O(m \log n)$, which improves the running time of $O(n^3)$ given by Ibarra.

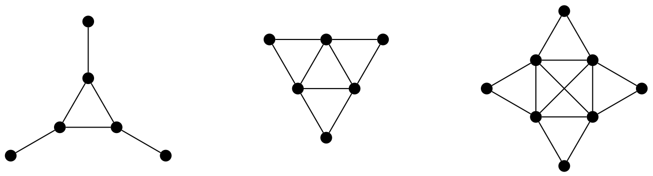


Fig. 1. The graphs net (left), 3-sun (center) and 4-sun (right)

Because of space limitations all proofs are omitted.

2 Preliminaries

For a comprehensive survey on graph classes we would like to refer to [2].

In this paper, all graphs are simple, without loops and, if not stated explicitly, with vertex set V and edge set E, where $|V| = n$ and $|E| = m$. An *independent set* in G is a set of mutually nonadjacent vertices. Conversely, a *clique* $C \subseteq V$ in G is a set of mutually adjacent vertices and C is called maximal, if there is no clique C' with $C \subset C'$. The set of all maximal cliques of a graph G is denoted by $\mathcal{C}(G)$.

For any subset $U \subseteq V$ we let $G[U] = (U, E_U)$ denote the subgraph of G induced by U. We simply write $G - U$ for $G[V \setminus U]$ and $G - x$ for $G - \{x\}$.

We call a set $U = \{x_1, \ldots, x_k\} \subseteq V$ an *induced k-cycle* in G, if $G[U]$ contains exactly the edges x_1x_k and x_i, x_{i+1} for all $i \in \{1, \ldots, k-1\}$. A vertex set $U = \{x_0, \ldots, x_{k-1}, y_0, \ldots, y_{k-1}\} \subseteq V$ induces a *k-sun* in G, if $G[U]$ is partitioned into the clique $X = \{x_0, \ldots, x_{k-1}\}$ and the independent set $Y = \{y_0, \ldots, y_{k-1}\}$ and for every edge x_iy_i between X and Y, either i equals j or $i+1$ equals j, where the indices are counted modulo k. By definition, a graph is *chordal*, if and only if it does not contain induced k-cycles for all $k \geq 4$. By Farber [5], a chordal graph is *strongly chordal*, if and only if every cycle of even length has an odd chord, that is, an edge between two vertices that have odd distance on the cycle. Farber also characterizes strongly chordal graphs by forbidden induced k-suns for all $k \geq 3$.

For all vertices $x \in V$, we let $N(x) = \{y \mid xy \in E\}$ denote the *open neighborhood* and $N[x] = N(x) \cup \{x\}$ the *closed neighborhood* of x in G. A vertex $x \in V$ is *simplicial* in G, if $N(x)$ is a clique. An ordering $x_1, \ldots, x_n$ of G's vertices is called *perfect elimination ordering*, if x_i is simplicial in $G[\{x_i, \ldots, x_n\}]$ for all $i \in \{1, \ldots, n\}$. If either $N[y] \subseteq N[z]$ or $N[z] \subseteq N[y]$ for all $y, z \in N(x)$, then a simplicial vertex x is called *simple* and accordingly a perfect elimination ordering is called *simple elimination ordering*, if x_i is simple in $G[\{x_i, \ldots, x_n\}]$ for all $i \in \{1, \ldots, n\}$. Moreover, a simple elimination ordering is called *strong elimination ordering*, if for all $i, j, k : 1 \leq i < j < k \leq n$ with $x_j, x_k \in N(x_i)$, $N[x_j] \cap \{x_i, \ldots, x_n\} \subseteq N[x_k] \cap \{x_i, \ldots, x_n\}$. According to Dirac [3], chordal graphs are characterized by the existence of perfect elimination orderings and by Paige and Tarjan [13], a graph is strongly chordal, if and only if it admits a simple elimination ordering. Interestingly, a graph also is strongly chordal, if and only if it admits a strong elimination ordering.

For all $x, y \in V$, $S \subset V$ is an *xy-separator*, if x and y are not connected in $G - S$. Moreover, S is a *minimal xy-separator*, if there is no xy-separator S' with $S' \subset S$. We call $S \subseteq V$ *separator* for short, if S is a minimal xy-separator for at least two vertices $x, y \in V$ and $\mathcal{S}(G)$ denotes the set of all separators in G.

We also use directed graphs in this paper. Actually, the data structure proposed in Section 3 is an acyclic directed graph. Directed edges are called *arcs*. For a directed graph $G = (V, A)$, a sequence $x_1, \ldots, x_k$ of vertices is called *directed path*, if $x_ix_{i+1} \in A$ for all $i \in \{1, \ldots, k-1\}$.

The data structure proposed in this paper is related to the *clique-separator graph* $\mathcal{CS}(G) = (\mathcal{C} \cup \mathcal{S}, E \cup A)$, which was introduced by Ibarra [9] and is a mixed graph built on the node set $\mathcal{C} \cup \mathcal{S}$ with $\mathcal{C} = \mathcal{C}(G)$ and $\mathcal{S} = \mathcal{S}(G)$. Any pair of nodes $X \in \mathcal{S}, Z \in \mathcal{C} \cup \mathcal{S}$ is adjacent by an edge of E or an arc of A, if and only if $X \subset Z$ and there is no other node $Y \in \mathcal{C} \cup \mathcal{S}$ with $X \subset Y \subset Z$. In particular, $XZ \in E$ if $Z \in \mathcal{C}$, and $XZ \in A$ if $Z \in \mathcal{S}$.

3 Clique Arrangements for Chordal Graphs

For a chordal graph $G = (V, E)$ we call a directed acyclic graph $\mathcal{A}(G) = (\mathcal{X}, \mathcal{E})$ the *clique arrangement* of G, if the node set $\mathcal{X}$ contains exactly all nonempty intersections of maximal cliques of G, that is,

$$\mathcal{X} = \left\{ X \;\middle|\; X = \bigcap_{C \in \mathcal{C}} C \text{ with } \mathcal{C} \subseteq \mathcal{C}(G) \text{ and } X \neq \emptyset \right\},$$

and the arc set $\mathcal{E}$ describes their mutual inclusion, that is,

$$\mathcal{E} = \{XZ \mid X, Z \in \mathcal{X} \text{ with } X \subset Z \text{ and } \nexists Y \in \mathcal{X} : X \subset Y \subset Z\}.$$

Naturally, we can find the set $\mathcal{C}(G)$ in $\mathcal{X}$ by focusing on the sinks of $\mathcal{A}(G)$. Every other $X \in \mathcal{X}$ is the overall intersection of a set $C_1, \ldots, C_k$ of at least two maximal cliques. This means that $\mathcal{X}$ contains in particular all separators of G.

To illustrate how the nodes of a clique arrangement are related, we list the following two observations:

Observation 1. *If $X \in \mathcal{X}$ is a node in the clique arrangement $\mathcal{A}(G) = (\mathcal{X}, \mathcal{E})$ of a chordal graph G and if $Y_1, \ldots, Y_\ell \in \mathcal{X}$ are the direct successors of X, that is, the nodes such that $XY_1, \ldots XY_\ell \in \mathcal{E}$, then $X = Y_1 \cap \ldots \cap Y_\ell$. Moreover, if $C_1, \ldots, C_k$ are the sinks of $\mathcal{A}(G)$ that are reached from X by directed paths, then $X = C_1 \cap \ldots \cap C_k$.*

Observation 2. *If $Y_1, \ldots, Y_k \in \mathcal{X}$ are nodes in the clique arrangement $\mathcal{A}(G) = (\mathcal{X}, \mathcal{E})$ of a chordal graph G such that their intersection $X = Y_1 \cap \ldots \cap Y_k$ is not empty, then $X \in \mathcal{X}$.*

As stated above, many nodes in a clique arrangement may be neither maximal cliques nor separators. Actually, the existence of these nodes makes up the difference to the clique-separator graph by Ibarra [9]. See for example the graph and its clique arrangement in Figure 3. It contains $\{7, 8\}$, which is the intersection of $\{4, 5, 7, 8\}$ and $\{0, 2, 3, 7, 8\}$, but it is not a separator.

Taking all possible maximal clique intersections into $\mathcal{X}$ also has a remarkable drawback. In fact, the cardinality $|\mathcal{X}|$ can be exponentially high for general chordal graphs and even for the subclass of split graphs. For example, consider the split graph build on the clique $\{x_1, \ldots, x_k\}$ and the independent set $\{y_1, \ldots, y_k\}$, where $x_i y_j$ is an edge, if and only if $i \neq j$ for all $i, j \in \{1, \ldots, k\}$. The corresponding clique arrangement contains, amongst some others, one node for every subset of $\{x_1, \ldots, x_k\}$, hence more than 2^k nodes.

This makes clique arrangements a little unwieldy to represent chordal graphs in general. For that reason we turn our attention to strongly chordal graphs in the next sections and show for this important and well studied subclass of chordal graphs that the corresponding clique arrangements have a nice structural characterization and remain of linear size.

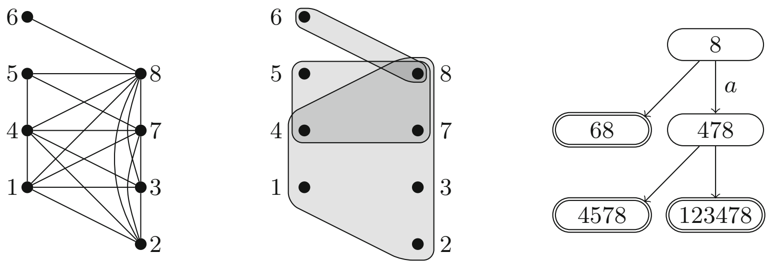


Fig. 2. A strongly chordal graph G (left) with strong elimination ordering 1, 2, 3, 4, 5, 6, 7, 8, a schematic diagram of its maximal cliques (center), and its clique arrangement (right). Maximal cliques are double framed and separators are single framed.

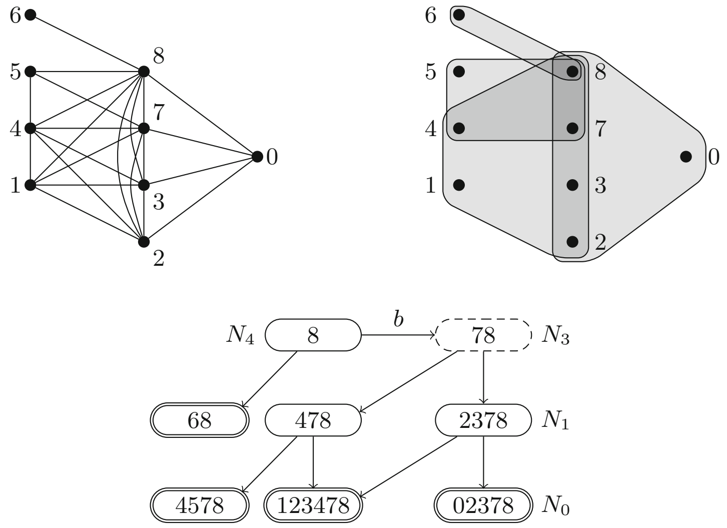


Fig. 3. The strongly chordal graph of Figure 2 extended by the simple vertex 0 (left) with strong elimination ordering $0, 1, 2, \ldots, 8$, the according schematic diagram of its maximal cliques (right), and the modified clique arrangement (below). Compared to Figure 2, the clique arrangement contains three new nodes that result from intersections of the new maximal clique N_0 and the arc a is replaced by the arc b. The dashed framed node is neither a maximal clique nor a separator.

4 Clique Arrangements for Strongly Chordal Graphs

In contrast to general chordal graphs, the clique arrangements of strongly chordal graphs have only polynomial size, which is shown by the following lemma:

Lemma 3. *If G is a strongly chordal graph with clique arrangement $\mathcal{A}(G) = (\mathcal{X}, \mathcal{E})$ and $X \in \mathcal{X}$, then there are $C_1, C_2 \in \mathcal{C}(G)$ such that $X = C_1 \cap C_2$.*

By Lemma 3, the cardinality of $\mathcal{X}$ is at most $|\mathcal{C}(G)|^2$. As chordal graphs have at most $n-1$ maximal cliques [6], it follows that $|\mathcal{X}| < n^2$. On the other hand, $O(n^2)$

is the worst case lower bound for $|\mathcal{X}|$, which holds even for the very restricted subclass of unit interval graphs. For example, the clique arrangement of the unit interval graph with vertices $x_1, \ldots, x_{3k-1}$ and cliques $C_i = \{x_i, \ldots, x_{i+k-1}\}, i \in \{1, \ldots, 2k\}$ contains the node $\{x_i, \ldots, x_j\}$ for all $i, j : k \leq i \leq j < 2k$, and thus, $\Omega(k^2) = \Omega(n^2)$ nodes.

By the upper size bound, clique arrangements might be useful in yet to develop algorithms coping with strongly chordal graphs. In fact, it would be nice, if this new representation helped to answer the open question whether strongly chordal graphs can be recognized in linear time. To support the ongoing analysis of this and other complex problems, we provide a characterization for clique arrangements of strongly chordal graphs in this section and analyze the complexity of constructing these clique arrangements in the following section.

The characterization of strongly chordal graphs is based on the following definition of cycles in the clique arrangement: For any $k \in \mathbb{N}$, a *k-cycle* of $\mathcal{A}(G)$ is a set of nodes $S_0, \ldots, S_{k-1}, T_0, \ldots, T_{k-1}$ such that for all $i \in \{0, \ldots, k-1\}$ there is a directed path from S_i to T_i and a directed path from S_i to T_{i-1} (counted modulo k). The nodes $S_0, \ldots, S_{k-1}$ are called *starters* of the cycle and the nodes $T_0, \ldots, T_{k-1}$ are called *terminals* of the cycle. Note that, by definition, $S_i \subseteq T_i$ and $S_i \subseteq T_{i-1}$ for all $i \in \{0, \ldots, k-1\}$.

But like cycles in chordal graphs are forbidden only if they are induced, a k-cycle is not per se forbidden in the clique arrangement of a strongly chordal graph. We call a k-cycle *bad* , if $k \geq 3$ and for all $i, j \in \{0, \ldots, k-1\}$ there is a directed path from S_i to T_j only if $j \in \{i-1, i\}$ (counted modulo k). In the following theorem we show that the clique arrangements of strongly chordal graphs are characterized by forbidden bad k-cycles with $k \geq 3$:

Theorem 1 *Let $G = (V, E)$ be a chordal graph and $\mathcal{A}(G) = (\mathcal{X}, \mathcal{E})$ be the clique arrangement of G. Then G is strongly chordal, if and only if $\mathcal{A}(G)$ is free of bad k-cycles for all $k \geq 3$.*

5 Efficient Construction of Clique Arrangements for Strongly Chordal Graphs

In this section, we show how to construct a clique arrangement of a strongly chordal graph in time $O(m \log n)$. In fact, the construction works in linear time on a given strong elimination ordering. The $O(m \log n)$ time bound results from Paige and Tarjan's strong elimination ordering construction algorithm [13].

By a strong elimination ordering $x_1, \ldots, x_n$ of the input graph G, the algorithm constructs $\mathcal{A}(G)$ step by step. As the clique arrangement of a graph containing a single vertex is fairly simple, the initialization easily constructs $\mathcal{A}(G[\{x_n\}])$. In every step, the algorithm modifies the current clique arrangement $\mathcal{A}(G[\{x_{i+1}, \ldots, x_n\}])$, to obtain the next clique arrangement $\mathcal{A}(G[\{x_i, \ldots, x_n\}])$. For that purpose, we have to analyze what happens to the clique arrangement, when a simple vertex x_i is added to the graph. If the neighborhood of x_i is empty or a maximal clique, things are not very complicated. In the first case, x_i forms a

Table 1. The explicit procedure

```
PROCEDURE: AddSimpleVertex
Input: A strongly chordal graph $G = (V, E)$ with simple vertex $x$, the neighborhood ordering $y_1, \ldots, y_d$ of $x$, the clique arrangement $\mathcal{A}(G - x) = (\mathcal{X}, \mathcal{E})$ and the set $\mathcal{S}$ of separators of $G - x$.
Output: The clique arrangement $\mathcal{A}(G) = (\mathcal{X}_x, \mathcal{E}_x)$ and the set $\mathcal{S}_x$ of separators of $G$.

1. Let $\mathcal{X}_x := \mathcal{X}$ and $\mathcal{E}_x := \mathcal{E}$.
2. If $N(x)$ is a maximal clique of $G - x$, replace every occurrence of $N(x)$ in $\mathcal{X}_x$ and
   $\mathcal{E}_x$ by $N[x]$. STOP.
3. Let $N_0 = N[x]$ and add the new element $N_0$ to $\mathcal{X}_x$.
4. Let $p := 0$ and $i := 1$.
5. While $\{y_i, \ldots, y_d\} \notin \mathcal{X}$ and $i \le d$, do:
   (a) Find $T_i \in \mathcal{X}$ such that $y_i \in T_i$ but there is no $X \in \mathcal{X}$ with $y_i \in X$ and $X \subset T_i$.
   (b) If $i > 1$ and $y_{i-1} \in T_i$, continue with the next iteration of the loop.
   (c) Add the new element $N_i = \{y_i, \ldots, y_d\}$ to $\mathcal{X}_x$.
   (d) Add the arcs $N_iT_i$ and $N_iN_p$ to $\mathcal{E}_x$.
   (e) Set $p := i$ and $i := i + 1$.
6. If $i \le d$, find $T_i$ as in Step 5a and replace arc $T_iT_p \in \mathcal{E}_x$ by $T_iN_p$.
7. If $i > 1$, set $\mathcal{S}_x := \mathcal{S} \cup \{N_1\}$ and otherwise, set $\mathcal{S}_x := \mathcal{S}$.
8. STOP.
```

connected component of $G[\{x_i, \ldots, x_n\}]$ and hence, the new clique arrangement is simply provided by an additional isolated node containing exactly x_i. In the second case, $N(x_i)$ is a sink of $\mathcal{A}(G[\{x_{i+1}, \ldots, x_n\}])$, which has to be extended with x_i in the new clique arrangement. Since this node is the only maximal clique that contains x_i, all other nodes and their relations remain unchanged.

The interesting case occurs when $N(x_i)$ is neither empty nor a maximal clique. Then $G[\{x_i, \ldots, x_n\}]$ contains a new maximal clique, $N[x_i]$, and hence the clique arrangement needs a new sink for $N[x_i]$. Furthermore, additional nodes may be introduced in $\mathcal{A}(G[\{x_i, \ldots, x_n\}])$ due to intersections of $N[x_i]$ and other maximal cliques. To characterize these new nodes and their relations in $\mathcal{A}(G[\{x_i, \ldots, x_n\}])$, the algorithm takes advantage of x_i being a simple vertex. In particular, there is an ordering $y_1, \ldots, y_d$ of the neighbors of x_i such that $N[y_j] \subseteq N[y_k]$ if $j < k$. In fact, it turns out that every node added to the clique arrangement has the form $\{y_j, \ldots, y_d\}$ for some $j \in \{1, \ldots, d\}$. Hence, all these new nodes are ordered in terms of set inclusion and we can show that they form a directed path in $\mathcal{A}(G[\{x_i, \ldots, x_n\}])$ that ends in $N[x_i]$.

Before we are able to describe the entire algorithm, we provide the procedure `AddSimpleVertex` in Table 1, which inserts the mentioned directed path of new nodes into $\mathcal{A}(G[\{x_{i+1}, \ldots, x_n\}])$. For the input of this procedure we need, beside a strongly chordal graph G with simple vertex x, also the following ordering for the neighborhood of x. The *neighborhood ordering* of a simple vertex x is an ordering $y_1, \ldots, y_d$ of $N(x)$ such that $N[y_i] \subseteq N[y_j]$, if and only if $i < j$ for all $i, j \in \{1, \ldots, d\}$.

The procedure `AddSimpleVertex` handles the first case, $N(x_i) = \emptyset$, in Step 3 and by terminating without entering the loop in Step 5. The second case, where

$N(x_i)$ is a maximal clique, is handled in Step 2. The loop in Step 5 deals with the last case. Here, we test for every possible new node $N_j = \{y_j, \dots, y_d\}$, whether it is the intersection of $N[x_i]$ and another maximal clique of $G[\{x_i, \dots, x_n\}]$. Only in that case we add N_j and the appropriate arcs to the clique arrangement.

The following lemma states the correctness of the procedure for all three cases:

Lemma 4. *Procedure* `AddSimpleVertex` *is correct.*

To illustrate the principles of procedure `AddSimpleVertex`, we refer to Figure 2 and Figure 3, which show the clique arrangement of a strongly chordal graph before and after adding the simple vertex 0. As the neighborhood of vertex 0, that is, $N(0) = \{2, 3, 7, 8\}$, is neither empty nor a maximal clique, we are in case three. Hence, in Step 3 the procedure adds the new sink $N_0 = \{0, 2, 3, 7, 8\}$ to the clique arrangement. After this, the loop in Step 5 is entered, which step by step considers $N_1 = \{2, 3, 7, 8\}$, $N_2 = \{3, 7, 8\}$ and $N_3 = \{7, 8\}$. But only N_1 and N_3 become part of the new clique arrangement, as the procedure decides in Step 5b, that N_2 is not an intersection of N_0 and another maximal clique. The loop is left for $N_4 = \{8\}$, because N_4 is already contained in the old clique arrangement, and finally, the arc a is replaced by the arc b.

On the basis of `AddSimpleVertex`, a complete algorithm would (1) compute a strong elimination ordering $x_1, \dots, x_n$ for the input graph G including all neighborhood orderings, (2) initialize the clique arrangement $\mathcal{A}(G[\{x_n\}])$, and (3) call `AddSimpleVertex` with simple vertex x_i for every i from $n-1$ to 1. By Lemma 4 this method correctly leads to the clique arrangement $\mathcal{A}(G)$.

Notably, we find all maximal cliques and all separators of G along the way. Whereas the maximal cliques are simply the sinks of the clique arrangement, the separators are identified in Step 7 of `AddSimpleVertex`. In particular, by Theorem 3 in [11], a clique X of G is a separator, if and only if there is a vertex x_i in a perfect elimination ordering $x_1, \dots, x_n$, such that X is not a maximal clique in $G[\{x_{i+1}, \dots, x_n\}]$ and X is the intersection $N(x_i) \cap \{x_i, \dots, x_n\}$.

The problem with the algorithm described above is that we cannot expect it to run in time $O(m \log n)$. In worst case, we may have $O(n^2)$ nodes in the final clique arrangement and, as nodes have an average worst case size of $O(n)$, the construction takes at least $O(n^3)$ time. Actually, the procedure in Table 1 is even more inefficient.

But as it is most likely sufficient for algorithmic purposes to compute just the graph structure of the clique arrangement, we propose another algorithm in this section that desists from explicitly determining the node contents. In fact, Ibarra [9] constructs his clique-separator graph in just the same way, as otherwise he would not be able to achieve a running time below $O(n^2)$ for unit interval graphs, either.

Hence, the time improvement of our second algorithm is achieved by rather constructing the following abstract version of the clique arrangement: The *abstract clique arrangement* $\mathcal{A}'(G) = (\mathcal{X}', \mathcal{E}', s, t, \mathcal{S}')$ of a strongly chordal graph $G = (V, E)$ is a directed acyclic graph with functions $s : \mathcal{X}' \to \mathbb{N}$ and $t : V \to \mathcal{X}'$ and a set $\mathcal{S}' \subseteq \mathcal{X}'$. It is related to $\mathcal{A}(G) = (\mathcal{X}, \mathcal{E})$ in the following way: There

Table 2. The fast procedure

```
PROCEDURE: AddSimpleVertexAbstract
Input: A strongly chordal graph $G = (V, E)$ with simple vertex $x$ and its neighborhood ordering $y_1, \dots, y_d$ and the abstract clique arrangement $\mathcal{A}'(G - x) = (\mathcal{X}', \mathcal{E}', s, t, \mathcal{S}')$.
Output: The abstract clique arrangement $\mathcal{A}'(G) = (\mathcal{X}'_x, \mathcal{E}'_x, s_x, t_x, \mathcal{S}'_x)$.

1. Let $\mathcal{X}'_x := \mathcal{X}'$ and $\mathcal{E}'_x := \mathcal{E}'$.
2. Let $s_x(X) := s(X)$ and $t_x(z) := t(z)$ for all $X \in \mathcal{X}'$ and $z \in V \setminus \{x\}$.
3. If $d > 0$ and $t(y_1)$ is a sink in $\mathcal{A}'(G - x)$ with $s(t(y_1)) = d$, set $s_x(t(y_1)) := d + 1$ and $t_x(x) := t(y_1)$, STOP.
4. Add a new element $N'_0$ to $\mathcal{X}'_x$ and set $s_x(N'_0) := d + 1$ and $t_x(x) := N'_0$.
5. Let $p := 0$ and $i := 1$.
6. While $s(t(y_i)) \neq d - i + 1$ and $i \leq d$, do:
   (a) If $i > 1$ and $t(y_i) = t(y_{i-1})$, continue with the next iteration of the loop.
   (b) Add a new element $N'_i$ to $\mathcal{X}'_x$ and set $s_x(N'_i) := d - i + 1$ and $t_x(y_i) := N'_i$.
   (c) Add the edges $N'_i t(y_i)$ and $N'_i N'_p$ to $\mathcal{E}'_x$.
   (d) Set $p := i$ and $i := i + 1$.
7. If $i \leq d$, replace edge $(t(y_i), t(y_p)) \in \mathcal{E}'_x$ by $(t(y_i), t_x(y_p))$.
8. If $i > 1$, set $\mathcal{S}'_x := \mathcal{S}' \cup \{N'_1\}$ and otherwise, set $\mathcal{S}'_x := \mathcal{S}'$.
9. STOP.
```

is a bijective function $b : \mathcal{X}' \to \mathcal{X}$ such that for all $X, Y \in \mathcal{X}'$ we have $XY \in \mathcal{E}' \Leftrightarrow b(X)b(Y) \in \mathcal{E}$ and for all $X \in \mathcal{X}'$:

1. $s(X) = |b(X)|$,
2. $t(x) = X$, if and only if $x \in b(X)$ and there is no $Y \in \mathcal{X}'$ with $|b(Y)| < |b(X)|$ and $x \in b(Y)$, and
3. $X \in \mathcal{S}' \Leftrightarrow b(X) \in \mathcal{S}(G)$.

Notice that the value of $t(x)$ is well defined, as the existence of two nodes $X, Y \in \mathcal{X}'$ with $x \in b(X)$, $x \in b(Y)$, and $|b(X)| = |b(Y)|$ implies another node $Z \in \mathcal{X}'$ with $x \in b(Z) = b(X) \cap b(Y)$ and $|b(Z)| < |b(X)|$.

The idea for the faster algorithm is to keep the described framework, but to replace `AddSimpleVertex` by the procedure `AddSimpleVertexAbstract`, given in Table 2, which simulates the step by step construction process to build the abstract clique arrangement. In fact, we firstly introduced `AddSimpleVertex` just to clarify the approach, because the function of the new procedure is technical and hard to follow.

The ability of `AddSimpleVertexAbstract` to work without explicitly knowing the contents of the nodes, arises from the possibility to infer these contents just from the information stored in the functions s and t. The correctness and the running time of `AddSimpleVertexAbstract` is stated by the following lemma:

Lemma 5. *Procedure* `AddSimpleVertexAbstract` *runs in time* $O(deg(x) + 1)$ *and is correct.*

We would like to conclude with a little remark on the relation between abstract clique arrangements and clique-separator graphs. Since we can identify the nodes in $\mathcal{A}'(G)$ that represent the maximal cliques and the separators of G, it is possible to derive the clique-separator graph from $G'(G)$ in linear time:

Corollary 2. *The clique-separator graph of a strongly chordal graph G can be computed in time $O(m \log n)$.*

6 Conclusions and Future Work

This paper characterizes clique arrangements for strongly chordal graphs in terms of forbidden bad cycles and gives a respective $O(m \log n)$ time algorithm to construct a clique arrangement for any given strongly chordal graph. We leave it as an open question, whether the presence of bad cycles can be tested in linear time. In that case, it would be exciting to think about possibilities of constructing clique arrangements for strongly chordal graphs in linear time, because this would accordingly imply linear time recognition of strongly chordal graphs – a long standing open problem.

Moreover, as clique arrangements for strongly chordal graphs are characterized by forbidden bad cycles, it would be nice to know if we could find similar forbidden structures for subclasses, such as rooted directed path graphs or interval graphs. In case of ptolemaic graphs, it is already known that the clique arrangement is free of any kind of cycles.

Conversely, one could ask if there are natural subclasses of strongly chordal graphs defined by banning other cycles from the clique arrangement. In fact, we would like to ask for a graph characterization when additionally 2-cycles are forbidden.

References

1. Bernstein, P.A., Goodman, N.: Power of natural semijoins. SIAM Journal on Computing 10, 751–771 (1981)
2. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. SIAM Monographs on Discrete Math. Appl. 3 (1999)
3. Dirac, G.: On rigid circuit graphs. Abh. Math. Sem. Univ. Hamburg 25, 71–76 (1961)
4. Fagin, R.: Degrees of Acyclicity for Hypergraphs and Relational Database Schemes. Journal ACM 30, 514–550 (1983)
5. Farber, M.: Characterizations of strongly chordal graphs. Discrete Mathematics 43(2-3), 173–189 (1983)
6. Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. Pacific Journal of Mathematics 15, 835–855 (1965)
7. Galinier, P., Habib, M., Paul, C.: Chordal graphs and their clique graphs. In: Nagl, M. (ed.) WG 1995. LNCS, vol. 1017, pp. 358–371. Springer, Heidelberg (1995)
8. Ho, C., Lee, R.C.T.: Counting clique trees and computing perfect elimination schemes in parallel. Information Processing Letters 31, 61–68 (1989)
9. Ibarra, L.: The clique-separator graph for chordal graphs. Discrete Applied Mathematics 157(8), 1737–1749 (2009)

10. McKee, T.A.: How chordal graphs work. Bulletin of the ICA 9, 27–39 (1993)
11. Rose, D.J.: Triangulated graphs and the elimination process. Journal of Mathematical Analysis and Applications 32, 597–609 (1970)
12. Shibata, Y.: On the tree representation of chordal graphs. Journal on Graph Theory 12(3), 421–428 (1988)
13. Paige, R., Tarjan, R.E.: Three partition refinement algorithm. SIAM Journal on Computing 16, 973–989 (1987)
14. Uehara, R., Uno, Y.: Laminar structure of ptolemaic graphs and its applications. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 186–195. Springer, Heidelberg (2005)

Beyond Knights and Knaves

Christine T. Cheng[1], Andrew McConvey[2,⋆], Drew Onderko[1,⋆], Nathaniel Shar[3,⋆], and Charles Tomlinson[4]

[1] Department of Computer Science, University of Wisconsin-Milwaukee, Milwaukee, WI 53211, USA
{ccheng,donderko}@uwm.edu

[2] Department of Mathematics, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
mcconve2@illinois.edu

[3] Department of Mathematics, Rutgers University, Piscataway, NJ 08854, USA
nshar@math.rutgers.edu

[4] Department of Mathematics, University of Nebraska-Lincoln, Lincoln, NE 68588, USA
ctomlinson2@math.unl.edu

Abstract. In the classic *knights and knaves problem*, there are n people in a room each of whom is a *knight* or a *knave*. Knights always tell the truth while knaves always lie. Everyone in the room knows each other's identity. You are allowed to ask questions of the form "Person i, is person j a knight?" and you are told that there are more knights than knaves. What is the fewest number of questions you can ask to determine a knight? How about to determine everyone's identity?

In this paper, we consider the knights and no-men problem, where a no-man is a person who always answers "no". Assuming there are at least k knights, we show that $\binom{n-1}{2} - \lfloor \frac{(k-2)(n-1)^2}{2(k-1)} \rfloor$ questions are necessary and sufficient in the worst case to identify a knight. We also show that $n-2$ questions suffice to identify a no-man, and $\binom{n-1}{2} - \lfloor \frac{(k-2)(n-1)^2}{2(k-1)} \rfloor + n - 2$ questions suffice to identify everyone in the room.

We then consider a generalization of the knights and knaves problem that captures most of the variants of the knights and knaves problem in the literature. In the *agent labeling problem*, we wish to identify everyone's type; in the *agent identification problem*, we wish to identify an agent having a particular type. We present results with regards to the fewest number of questions needed in the worst case to solve both the agent labeling and agent identification problems. Our tools and results are graph theoretic in nature.

1 Introduction

In the classic *knights and knaves problem* there are n people in a room. Each person is either a *knight* or a *knave*. Knights always tell the truth while knaves

⋆ Supported in part by the National Science Foundation grant DMS 08-38434 "EMSW21-MCTP: Research Experience for Graduate Students".

always lie. Everyone in the room knows each other's identity. You are allowed to ask questions of the form "Person i, is person j a knight?" and you are told that there are more knights than knaves. What is the fewest number of questions you can ask to determine a knight? How about to determine everyone's identity? Popular variations to this problem introduce new "types" of people such as *spies*, who lie or tell the truth arbitrarily, and *yes-men*, who answer "yes" all the time.

Several equivalent formulations of the knights and knaves problem have been studied by other authors including the problem of classifying computer chips as faulty or reliable [1], the problem of identifying a coin of the majority weight by performing weight comparisons [2], the problem of identifying a ball of the most common color by performing color comparisons [3,4,5], etc.

Saks and Werman [5] studied a version of the knights and knaves problem where knights are in the majority if such a majority exists. Their goal was to identify a knight or declare the nonexistence of such a majority. They showed that $n - B(n)$ questions are necessary and sufficient in the worst case, where $B(n)$ is the number of ones in the binary representation of n. Simpler proofs were given later by Alonso et al. [4] and Wiener [2]. If the number of knights is at least k, and $k > n/2$, Aigner [3] showed that $n-1-p$ questions are necessary in the worst case, where p is the highest power of two dividing $\binom{n-1}{k-1}$, and that $2(n-k) - B(n-k)$ questions are necessary and sufficient in the worst case when it is known that a majority exists.

When we want to identify everyone as either a knight or a knave, Aigner [3] observed that $n-1$ questions are necessary in the worst case when $n > 2$. Aigner also showed that if the number of knights is at least k, $k > n/2$, $\lceil \frac{(n-k)n+(n-k)}{n-k+1} \rceil$ questions are necessary in the worst case, up to a possible error of 1.

The *knights and spies* problem is just like the knights and knaves problem except that agents which are not knights are spies. Alonso et al. [1] showed that, surprisingly, any strategy that can identify a knight in the knights and knaves setting can also be used to identify a knight in the knights and spies setting. Therefore, the two problems require the same number of questions to identify a knight in the worst case. Aigner [3] also proved that the same result is true when the number of knights is bounded below by k, $k > n/2$.

If we wish to identify everyone in a room of knights and spies and there are $k > n/2$ knights, Blecher [6] and Wildon [7] showed independently that $2n-k-1$ questions are necessary and sufficient in the worst case. Thus, it follows that $3n/2 - 1$ questions are necessary and sufficient in the worst case if it is only known that knights are in the majority.

Most recently, Hanajiri [8] introduced the concept of *yes-men* and studied the *knights and yes-men* problem. Suppose there are n people in the room, at most p of whom are knights and at most q of whom are yes-men. Hanajiri showed that $n + \lfloor log_2(p+q-n) \rfloor$ questions suffice for identifying everyone in the room. Additionally, he conjectured that the same number of questions are necessary in the worst case when $p, q \notin \{0, n\}$. He was able to verify his conjecture when $p + q \in \{n, n+1, 2n-1\}$.

Our Results. Inspired by Hanajiri's work, we introduce the notion of a *no-man*, a person who always answers "no". Assume that there are n people each of whom is a knight or a no-man, and that there are at least k knights. We show that $\binom{n-1}{2} - \lfloor \frac{(k-2)(n-1)^2}{2(k-1)} \rfloor$ questions are necessary and sufficient in the worst case to identify a knight. We also show that $n-2$ questions suffice to identify a no-man, and $\binom{n-1}{2} - \lfloor \frac{(k-2)(n-1)^2}{2(k-1)} \rfloor + n - 2$ questions suffice to identify everyone.

Next, we present a generalization of the knights and knaves problem that allows us to consider new types of people and/or put multiple types (not just two types) of people in a room. An instance consists of n people, the m types that each person can have, and a 0-1 matrix Q of order m that describes how the m types of people interact with each other. We use D_Q to denote the directed graph on $[m]$ whose adjacency matrix is Q. In the *agent labeling problem*, we wish to determine the type of every individual in the room; in the *agent identification problem*, we wish to find an agent having a particular type.

It turns out that even when we have asked the agents all the possible questions we can ask, we may not be able to solve the agent labeling or agent identification problems. When this happens, we call the instance *ill-formed*; otherwise, it is *well-formed*. We provide a characterization of the well-formed instances of both the agent labeling and agent identification problems in terms of D_Q.

For the agent labeling problem, we prove that when a well-formed instance's D_Q has what we call an *uninformative partition*, identifying everyone's type requires $\Omega(n^2/m^4)$ questions in the worst case. On the other hand, when the instance's D_Q has no uninformative partitions, everyone's type can be determined using $O(mn)$ questions. This implies that when $m = o(n^{1/5})$ there are hard and easy variants of the knights and knaves problem. In particular, when m is a constant, the hard variants need $\Omega(n^2)$ questions to solve the agent labeling problem in the worst case. The easy variants, however, can solve the agent labeling problem using $O(n)$ questions.

For the agent identification problem, we prove that when a well-formed instance's D_Q has uninformative partitions and s^* is an *informative type* with respect to one of the uninformative partitions, then finding an agent of type s^* requires $\Omega(n^2/m^4)$ questions in the worst case. On the other hand, when the instance's D_Q has no uninformative partitions, finding an agent of any type can be solved using $O(mn)$ questions. Thus, setting aside the case when s^* is an uninformative type with respect to every uninformative partition of D_Q, our results again imply that when $m = o(n^{1/5})$, there are easy and hard variants of the knights and knaves problem when solving the agent identification problem.

We emphasize that although our problems do not inherently involve graphs, our tools and results are graph theoretic in nature. A full version of our paper can be found at `http://www.cs.uwm.edu/~ccheng/`.

2 The Knights and No-Men Problem

Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of *agents* each of whom is a knight or a no-man. We are allowed to ask questions of the form "a_i, is a_j a knight?" for

$i \neq j$. A knight always answers truthfully while a no-man always answers "no". Additionally, we are told that there are at least $k \geq 2$ knights. Let us consider the problem of finding a knight.

Suppose we asked the agents a set of questions S. Let G_S be the undirected graph where $V(G_S) = A$ and $E(G_S) = \{\{a_i, a_j\}$: "a_i, is a_j a knight?" or "a_j, is a_i a knight?" is in $S\}$. We shall say that a labeling of the agents $f : A \rightarrow \{\text{knight, no-man}\}$ is *consistent with respect to* S if based on the agents' answers it is plausible that each a_i has type $f(a_i)$. More specifically, f *is a consistent labeling with respect to* S if for each question "a_i, is a_j a knight?" in S, the answer given by a_i is the same as the answer an agent of type $f(a_i)$ would give if asked about an agent of type $f(a_j)$.

Lemma 1. *After asking the questions in S, we can conclude that some particular agent is a knight if and only if at least one of these conditions hold:*
(1) some question's answer is "yes", or
(2) the answers to all the questions are "no" and there is some a_i that is in every independent set of size k in G_S.

Proof. First, we argue that the conditions are sufficient. We get a response of "yes" if and only if one knight is asked about another knight. Thus, if condition (1) holds we can identify two agents as knights. So suppose the answers to the questions in S are all "no". It must be the case that no knight has been asked about another knight. The set of knights form an independent set of size at least k in G_S. If condition (2) holds, some a_i is part of *every* independent set of size k in G_S, including the independent sets containing k knights. We can then conclude that a_i is a knight.

Next, let us show that the conditions are necessary. Suppose neither condition holds. Since condition (1) does not hold, the responses to the questions in S are all "no". Once again the set of knights form an independent set I of size at least k in G_S. Since condition (2) does not hold, for every $a_i \in A$, there exists an independent set I_{a_i} in G_S of size k such that $a_i \notin I_{a_i}$. Now, for each $a_i \in A$, let f_i be a labeling of the agents that assigns the agents in I_{a_i} as knights and all other agents as no-men. Clearly, f_i is consistent with respect to S. Since it is possible that each a_i is not a knight, we cannot conclude that any particular agent in A is a knight. □

The Turán graph $T(n, k)$ is the graph whose n vertices are partitioned into k parts of as equal size as possible, and two vertices are adjacent if and only if they belong to different parts. Turán's theorem [9] states that among n-vertex graphs without a clique of size $k + 1$, the graph with the most number of edges is $T(n, k)$. In particular, $T(n, k)$ has $\lfloor \frac{(k-1)n^2}{2k} \rfloor$ edges.

Lemma 2. *Let $\mathcal{G}(n, k)$ contain all graphs G on n vertices such that G has a vertex that is part of every independent set of size k. Let $G^* \in \mathcal{G}(n, k)$ so that among all the graphs in $\mathcal{G}(n, k)$ it has the fewest number of edges. Then G^* is the complement of $T(n - 1, k - 1)$ unioned with an isolated vertex.*

Proof. Let v be a vertex of G^* that is part of every independent set of size k. Notice that $deg(v) = 0$. Otherwise, any edge incident to v can be removed and v will still part of every independent set of size k, contradicting our assumption about the minimality of $|E(G^*)|$. Next, consider $G^* - v$. By our assumptions about G^* and v, it must be the case that $G^* - v$ has no independent sets of size k, and among such graphs with $n-1$ vertices, it has the fewest number of edges. Consequently, its complement has the property that it has no cliques of size k, and among such graphs with $n-1$ vertices, it has the most number of edges. According to Turán's theorem [9], this complement is $T(n-1, k-1)$. □

Theorem 1. *Let $I(n,k)$ be the fewest number of questions needed in the worst case to identify a knight in the knights and no-men problem when there are n agents at least $k \geq 2$ of whom are knights. Then $I(n,k) = \binom{n-1}{2} - \lfloor \frac{(k-2)(n-1)^2}{2(k-1)} \rfloor$, the number of edges in the complement of $T(n-1, k-1)$.*

Proof. Let $t'(n-1, k-1)$ denote the number of edges in the complement of $T(n-1, k-1)$. First, we describe a strategy for finding a knight. Start by setting aside an agent. Partition the remaining $n-1$ agents into $k-1$ parts of as equal size as possible. For each part, ask every pair $\{a_i, a_j\}$ of agents the question "a_i, is a_j a knight?". (The order of a_i and a_j doesn't matter here.) If the response is "yes", stop and conclude that that a_i and a_j are knights. If none of the agents answered "yes", conclude that (1) there are exactly k knights, (2) each part has exactly one of them, and (3) the agent we had set aside is a knight. Using the fact that there are at least k knights and at most $k-1$ parts, it is easy to verify that our strategy's conclusions are correct. It also asks the most number of questions when the answers to all its questions are "no". Now, we have chosen the questions so that they form a graph that is isomorphic to G^* so the number of questions our strategy will ask in the worst case is $t'(n-1, k-1)$; i.e., $I(n,k) \leq t'(n-1, k-1)$.

Next, consider an arbitrary strategy. Suppose there are situations where the strategy will conclude that some agent is a knight even though all the answers to its questions are "no". Let G be the graph formed by these questions. According to Lemma 1, G must have the property described in condition (2). Thus, $G \in \mathcal{G}(n,k)$. According to Lemma 2, G must have at least $t'(n-1, k-1)$ edges. Now, suppose the strategy will conclude that some agent is a knight *only* when it receives a "yes" response. Assume that in the worst case it will ask r questions. This means that in every situation where the strategy has already asked $r-1$ questions and the answers were all "no", the rth question *must* generate a "yes" answer and the two agents that are part of the question are knights. Thus, if the strategy knows about the worst case bound r, it can avoid asking the rth question since after receiving the "no" answer to the $(r-1)$st question, it can already identify an agent that is a knight.

Applying the reasoning we used in the previous case, we have that $r - 1 \geq t'(n-1, k-1)$ so $r > t'(n-1, k-1)$. We have shown that every strategy for finding a knight where there are n agents and at least $k \geq 2$ knights will need to ask at least $t'(n-1, k-1)$ questions. □

Theorem 2. *Let $\hat{I}(n,k)$ be the fewest number of questions needed in the worst case to identify a no-man in the knights and no-men problem when there are n agents, at least $k \geq 2$ knights, and at least 2 no-men. Then $\hat{I}(n,k) \leq n-2$.*

Theorem 3. *Let $L(n,k)$ be the fewest number of questions needed in the worst case to identify all agents as a knight or no-man when there are n agents at least $k \geq 2$ of whom are knights. Then $I(n,k) \leq L(n,k) \leq I(n,k)+n-2$.*

Suppose we have a room with n people and all we know is that there are at least two knights and two non-knights. Under these conditions Hanajiri's [8] result implies that $O(n)$ questions suffice to identify everyone in a room of knights and yes-men, while Theorem 3 states that $\Omega(n^2)$ questions may be needed in a room of knights and no-men. A similar contrast exists when we want to find a knight or a no-man in a room of knights and no-men: Theorem 2 states that $O(n)$ questions suffice to find a no-man while Theorem 1 states that $\Omega(n^2)$ questions may be necessary to find a knight. Our results in the next section explain why these situations contrast as they do.

3 Generalizing the Knights and Knaves Problem

We now consider a generalization of the knights and knaves problem that encompasses most of the variants we discussed in the introduction. It has the following set-up: There are n agents in a room each of whom is one of m types. The agents know each other's types. Outsiders are allowed to communicate with the agents by asking only one kind of question. This question is "directed" – it is addressed to some agent a_i and is about another agent a_j – and has a yes or no answer. Agent a_i's response is a deterministic function of his and a_j's types. You are told how agents respond to the directed questions as a function of their types. Your goal is to determine the types of all the agents or identify an agent of a particular type using as few questions as possible.

Notice that in our set-up, the actual question itself is not a factor since it is the only question an outsider can ask. What matters instead is the "direction" of the question and the response to it. Assume the types are from the set $[m] = \{1, 2, \ldots, m\}$. We shall use a $(0,1)$-matrix Q of order m to encode responses where Q_{st} is equal to 0 if the answer to the question addressed to an agent of type s about an agent of type t is "no" and is equal to 1 otherwise. Throughout the paper, we shall represent the structure of agent types in terms of $D_Q = ([m], Q)$, the directed graph whose vertex set is $[m]$ and whose adjacency matrix is Q.

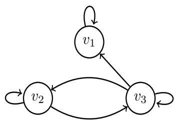
Formally, our model has a set of n agents $A = \{a_1, a_2, \ldots, a_n\}$, where $t(a_i)$ denotes the type of agent a_i. Every $t(a_i) \in [m]$. As an outsider, we do not know the agents' types. However, we are allowed to ask one agent about another agent using some standard format. Let $q(a_i, a_j)$ denote the answer to the question addressed to a_i about a_j, where $a_i \neq a_j$. This answer, given by a_i, is $Q_{t(a_i),t(a_j)}$.

We are interested in the following problems, which we would like to solve using as few questions as possible. In the *agent labeling problem* we are given A and Q, and our goal is to determine the types of all the agents in A. In the *agent* *identification problem* we are given A, Q, and $s^* \in [m]$, and our goal is to find an agent in A of type s^*.

As an example, consider an instance where the agents can have three types: knights, knaves, or yes-men. The table on the left shows how the three types of agents respond to the question "Agent a_i, is agent a_j a knight?", the matrix in the middle encodes these responses, while the directed graph on the right has the matrix as its adjacency matrix.

	Knight	Knave	Yes-man
(v_1) Knight	Yes	No	No
(v_2) Knave	No	Yes	Yes
(v_3) Yes-man	Yes	Yes	Yes

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$



Assumptions. Finally, we make the following assumptions about our model: First, for each $s \in [m]$, there are at least two agents with type s. This is a technicality we need because we cannot ask an agent about himself. For example, if there is only one agent of a certain type, we will not be able to tell if this agent's type has a loop or not in D_Q. Second, for any two types s and s', they either have a different in-neighborhood or a different out-neighborhood in D_Q. Otherwise, we have two versions of the same type, and it would be impossible for us to distinguish between these two versions. Third, we have "infinite computational power". That is, we can develop strategies and process the agents' responses using algorithms that may not be efficient. This will allow us to focus solely on analyzing the number of questions we need to ask to solve the agent labeling and agent identification problems.

3.1 Ill-Formed Instances and Uninformative Partitions

Consider the scenario where we have asked the agents all the possible questions we can ask. We would like to group them according to how they answered as well as how others answered when asked about them.

Let agent $b \neq a_i, a_j$. We say that a_i and a_j are *similar with respect to* b, denoted as $a_i \sim_b a_j$, if $q(a_i, b) = q(a_j, b)$ and $q(b, a_i) = q(b, a_j)$. Additionally, we say that a_i and a_j are *similar*, denoted as $a_i \sim a_j$, if they are similar with respect to every agent $b \neq a_i, a_j$.

Lemma 3. *For any two agents a_i and a_j, $a_i \sim a_j$ if and only if $t(a_i) = t(a_j)$. Hence $\sim$ is an equivalence relation over A and has m equivalence classes.*

Next, we extend the notion of consistent labelings described from the previous section to our general setting. Let S be a set of questions we have asked the agents. Based on their answers, we assign each agent a label from $[m]$. We say that such a labeling of the agents $f : A \to [m]$ is *consistent with respect to S* if for each question addressed to a_i about a_j in S, the answer given by a_i is the same as the answer an agent of type $f(a_i)$ would give if asked about an agent of type $f(a_j)$. When S contains all the questions we can ask the agents of A, we simply say that f is a *consistent labeling.* One such example is the labeling f^* that assigns each agent their type (i.e., $f^*(a) = t(a)$ for each $a \in A$).

When we trade questions and answers with the agents, we are in effect narrowing down the set of consistent labelings of the agents. In the agent labeling problem, our goal is to get to a point where there is only one consistent labeling left so that we are certain about each agent's type. In the agent identification problem where we want to identify an agent of type s^*, our goal is to reach a point where there is an agent a_i, so that in all of the labelings consistent with the set of questions asked so far, a_i is assigned the label s^*.

We call (A, Q) an *ill-formed instance of the agent labeling problem* if there is more than one consistent labeling of the agents. That is, the agent labeling problem cannot be solved even after we have asked all the questions. Similarly, (A, Q, s^*) is an *ill-formed instance of the agent identification problem* if there is no agent a_i so that a_i is assigned the label s^* in every consistent labeling of the agents. For both problems, we call an instance *well-formed* if it is not ill-formed.

Theorem 4. *An instance (A, Q) of the agent labeling problem is ill-formed if and only if D_Q has more than one automorphism. An instance (A, Q, s^*) of the agent identification problem is ill-formed if and only if some automorphism of D_Q does not fix s^* (i.e., the automorphism does not map s^* to itself).*

Let $E_1, E_2, \dots, E_m$ denote the equivalence classes of $\sim$. For each E_i, let b_i and b'_i be two agents in E_i. Denote by $D_\sim$ the directed graph whose vertex set is $\{E_1, E_2, \dots, E_m\}$ and whose edge set is $\{(E_i, E_j) : q(b_i, b_j) = 1\} \cup \{(E_i, E_i) : q(b_i, b'_i) = 1\}$. Since the equivalence classes are in one-to-one correspondence with the m types of agents, $D_\sim$ is in fact isomorphic to D_Q. We now describe a procedure for labeling the agents of A:

Label$(A, Q, \{q(a_i, a_j) : a_i, a_j \in A\})$

```
Partition the agents according to the equivalence classes of $\sim$.
Construct the directed graph $D_\sim$.
Find an isomorphism $\alpha$ from $D_\sim$ to $D_Q$.
For each agent $a$ in equivalence class $E$, set $f(a) = \alpha(E)$.
Return $f$.
```

Theorem 5. *The labeling f returned by* `Label` *is consistent. Consequently,*
(1) when (A, Q) is awell-formed instance of the agent labeling problem, $f = f^*$.
(2) when (A, Q, s^*) is a well-formed instance of the agent identification problem, $f(a) = s^*$ if and only if $t(a) = s^*$ for each $a \in A$.

Theorem 5 shows that when we have asked the agents all the questions we can ask and the instances are well-formed, `Label` will solve the agent labeling and agent identification problems. From here on, we will assume that the inputs to the agent labeling or agent identification problems are well-formed.

In order to address the issues raised at the end of Section 2 we now introduce the concept of *informative partitions.* Any two (not necessarily distinct) types s

and s' have four possible relationships based on the questions that can be asked between agents of type s and with those of type s': (1) $Q_{s,s'} = Q_{s',s} = 1$; (2) $Q_{s,s'} = 1$, $Q_{s',s} = 0$; (3) $Q_{s,s'} = 0$, $Q_{s',s} = 1$; (4) $Q_{s,s'} = Q_{s',s} = 0$. Keeping this in mind, let $\mathcal{P} = \{P_1, P_2, \ldots, P_r\}$ be a partition of $[m]$ with $r < m$. Type s is *uninformative with respect to* $\mathcal{P}$ if, for $i = 1, \ldots, r$, the relationship of s with each type in P_i is the same. That is, we cannot use questions addressed to or about agents of type s to distinguish between agents whose types belong to the same part in $\mathcal{P}$. Otherwise, s is *informative with respect to* $\mathcal{P}$. We call $\mathcal{P}$ an *uninformative partition* of $D_Q = ([m], Q)$ if each of its parts contains an uninformative type with respect to $\mathcal{P}$.

For example, in the knights and yes-men problem, the only possible candidate for an uninformative partition is $\mathcal{P} = \{\{\text{knight, yes-man}\}\}$ since we can only consider partitions with just one part. A knight is informative with respect to $\mathcal{P}$ because two knights have a relationship that is described by (1) whereas a knight and a yes-man have a relationship that is described by (3). Similarly, a yes-man is also informative with respect to $\mathcal{P}$ because the relationship between a yes-man and a knight is described by (2) whereas that between a yes-man and another yes-man is described by (1). Thus, the knights and yes-men problem has no uninformative partitions. On the other hand, in the knights and no-men problem, we leave it up to the reader to verify that the no-man is uninformative with respect to $\mathcal{P} = \{\{\text{knight, no-man}\}\}$ while the knight a informative with respect to the same partition. Hence, the knights and no-man problem has an uninformative partition.

Proposition 1. *Let $\mathcal{P} = \{P_1, P_2, \ldots, P_r\}$ be an uninformative partition of D_Q. Each part of $\mathcal{P}$ contains exactly one uninformative type with respect to $\mathcal{P}$.*

Let $L(n, Q)$ denote the fewest number of questions needed in the worst case to solve the agent labeling problem whose input is a well-formed instance (A, Q) with $|A| = n$. Similarly, let $I(n, Q, s^*)$ be the fewest number of questions needed in the worst case to solve the agent identification problem whose input is a well-formed instance (A, Q, s^*) with $|A| = n$. The proof of Theorem 6, omitted due to length constraints, follows a line of reasoning closely matching that used to prove Theorem 1. In particular, we have counterparts to Lemmas 1 and 2.

Theorem 6. *When $D_Q = ([m], Q)$ has an uninformative partition, $L(n, Q) \in \Omega(n^2/m^4)$. Additionally, when s^* is an informative type in some uninformative partition of D_Q, $I(n, Q, s^*) \in \Omega(n^2/m^4)$.*

3.2 LabelOrDetect

Given (A, Q) where $|A| = n$ and Q has order m, we now present an algorithm called `LabelOrDetect` that either outputs a consistent labeling of the agents in A or detects the presence of an uninformative partition in D_Q using $O(mn)$ questions. In the first part of the algorithm, a set of *special* agents are chosen. All questions addressed to the special agents or about the special agents are asked during the course of the algorithm. The answers are then used to partition a subset of the agents into parts. The goal is to get to a point where each part has exactly one special agent because the algorithm intends to make a special agent represent the agents in its part. This seems valid because if special agent b_i is in part A_i, b_i's answers are just like those of the agents in A_i. In the second part of the algorithm, the answers to all the unanswered questions are then obtained by assuming that, for each part A_i, the agents in it have the same type as b_i. The agents in A are then partitioned according to the $\sim$ relation. If the number of equivalence classes is less than what is expected – which is m – the earlier assumption must be false. The algorithm concludes that it had encountered an uninformative partition. However, if the number of equivalence classes is m, the algorithm passes the answers to all the questions to the procedure `Label`, which then outputs a labeling f.

```
AskAndPartition($A, Q$)

$A' \leftarrow A$, $\mathcal{A}' \leftarrow \{A\}$, $B \leftarrow \emptyset$
For every pair of agents $a$ and $a'$, initialize $q'(a, a')$ to $-1$.
Mark all agents in $A'$ as not special.

while (some part of $\mathcal{A}'$ has no agent marked special)
    Denote the parts in $\mathcal{A}'$ as $A'_1, A'_2, \ldots, A'_k$.
    $B_{new} \leftarrow \emptyset$
    for each part $A'_i$ in $\mathcal{A}'$
        $\mathcal{A}'_i \leftarrow \{A_{i'}\}$
        if $A_{i'}$ does not have an agent marked special
            Choose an arbitrary agent $b_i \in A_{i'}$, mark it as special, and
             add it to $B_{new}$.
            Ask all questions between $b_i$ and every other agent in $A'$ and
             record the answers in $q'$.
            if $A'_i \neq \{b_i\}$
                $\mathcal{A}'_i \leftarrow$ refine($A_{i'} - b_i, \mathcal{A}_{i'} - b_i, b_i$)
                if $\mathcal{A}'_i$ contains two or more parts
                    Move $b_i$ from $A'_i$ to $B$.
                else
                    $\mathcal{A}'_i \leftarrow \{A'_i\}$
    endfor
    for each $b_i \in B_{new}$
        for each part $A'_j$ that does not contain $b_i$
            /* Questions between $b_i$ and $A'_j$ were asked earlier. */
            $\mathcal{A}'_j \leftarrow$ refine($A'_j, \mathcal{A}'_j, b_i$)
        endfor
    endfor
    $A' \leftarrow \bigcup_{i=1}^{k} A'_i$, $\mathcal{A}' \leftarrow \bigcup_{i=1}^{k} \mathcal{A}'_i$
endwhile
return($A', \mathcal{A}', B, \{q(a, a') : a, a' \in A\}$)
```

Fig. 1. The procedure `AskAndPartition`

Consider any two agents of A. Just like any two types, these two agents have four possible relationships based on how they answer queries about each other. Let $A' \subseteq A$, let $\mathcal{A}'$ be a partition of the agents in A', and let b be an agent not in A'. In our algorithm, we use `refine(`$A', \mathcal{A}', b$`)` to denote the operation that partitions the agents in each part of $\mathcal{A}'$ according to their relationship with b.

The first part of the algorithm is done through the procedure `AskAndPartition`, shown in Figure 1. Initially, A' and $\mathcal{A}'$ are set to A and $\{A\}$ respectively, and all agents are unmarked. The values $q'(a, a')$ for each $a, a' \in A$ are set to -1 as an indication that the question addressed to a about a' has not been asked. The algorithm then enters a while loop. At the beginning of each iteration, the parts in $\mathcal{A}'$ may or may not contain special agents. For those parts A'_i that are missing a special agent, an agent $b_i \in A'_i$ is chosen, marked special and noted as new. All questions between b_i and every agent in $A' - b_i$ are asked and then stored in q'. Based on the answers, b_i is used to refine $A'_i - b_i$ and every other part of $\mathcal{A}'$ later. If b_i subdivided $A'_i - b_i$ into two or more parts, b_i is moved from A'_i to B because it is no longer obvious which part b_i should belong to; otherwise, b_i stays in A'_i.

Notice that `AskAndPartition` is *type-preserving*. That is, if two agents have the same type and neither one was moved to B, they stayed in the same part of $\mathcal{A}'$ throughout the execution of the procedure. Since there are only m types, the number of parts in $\mathcal{A}'$ never exceeds m. Additionally, from one iteration to the next in the while loop of `AskAndPartition`, the number of parts in $\mathcal{A}'$ either increases or stays the same.

Lemma 4. *The algorithm* `AskAndPartition` *terminates.*

Lemma 5. *Let A'_f, $\mathcal{A}'_f$, and B_f denote the sets A', $\mathcal{A}'$ and B at the end of the while loop in* `AskAndPartition`. *Either each part in $\mathcal{A}'_f$ consists of agents of the same type or D_Q has an uninformative partition.*

Although we omit the proof of Lemma 5 due to length constraints, we will describe how such an uninformative partition would be found. Let $A'_1, A'_2, \ldots, A'_{m'}$ be the parts in $\mathcal{A}'_f$, and let b_i be the special agent in A'_i for $i = 1, \ldots, m'$. Let $\mathcal{P} = \{P_1, \ldots, P_{m'}, \ldots, P_{m''}\}$ such that (1) for $1 \leq i \leq m'$, P_i contains the types of the agents in A'_i and (2) $P_{m'+1}, \ldots, P_{m''}$ are singleton sets containing the types not found in $\cup_{j=1}^{m'} P_i$. Then $\mathcal{P}$ is an uninformative partition and $(\cup_{j=1}^{m'} t(b_j)) \cup P_{m'+1} \cup \ldots \cup P_{m''}$ is exactly the set of labels which are uninformative with respect to $\mathcal{P}$.

Theorem 7. *The total number of questions asked in* `LabelOrDetect` *is $O(mn)$. Furthermore, if the algorithm outputs a function f, the function is a consistent labeling of A. On the other hand, when it concludes that D_Q has an uninformative partition, it really has an uninformative partition.*

According to Theorem 5, when the instances of the agent labeling and identification problems are well-formed, the consistent labeling produced by `Label` will solve the problems.

```
LabelOrDetect($A, Q$)

$(A', \mathcal{A}', B, \{q(a, a') : a, a' \in A\}) \leftarrow$ AskAndPartition($A, Q$)
for each pair of distinct agents $a, a' \in A$ such that $q'(a, a') = q'(a', a) = -1$
    if $a$ and $a'$ are not marked special
        Let $A'_i$ and $A'_j$ denote the parts of $\mathcal{A}'$ containing $a$ and
         $a'$ respectively.
        if $A'_i \neq A'_j$
            Set $q'(a, a')$ and $q'(a', a)$ to $q'(b_i, b_j)$ and $q'(b_j,\ b_i)$ respectively.
        else
            Set $q'(a, a')$ and $q'(a', a)$ to $q'(b_i, a')$ and $q'(a', b_i)$ respectively.
endfor
Based on $\{q'(a, a'), a, a' \in A\}$, partition the agents according to $\sim$.
if there are $m$ equivalence classes
    $f \leftarrow$Label($A, Q, \{q'(a, a') : a, a' \in A\}$), return($f$)
else
    return($D_Q$ has an uninformative partition)
```

Fig. 2. The algorithm LabelOrDetect

Corollary 1. *When $D_Q = ([m], Q)$ has no uninformative partitions, $L(n, Q) \in O(nm)$ and for each $s^* \in [m]$, $I(n, Q, s^*) \in O(nm)$. When $D_Q = ([m], Q)$ has only one uninformative partition, this partition is $\{[m]\}$, and s^* is the uninformative type in $\{[m]\}$, $I(n, Q, s^*) \in O(nm)$.*

References

1. Alonso, L., Chassaing, P., Reingold, E.M., Schott, R.: The worst-case chip problem. Information Processing Letters 89(6), 303–308 (2004)
2. Wiener, G.: Search for a majority element. Journal of Statistical Planning and Inference 100(2), 313–318 (2002)
3. Aigner, M.: Variants of the majority problem. Discrete Applied Mathematics 137(1), 3–25 (2004)
4. Alonso, L., Reingold, E.M., Schott, R.: Determining the majority. Information Processing Letters 47(5), 253–255 (1993)
5. Saks, M.E., Werman, M.: On computing majority by comparisons. Combinatorica 11, 383–387 (1991), doi:10.1007/BF01275672
6. Blecher, P.M.: On a logical problem. Discrete Mathematics 43(1), 107–110 (1983)
7. Wildon, M.: Knights, spies, games and ballot sequences. Discrete Mathematics 310(21), 2974–2983 (2010)
8. Hanajiri, A.: On knights, knaves, spies and yes-men problems. Master's thesis, Keio University (2012) (in Japenese). Supervisor: Prof. Oda
9. Turán, P.: On an extremal problem in graph theory. Mat. és Fiz. Lapok 48, 436–452 (1941) (in Hungarian)

Drawing Graphs with Few Arcs[⋆]

André Schulz

Institut für Mathematische Logik und Grundlagenforschung, Universität Münster, Germany
andre.schulz@uni-muenster.de

Abstract. Let $G = (V, E)$ be a planar graph. An arrangement of circular arcs is called a *composite arc-drawing* of G, if its 1-skeleton is isomorphic to G. Similarly, a *composite segment-drawing* is described by an arrangement of straight-line segments. We ask for the smallest ground set of arcs/segments for a composite arc/segment-drawing. We present algorithms for constructing composite arc-drawings for trees, series-parallel graphs, planar 3-trees and general planar graphs. In the case where G is a tree, we also introduce an algorithm that realizes the vertices of the composite drawing on a $O(n^{1.81}) \times n$ grid. For each of the graph classes we provide a lower bound for the maximal size of the arrangement's ground set.

1 Introduction

A graph is drawn by realizing its vertices as points in the plane and connecting adjacent vertices by continuous curves. There exists a large number of design criteria such as small area, good vertex and angular resolution, or a small number of edge crossings. All these measures assure that vertices and edges in a drawing are distinguishable for the observer. In this paper we propose a novel criterion for aesthetic and readable graph drawings. Our goal is to generate drawings that are easy to *perceive* by the viewer. When reading a drawing the human mind decomposes the received picture into geometric entities such as lines, segments, arcs, disks, circles, and so on. By interpreting the relationship between these entities an understanding of the drawing is obtained. We refer to the number of entities used in the drawing as its *visual complexity.*

Straight edges and the absence of crossings are desirable features for a drawing. A straight edge would be considered as one single entity, whereas, for example, a polygonal chain might be considered as a combination of several geometric entities. Something similar is true for edge crossings. If two edges cross, they introduce a new *perceptional feature* in the drawing, the crossing point. In this paper we go beyond crossing-free straight-line drawings and try to reduce the number of geometric entities of a drawing further. To make this possible, we group edges, such that they form a new entity. For example, if we are able to

[⋆] The research was performed in cooperation with Eurogiga project GraDR - Graph Drawings and Representations, and supported in part by ESF Eurogiga networking grant.

draw a path of the graph as a straight-line segment in the drawing (with vertices in its interior), the visual complexity of the drawing is reduced. More formally, we define.

Definition 1 (Composite drawing). *Let $\mathcal{A}$ be an arrangement of simple geometric bounded 1d objects in the plane. The objects might be subdivided by placing additional vertices on them. Let G be the 1-skeleton of the subdivided arrangement. The arrangement $\mathcal{A}$ is called a* composite drawing *of G. If $\mathcal{A}$ contains only line segments it is called a* composite segment-drawing*, if $\mathcal{A}$ contains also circular arcs it is called a* composite arc-drawing*. The number of arcs/segments of $\mathcal{A}$ refers to the cardinality of the ground set of $\mathcal{A}$.*

Fig. 1 shows examples of composite arc-drawings.

Our motivation for the perception based approach stems partially from the work of the artist Mark Lombardi. Lombardi's visual art was focused on graph drawings of social networks within the political and financial sector [9]. The drawings of Lombardi had a unique style. Maybe the most characteristic feature is the use of circular arcs to represent consecutive edges. These circular-arc paths kept the visual complexity of the drawings low. By aligning edges Lombardi enhanced his drawings with additional informations. For example, these alignments were used to decode temporal or sequential dependencies of events represented by the vertices.

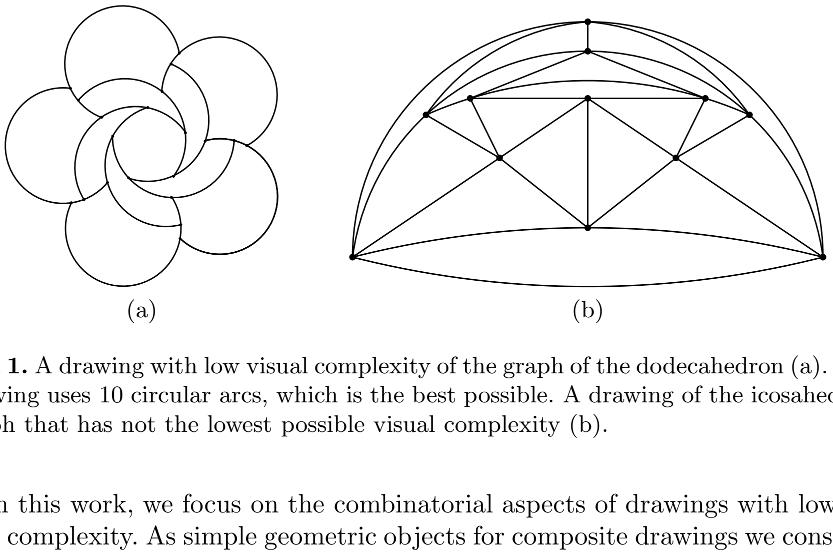


Fig. 1. A drawing with low visual complexity of the graph of the dodecahedron (a). The drawing uses 10 circular arcs, which is the best possible. A drawing of the icosahedron graph that has not the lowest possible visual complexity (b).

In this work, we focus on the combinatorial aspects of drawings with low visual complexity. As simple geometric objects for composite drawings we consider (straight-line) segments and circular arcs. Using straight-line segments is the most natural way for drawing edges, but also circular arcs have been proposed as "edge shapes" before [1,3]. In our understanding, a line segment is a degenerated circular arc, and by a suitable Möbius transformation these segments can be converted to circular arcs. We present bounds on the maximal number of arcs/segments necessary in a composite drawing. Our approach cannot handle

Table 1. Combinatorial results obtained in the paper. Some lower bounds are presented as slightly simplified expressions.

graph class	upper bounds		lower bounds	
	segments [5]	arcs	segments	
trees	$\lceil \lvert E\rvert/2\rceil$	$\lceil \lvert E\rvert/2\rceil$	$\lceil \lvert E\rvert/2\rceil$	[5]
trees on $O(n^{1.81}) \times n$ grid	–	$\lceil 3\lvert E\rvert/4\rceil$	$\lceil \lvert E\rvert/2\rceil$	Thm. 1
series-parallel	$3\lvert E\rvert/4 + 1$	$\lvert E\rvert/2 + 1$	$\lvert E\rvert/4$	Thm. 2
planar 3-trees	$2\lvert E\rvert/3 + 4$	$11\lvert E\rvert/18 + 3$	$\lvert E\rvert/6$	Thm. 3
planar 3-connected	$5\lvert E\rvert/6 + 2$	$2\lvert E\rvert/3$	$\lvert E\rvert/6$	Thm. 4

edge crossings, since every crossing defines a subdivision of geometric objects and hence introduces an additional vertex in the composite drawing. Therefore we only study (noncrossing) drawings of planar graphs, such as trees, series-parallel graphs, and planar 3-trees. Moreover, all graphs that we consider are simple, which means that we forbid parallel edges and self-loops. The results of this paper are listed in Tab. 1. All lower bounds presented in this paper are due to the following simple observation.

Lemma 1. *Let G be a graph with N vertices of odd degree. Every composite arc-drawing or segment-drawing of G requires at least $N/2$ arcs.*

Proof. In every odd degree vertex at least one arc/segment has to start, respectively end. Hence we have at least N endpoints of arcs. □

Related Work. Dujmović et al. [5] studied the complexity of composite segment-drawings. They presented their results in a slightly different form, namely, the bounds on the number of segments are expressed in terms of $|V|$, instead in terms of $|E|$. We are however convinced that a bound in terms of $|E|$ gives a more universal expression since a graph with fewer edges tends to require fewer segments or arcs. The results of Dujmović et al. are presented in Tab. 1. Our results imply that in a composite drawing circular arcs are indeed more powerful than segments, since they are an improvement over the (straight-line segment) bounds of Dujmović et al.. None of the drawings of Dujmović et al. fulfilled additional aesthetic quality criteria. In fact, they stated the problem of designing algorithms with small area as an open problem. From this perspective, Theorem 1 gives the first algorithm that constructs composite drawings on a small polynomial grid.

Recently, user studies comparing straight-line drawings with circular-arc drawing were conducted [10,13]. Both studies showed that certain tasks are easier to carry out by the observer, when straight edges are used. On the other hand, users preferred the aesthetics of circular arc drawings over straight-line drawings in one of the studies [10]. Note that these studies have not considered drawings with low visual complexity, but only drawings with circular arcs. The hypothesis that drawings with low visual complexity are indeed easier to perceive still needs to be checked empirically, which is work in progress.

2 Composite Drawing of Trees

Let $T = (V, E)$ be a tree that we want to realize as a composite segment-drawing. Drawings with $\lceil |E|/2 \rceil$ segments can be constructed by a greedy algorithm [5], which is optimal.

2.1 Grid Drawings of Trees with Few Arcs

In this subsection we show how to draw an unordered tree as a composite arc-drawing with few arcs and the additional constraint that all vertices lie on the $\mathbf{Z}^2$ grid. Our objective is to obtain a drawing that uses few arcs but also requires a small grid. Note that the greedy algorithm yields an embedding on a grid exponential in $O(|V|)$. Therefore, the produced drawing cannot be placed on a polynomial grid.

To obtain a drawing on a small grid we do not aim at drawings with the *lowest* visual complexity. We believe that both grid size, and visual complexity cannot be optimized at the same time. As an easy example, the reader might consider the realization of a simple cycle. Obviously this graph can be drawn with only one circle. However realizing a circle such that it contains many grid points is a highly nontrivial task. To our knowledge the best method uses a grid of size $O(5^{n/4})$ [11].

Heavy Edge Path-Decomposition. The drawing algorithm is based on a decomposition scheme for trees, called the *heavy edge path-decomposition* [12], which works as follows. We root the tree $T = (V, E)$ at some vertex r. Let u be a node of T, then T_u denotes the subtree rooted at u, and $N(u)$ denotes the size of this subtree. For every non-leaf u we select a child v, for which $N(v)$ is maximal (with respect to the size of the subtrees of the other children). The edge (u, v) is called a *heavy edge* and all edges that are not heavy are called *light edges*. A maximal connected component of heavy edges is called a *heavy path*. The tree T decomposes into heavy paths and light edges. Note that every path in T to the root visits at most $\lceil \log |V| \rceil$ light edges.

For the drawing algorithm it is convenient to introduce the following definitions. We call the node on a heavy path that is closest to r its *top node*. The subtree induced by a heavy path is the subtree rooted at its top node. The light edge that links the top node with its parent in T is called *light parent edge*. The *depth* of a heavy path P is defined as follows: If P is not incident to light parent edges of other heavy paths it has depth one. Otherwise we obtain the depth of P by adding one to the maximal depth of a heavy path linked to P via its light parent edge. Note that the subtrees of heavy paths of a fixed depth are all disjoint.

Algorithm Outline. The drawing algorithm works (high-level) as follows. We draw all subtrees of heavy paths with increasing order of their depth. Furthermore, we associate every subtree of a heavy path with an axis-aligned rectangle called its *safe box*. The drawing of a subtree is exclusively contained inside its safe box, and the root of the subtree is placed on the top edge of its safe box, but not on its corners. For convenience we require that every safe box has width at least 3, in particular every leaf is placed inside a 3×1 safe box. When going from a depth k to a depth $k+1$ subtree we arrange the drawings of the subtrees whose heavy paths have smaller depth to a new drawing (details will be given later). The algorithm terminates when the heavy path subtree with the largest depth has been drawn.

Let us explain how to build the subtrees of the heavy paths (see Fig. 2(a) for an illustration). The heavy path is drawn as a single vertical segment. The only exception might be its edge (u, v) incident to the leaf v. Note that every subtree incident to u has to be a leaf as well. Hence all k children of u are leaves, a node with this property is called a *k-fork* in the following. The children of u are placed on the line $y = 0$ and u is placed on $(0, 1)$. In case that k is even, we place the children of u symmetrically around the y-axis such that they have x-coordinates $-k/2, -k/2 + 1, \ldots, -1, 1, \ldots k/2 - 1, k/2$. Two vertices are joined by an arc through u when they have the same absolute x-coordinate (see Fig. 2(a)). In case that k is odd we place the light edges as in the even case and realize the heavy edge (u, v) by extending the vertical segment that contains the remaining heavy path (see Fig. 2(b)).

Assume now that u is not a fork. All safe boxes of subtrees incident to u will be drawn, such that their roots lie on the same horizontal line which is one unit vertically apart from u. Moreover, they will be distributed, such that two of them are connected by a single arc running through u. Note that if we have an odd number of light edges for u, one of the safe boxes does not have a sibling to pair with. In this case we draw the arc as if there would be a sibling (leaf) but we draw only the half of the arc that connects to v. The location of the safe boxes incident to u needs vertical space, which is determined by the safe box with the largest height. The smallest horizontal strip containing all safe boxes incident to u is called a *row*. The tree is constructed such that all of its rows are separated vertically by one unit. The node w following u on the heavy path is placed at the bottom boundary of the row directly below u.

Box Displacement. We now discuss how to arrange the safe boxes within each row. Let u be a node on the heavy path P (not a leaf or fork) and let $v_1, v_2, \ldots, v_k$ be the k children of u not on P. By recursion, the subtrees rooted at the v_is have already been drawn, so we have for every v_i a safe box B_i with width w_i and height h_i. Recall that v_i is placed on the top edge of B_i. We will arrange all safe boxes B_i such that their top edges lie on a common horizontal line, the node v_i has x-coordinate x_i, and the node u is placed one unit above at $x = 0$. To draw multiple light edges with a single arc, we pair two children, say v_i and v_j, and connect both by an arc running through u. This implies that $x_i = -x_j$ for every such pair of vertices.

We determine the location of the safe boxes by a greedy strategy (see Fig. 3). Let ℓ_i be the distance from v_i to the top left corner of B_i, and similarly, let r_i be the distance from v_i to the top right corner of B_i. We first orient all boxes such that $\ell_i \geq r_i$ (it is valid to reflect the whole safe box including the drawing).

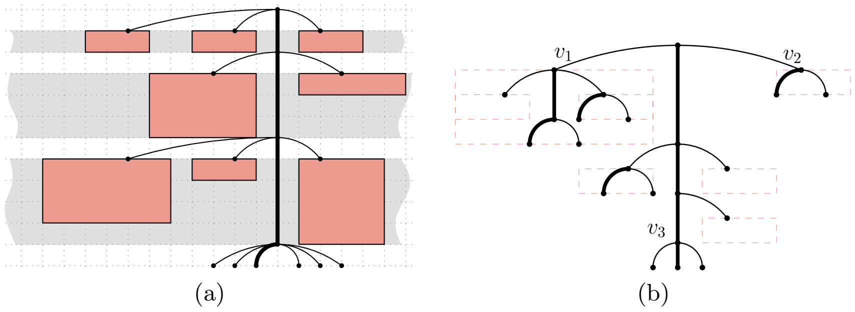


Fig. 2. (a) A drawing of a heavy path's subtree. Rows are drawn shaded and the heavy path is drawn thick. (b) An example of a composite arc-drawing of a tree. The safe boxes used in the algorithm are indicated by dashed rectangles. Vertex v_1 is a hep, v_2 is a 2-fork and, v_3 is a 3-fork.

Then we sort the boxes by ℓ_i in increasing order, and finally we flip all boxes with an even index vertically, such that $r_i \geq \ell_i$.

Assume for now that k is an even number. We place the safe boxes in *rounds.* In round t we place the safe boxes B_{2t-1} and B_{2t} and connect them by an arc passing through u. For convenience we introduce the following notation: If a box B_i is placed left of the heavy edge, then $c_i := r_i$ and $b_i := \ell_i$, otherwise $c_i := \ell_i$ and $b_i := r_i$.

In the first round we place B_1 and B_2. Without loss of generality we assume that $c_1 \leq c_2$ (otherwise the strategy is symmetric). We place B_2 as close as possible to the $x = 0$ line. Since no safe boxes have been placed before, we only have to avoid the heavy edge emanating from u, hence, the safe box is placed such that $x_2 = c_2+1$. Next, we place B_1. The location of x_1 is already determined since we have fixed x_2. Let $\mathcal{S}$ be vertical strip with smallest width centered at the y-axis that contains the safe boxes placed so far. In the following rounds we place the remaining safe boxes such that they are separated from $\mathcal{S}$ by one unit and update $\mathcal{S}$ after every round.

In case k is odd, only one safe box needs to be placed in the final round. We draw the final safe box on the left side, such that it is separated from $\mathcal{S}$ by one unit. When all safe boxes have been arranged we determine the *width of the displacement* Δ, that is the distance between the most extreme top corners. The only exception is when $k = 1$; in this case Δ equals the width of the only safe box plus 2.

Lemma 2. *Assume we carried out the box displacement for the safe boxes incident to some u on P by the greedy strategy as explained above. We have*

$$\Delta \leq 7/4 \sum_{i=1}^{k} w_i.$$

The proof of the lemma can be found in the full version. Let t be the top node of a heavy path P. After carrying out the box displacements for all rows we

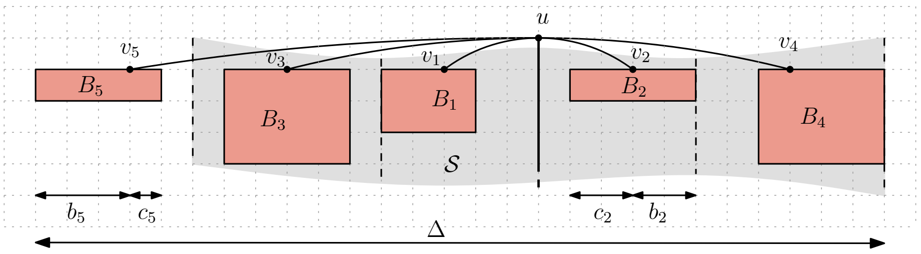


Fig. 3. A snapshot during the execution of the greedy strategy for the box displacement. The safe boxes up to B_4 have already been placed. When placing the last box B_5 we avoid the restricted strip $\mathcal{S}$. The boundaries of the strip $\mathcal{S}$ after round 1 and 2 are drawn as dashed lines.

can define the safe box for the subtree of P. Its width is determined by the row with the maximal displacement width. By construction, t lies on the top edge, but not on a corner of the new safe box. Fig. 3 shows an example of the greedy strategy.

Lemma 3. *By inductively laying out the safe boxes with the greedy strategy explained above, the heavy edge path-decomposition yields a drawing where every vertex is placed on a $O(n^{1.81}) \times n$ grid.*

Proof. In every inductive step we construct a drawing of a subtree and its safe box out of smaller safe boxes. Assume we have k such safe boxes $B_1, \dots, B_k$. Due to Lemma 2 the width of the new safe box is at most $7/4 \sum_{i=1}^{k} w_i$, since it might happen that all safe boxes are placed in one row. On the other hand, at least one box is placed in every row, and these rows are vertically separated by one unit. This shows that the height of the new safe box is at most $m + \sum_{i=1}^{k} h_i$, for m being the number of rows plus one.

The claim of the lemma follows by induction. We first discuss the height. When a small box contains only a single vertex, its height is one. When combining the small boxes to a new subtree, we obtain as new height $m + \sum_{i=1}^{k} h_i$. This new subtree, however, has at least the vertices contained in the smaller safe boxes and the m vertices on its heavy path. Hence the height of its safe box is at most the number of its vertices.

For the width we notice that due to the heavy edge path-decomposition the recursion depth is at most $\lceil \log n \rceil$. By induction a subtree of a heavy path with depth k and n' vertices is contained inside a safe box of width at most $3 \cdot (7/4)^k \cdot n'$. Hence the whole tree is contained in a box of width $3n \cdot (7/4)^{\lceil \log n \rceil}$ which is upper bounded by $O(n^{1.81})$. □

Analysis. A node is a *heavy even-prefork* (short *hep*), if its heavy edge child is a k-fork, with k even. Fig. 2(b) illustrates the definitions. A charging scheme for the "saved edges" in forks and heps leads to the following lemma, whose proof can be found in the full version of the paper. Combining Lemma 2, 3, 4 yields Theorem 1.

Lemma 4. *Let $T = (V, E)$ be a tree drawn as a composite arc-drawing with the algorithm based on the heavy edge path-decomposition. Then the drawing uses at most $\lceil 3|E|/4 \rceil$ arcs.*

Theorem 1. *The algorithm for realizing a tree $G = (V, E)$ as composite arc-drawings uses at most $\lceil 3|E|/4 \rceil$ arcs. The computed drawing realizes all vertices on a $O(n^{1.81}) \times n$ grid, for $n = |V|$.*

3 Composite Drawings of 2-Trees and Planar 3-Trees

In this section we study composite arc-drawings of series-parallel graphs (also known as 2-trees) and planar 3-trees. We start with the series-parallel graphs.

Every series-parallel graph $G = (V, E)$ can be decomposed into a sequence of paths $E_1, E_2, \ldots E_k$, such that (1) the endpoints for every path not E_1 lie both on some path E_j with smaller index (the path between the two endpoints on E_j is called a *nested interval*), (2) no interior point of a path is contained in a path with smaller index, and (3) all nested intervals are either disjoint or contain each other [6]. Such a decomposition is called a *nested open ear-decomposition.* Based on the series-parallel composition history of G a nested open ear-decomposition can be easily constructed.

Theorem 2. *Let $G = (V, E)$ be a series-parallel graph. Based on a nested open ear-decomposition we can obtain a composite arc-drawing with at most $(|E|+1)/2$ arcs. For every n there is a series-parallel graph $G = (V, E)$ with more than n vertices, whose composite segment-drawings need at least $|E|/4 + 1/2$ segments.*

The proof of the theorem can be found in the full paper.

The next class of graphs we consider are the planar 3-trees. A planar 3-tree is a triangulation that can be defined recursively as follows: Suppose $G = (\{v_1, \ldots, v_n\}, E)$ is a triangulation, we can pick one of its faces, say it is spanned by the vertices v_i, v_j, v_k and add a new vertex u inside this face together with the three edges connecting v_i, v_j, v_k with u. By this we remove one face and introduce 3 new faces. This operation is called a *stacking operation.* Any graph that can be generated from a triangle by a sequence of stacking operations is called a *planar 3-tree.* We say that a planar 3-tree is *k-fan* if it has $k + 3$ vertices and it contains the triangle v_1, v_2, v_3 and for every $4 \leq i \leq k + 3$ the edges $(v_i, v_1), (v_i, v_2)$, and (v_i, v_{i-1}).

To develop an algorithm for a composite arc-drawing we first introduce a crucial lemma. For the lemma we need the following definitions. A triangle is called *spherical* if its edges are circular arcs that do not intersect and every angle at a triangle corner is larger than zero and smaller than π. We say a vertex v inside a spherical triangle S is *spherically visible* from a triangle corner c, if there exists a circular arc connecting c and v that lies entirely in S (see Fig. 4(a)).

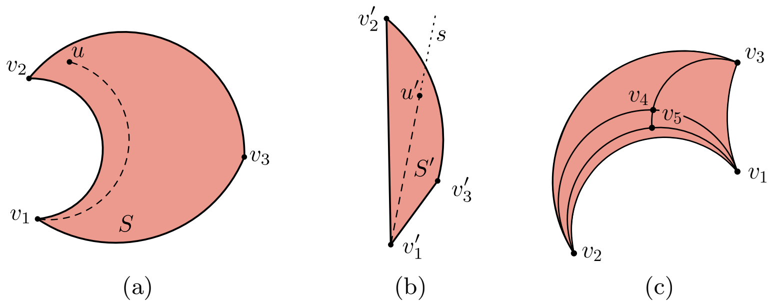


Fig. 4. (a) A spherical triangle spanned by v_1, v_2, v_3 with interior point u. (b) The image of the same spherical triangle under a Möbius transformation that turns two boundary arcs into straight-lines. (c) Construction in the proof of Lemma 6.

Lemma 5. *Let S be a spherical triangle and let u be a point inside S. Then u is spherically visible from every of the three corners of S. Furthermore, the arc that witnesses the spherical visibility and the boundary arcs of the corresponding corner c have all a distinct tangent at c, if u is not on a boundary arc incident to c.*

Proof. Let the corners of S be v_1, v_2, v_3. We prove the statement for the corner v_1. Let f be the Möbius transformation that maps the arcs v_1v_2 and v_1v_3 to straight-line segments. Clearly $u' := f(u)$ lies inside $S' := f(S)$. We set $v_i' := f(v_i)$. Let s be the ray that starts in v_1' and is pointed towards u'. The ray s cannot intersect the arcs $v_1'v_2'$ and $v_1'v_3'$, since Möbius transformations are conformal and therefore the angles at v_2' and v_3' in S' are both less than π. It follows that s hits first the vertex u' and then the boundary of S' without reentering. Therefore, the Möbius function f^{-1} maps the segment $v_1'u'$ to a circular arc that witnesses the spherical visibility of u in S. Clearly the tangents of v_1v_2, v_1v_3, and $f^{-1}(s)$ are all distinct in v_1 because f and f^{-1} are conformal. Fig. 4 shows an example of a triangle S and its image S'. □

Lemma 6. *Let G be a k-fan with outer face f_0, and let S be a spherical triangle. Then G can be drawn with $k+4$ circular arcs such that the boundary of S realizes f_0.*

Proof. Let us first discuss the case $k = 1$. Let the vertices of f_0 be v_1, v_2, and v_3. The vertex v_4 is placed reasonably close to the arc v_1v_2, such that the arc connecting v_1 with v_2 via v_4 lies inside S. By this we define a new spherical triangle $S' \subsetneq S$, which has the corners v_1, v_2 and v_3. Due to Lemma 5, v_4 is spherically visible from v_3 in S', and therefore we can connect v_3 with v_4 by an arc inside S'. The drawing needs five arcs.

Assume now that G is a 2-fan. We extend the arc ending at v_4 (without changing the curvature), such that it reaches inside the spherical triangle spanned by v_1, v_2, v_4. Let the endpoint of the extended arc be v_5. We can interpolate in between the two arcs between v_1 and v_2 such that we get a circular arc connecting

v_1 and v_2 via v_5. The new arc does not introduce any crossings. See Fig. 4(c) for an illustration. By repeating this argument, we can draw every k-fan with $k + 4$ arcs. $\square$

Theorem 3. *Every planar 3-tree* $G = (V, E)$ *can be drawn with* $3 + 11|E|/18$ *arcs as a composite arc-drawing. For every number* n*, there is a planar 3-tree* $G = (V, E)$ *with more then* n *vertices, whose composite arc-drawings require at least* $|E|/6$ *arcs.*

Proof. Note that we can naturally recurse on a planar 3-tree, since when the first vertex v_4 is stacked on the face $v_1v_2v_3$, the graphs contained in the three interior triangles are planar 3-trees as well. For the drawing algorithm we assume that there were at least 2 stacking operations. Let G_f the subgraph of G that is isomorphic to a k-fan and that includes the boundary face, such that k is maximal. We draw G as discussed in Lemma 6 including all induced 1-fans of G that would lie inside faces of G_f. For every such 1-fan we need 2 arcs. This implies that in the worst case there is a 1-fan for every face in G_f, except for v_1, v_2, v_{k+3}. Therefore we have $2k$ 1-fans, contributing a total of $4k$ arcs. The k-fan requires $k + 1$ arcs for the interior edges. Thus we have $5k + 1$ arcs for the $9k$ interior edges. This shows that the ratio between interior arcs and edges is at most 11/18 (recall that $k \geq 2$). The faces of G_f that contain parts of G with more than one additional vertex are analyzed by recursion. The asserted bound of $3 + 11|E|/18$ follows. The lower bound is due to Lemma 1, since an arbitrarily large planar 3-tree with odd degree vertices only can be easily constructed. $\square$

4 Composite Drawings of 3-Connected Planar Graphs

Let $G = (V, E)$ be a triangulation. We order the vertices of G with respect to some *canonical order* as defined by de Fraysseix, Pach, and Pollack [4]. In particular, let $v_1, v_2, \ldots, v_n$ be the vertices of G, such that D_i is the boundary face of the graph G_i induced by $V_i := \{v_1, v_2, \ldots v_i\}$. The graph G_{i+1} is obtained by introducing the new vertex v_{i+1} that is connected to some (at least 2) vertices of D_i. The boundary face D_{i+1} is updated accordingly, also $D_2 = G_2$ is the initial segment.

The composite arc-drawing is constructed in the reverse order of the canonical order. We start with drawing the face v_1, v_2, v_n, such that v_n lies on a circular arc that connects v_1 and v_2 and furthermore v_1 and v_2 are joined by a circular arc such that the boundary face is convex. We maintain as an invariant that the region enclosed by the current face D_i is strictly convex. Assume that we have already drawn the edges of D_k with $k \geq i$. Let D_j be the face with the largest index that contains points that have not be drawn yet. We draw D_j by adding a straight-line segment ℓ_j that connects the corresponding vertices in D_i. The new vertices $V_j \setminus V_i$ are placed arbitrarily on ℓ_j in the order they appear on D_j. Next we add the (possibly) remaining edges that connect v_i with the points on ℓ_j. The edges can be drawn either as straight-line segments or as circular arcs. To guarantee the convexity of the face D_j we perturb ℓ_j with its incident vertices to a slightly curved circular arc. We continue this way until all vertices are placed. Fig. 1(b) shows a drawing constructed with the described strategy. If G is a planar 3-connected graph, then we can find a similar construction based on the canonical ordering of Kant [8].

Theorem 4. *The above method constructs a composite arc-drawing of a planar 3-connected graph $G = (V, E)$ with at most $2|E|/3$ arcs. For every n there is a triangulation $G = (V, E)$ with more than n vertices whose composite segment-drawings need at least $|E|/6 + 1$ segments.*

Proof. The drawing obtained by the technique explained above draws for every vertex (except v_1,v_2) two edges as one arc. For a planar graph there are at most $3|V| - 6$ edges. Hence, the number of arcs differs from $|E|$ by $|V| - 2$, which proves the first statement of the theorem. The lower bound follows from the lower bound of Theorem 3. $\square$

5 Future Work

In this paper we presented the first algorithms for composite drawings. For all graph classes except for trees there is a gap between the lower and upper bound on the number of necessary arcs. We are interested in tightening these gaps, but we think that new methods are required for a substantial improvement.

This paper concentrates on the combinatorial question, i.e., how small can the visual complexity be. On the other hand, drawings with very low visual complexity might violate other criteria for readable drawings. We addressed this issue in Theorem 1 by combining classical graph drawing criteria (grid size) with low visual complexity. We would like to extend this result for more complicated graph classes in order to construct more readable drawings with low visual complexity.

It is ongoing research to evaluate our hypothesis, that a graph with low visual complexity is easier to percept by the viewer with empirical user studies. Our hope is that we can show that drawings with small visual complexity are easier to memorize and we think this might be especially applicable for drawings of graphs with a small number of vertices.

Finally, we would like to point out, that we are interested in small decompositions of planar graphs into edge-disjoint simple paths. This graph-theoretic question might yield better lower bounds. Although this problem seems elementary, only partial results are known. If the graph is a triangulation it can be decomposed into edge-disjoint simple paths that all have exactly three edges [7]. The same is true for cubic bridge-less graphs [2]. We would like to see a similar bound for general planar 3-connected graphs.

Acknowledgement. I thank Thomas van Dijk, Phillip Hanraths, Jan-Henrik Haunert, Birgit Vogtenhuber, and Alexander Wolff for helpful discussions on this subject.

References

1. Aichholzer, O., Aigner, W., Aurenhammer, F., Dobiášová, K.Č., Jüttler, B., Rote, G.: Triangulations with circular arcs. In: van Kreveld, M.J., Speckmann, B. (eds.) GD 2011. LNCS, vol. 7034, pp. 296–307. Springer, Heidelberg (2011)
2. Bouchet, A., Fouquet, J.-L.: Trois types de décompositions d'un graphe en chaînes. In: Berge, C., Bresson, D., Camion, P., Maurras, J., Sterboul, F. (eds.) Combinatorial Mathematics Proceedings of the International Colloquium on Graph Theory and Combinatorics, vol. 75, pp. 131–141. North-Holland (1983)
3. Cheng, C.C., Duncan, C.A., Goodrich, M.T., Kobourov, S.G.: Drawing planar graphs with circular arcs. Discrete & Computational Geometry 25(3), 405–418 (2001)
4. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10(1), 41–51 (1990)
5. Dujmović, V., Eppstein, D., Suderman, M., Wood, D.R.: Drawings of planar graphs with few slopes and segments. Comput. Geom. 38(3), 194–212 (2007)
6. Eppstein, D.: Parallel recognition of series-parallel graphs. Inf. Comput. 98(1), 41–55 (1992)
7. Häggkvist, R., Johansson, R.: A note on edge-decompositions of planar graphs. Discrete Mathematics 283(1-3), 263–266 (2004)
8. Kant, G.: Drawing planar graphs using the canonical ordering. Algorithmica 16(1), 4–32 (1996)
9. Lombardi, M., Hobbs, R.: Mark Lombardi: Global Networks. Independent Curators (2003)
10. Purchase, H.C., Hamer, J., Nöllenburg, M., Kobourov, S.G.: On the usability of Lombardi graph drawings. In: Didimo, W., Patrignani, M. (eds.) GD 2012. LNCS, vol. 7704, pp. 451–462. Springer, Heidelberg (2013)
11. Sally, J., Sally, P.J.: Roots to research: a vertical development of mathematical problems. American Mathematical Society, Rhode Island (2007)
12. Tarjan, R.E.: Linking and cutting trees. In: Data Structures and Network Algorithms, ch. 5, pp. 59–70. Society for Industrial and Applied Mathematics (1983)
13. Xu, K., Rooney, C., Passmore, P., Ham, D.-H., Nguyen, P.H.: A user study on curved edges in graph visualization. IEEE Transactions on Visualization and Computer Graphics 18(12), 2449–2456 (2012)

Connecting Terminals and 2-Disjoint Connected Subgraphs

Jan Arne Telle and Yngve Villanger

Department of Informatics, University of Bergen, N-5020 Bergen, Norway
{telle,yngvev}@ii.uib.no

Abstract. Given a graph $G = (V, E)$ and a set of terminal vertices T we say that a superset S of T is T-connecting if S induces a connected graph, and S is minimal if no strict subset of S is T-connecting. In this paper we prove that there are at most $\binom{|V \setminus T|}{|T|-2} \cdot 3^{\frac{|V \setminus T|}{3}}$ minimal T-connecting sets when $|T| \leq n/3$ and that these can be enumerated within a polynomial factor of this bound. This generalizes the algorithm for enumerating all induced paths between a pair of vertices, corresponding to the case $|T| = 2$. We apply our enumeration algorithm to solve the 2-Disjoint Connected Subgraphs problem in time $O^*(1.7804^n)$, improving on the recent $O^*(1.933^n)$ algorithm of Cygan et al. 2012 LATIN paper.

1 Introduction

The listing of all inclusion minimal combinatorial objects satisfying a certain property is a standard approach to solving certain NP-hard problems exactly. Some examples are the algorithms for Minimum Dominating Set in time $O^*(1.7159^n)$ [3], for Feedback Vertex Set in time $O^*(1.7548^n)$ [2], and for Minimal Separators in time $O^*(1.6181^n)$ [4]. At the time of their appearance these algorithms were the fastest ones available.

This is an approach that usually requires little in the way of correctness arguments. For example, in the minimum dominating set problem it is obvious that a dominating set of minimum cardinality is also an inclusion minimal dominating set. The main task in this approach is to firstly enumerate the inclusion minimal objects, preferably by an algorithm whose runtime is within a polynomial factor of the number of such objects, and secondly to provide a good upper bound on the number of objects. Probably the most famous example is the polynomial delay enumeration algorithm for Maximal independent set [7] where there are matching upper and lower bounds on the number of objects [8].

Another case with matching upper and lower bounds is the $O^*(3^{\frac{n}{3}})$ folklore algorithm enumerating all induced paths between two fixed vertices u and v in an n-vertex graph[1]. In this paper we consider some generalizations of this graph problem. We first generalize to the enumeration of induced paths starting in v and ending in a vertex from a given set R, with no intermediate vertices in $N(R)$.

[1] We have not been able to find a proof of this algorithm in the literature. The graph in Figure 1, with $|R| = 1$, shows optimality of the algorithm, up to polynomial factors.

The algorithm we give for this generalization will be optimal, up to polynomial factors. Given a subset of vertices T let us say that a superset S of T is T-connecting if S induces a connected graph, and that S is minimal T-connecting if no strict subset of S is T-connecting. Our main generalization is the following enumeration task:

Enumeration of minimal T-connecting Sets
Input: A graph $G = (V, E)$ and a set $T \subseteq V$.
Output: All minimal T-connecting sets.

Note that for the case $|T| = 2$ the minimal T-connecting sets are in 1-1 correspondence with the set of induced paths between the two vertices of T. We give an algorithm for Enumeration of minimal T-connecting Sets with runtime $O^*(\binom{n-|T|}{|T|-2} \cdot 3^{\frac{n-|T|}{3}})$ where $|T| \leq n/3$. For $|T| > n/3$ a trivial $O^*(2^{n-|T|})$ brute force enumeration can be used. We apply this enumeration algorithm to solve the following problem:

2-Disjoint Connected Subgraphs
Input: A connected graph $G = (V, E)$ and two disjoint subsets of terminal vertices $Z_1, Z_2 \subseteq V$.
Question: Does there exist a partition A_1, A_2 of V, with $Z_1 \subseteq A_1, Z_2 \subseteq A_2$ and $G[A_1]$, $G[A_2]$ both connected?

The general version of this problem with an arbitrary number of sets was used as one of the tools in the result of Robertson and Seymour showing that Minor containment can be solved in polynomial time for every fixed pattern graph H [11]. We require the input graph to be connected since otherwise it is easy to reduce the problem to a connected component.

Let us look at some previous work on this problem. Motivated by an application in computational geometry, Gray et al [6] showed that 2-Disjoint Connected Subgraphs is NP-complete on planar graphs. van't Hof et al [12] showed that on general graphs it is NP-complete even when $|Z_1| = 2$ and also that it remains NP-complete on P_5-free graphs but is polynomial-time solvable on P_4-free graphs. Notice that the naive brute-force algorithm that tries all 2-partitions of non-terminal vertices runs in time $O(2^k n^{O(1)})$, where $k = n - |Z_1 \cup Z_2|$. This shows that 2-Disjoint Connected Subgraphs is fixed-parameter tractable when parameterizing by the number of non-terminals. However, Cygan et al [1] show that breaking this $O^*(2^k)$ barrier for the number k of non-terminals would contradict the Strong Exponential Time Hypothesis, and that a polynomial kernel for this parameterization would imply $NP \subseteq coNP/poly$. Paulusma and van Rooij [10] gave an algorithm with runtime $O^*(1.2051^n)$ for P_6-free graphs and asked whether it was possible to solve the problem in general graphs faster than $O(2^n n^{O(1)})$. This question was recently answered affirmatively by Cygan et al [1] who gave an algorithm for 2-Disjoint Connected Subgraphs on general graphs, based on the branch and reduce technique, with runtime $O^*(1.933^n)$.

Our algorithm for 2-Disjoint Connected Subgraphs on general graphs will be based on Enumeration of minimal T-connecting Sets and have runtime $O^*(1.7804^n)$.

Our paper is organized as follows. In Section 2 we give the main definitions. In Section 3 we address the enumeration of induced paths starting in v and ending in a vertex from a given set R, with no intermediate vertices in R. In Section 4 we give an algorithm for Enumeration of minimal T-connecting Sets. In Section 5 we apply this enumeration algorithm to solve the 2-Disjoint Connected Subgraphs problem. We end in Section 6 with some questions.

2 Definitions

We deal with simple undirected graphs and use standard terminology. For a graph $G = (V, E)$ and $S \subseteq V$ we denote by $G[S]$ the graph induced by S. An induced subgraph $G[S]$ for $S \subset V$ is called connected if any pair of vertices of S are connected by a path in $G[S]$. We may also denote the vertex set of a graph G by $V(G)$. We denote by $N[S]$ the set of vertices that are in S or have a neighbor in S, and let $N(S) = N[S] \setminus S$.

A path P of a graph G is a sequence of vertices $(v_1, v_2, \ldots, v_q)$ such that $v_jv_{j+1} \in E$ for $1 \leq j < q$, and the path is called induced if $G[\{v_1, v_2, \ldots, v_q\}]$ has no other edges. A subpath of P is of the form $(v_1, v_2, \ldots, v_i)$ for some $i \leq q$.

Contracting an edge uv into vertex v in a graph G is defined as the operation of adding, for every vertex $w \in N(u) \setminus N[v]$, the edge vw to G if it is not already present, and then deleting u and all edges incident to u. Notice that a graph is connected after the contraction operation if and only if it was connected before the contraction operation.

Given a graph $G = (V, E)$, a vertex set $T \subset V$, a vertex $v_1 \in V \setminus T$, and an induced path $P = (v_1, v_2, ..., v_q)$ in $G[V \setminus T]$, we define the *branch depth* of path P to be

$$b(P) = |N[\{v_1, v_2, ..., v_{q-1}\}]| - 1.$$

3 Induced Paths from a Vertex to a Set of Vertices

It is folklore knowledge that the set of induced paths between a pair of vertices in an n-vertex graph can be enumerated in $O^*(3^{\frac{n}{3}})$ time. We have not been able to find a written proof of this in the literature. In the following theorem the induced paths between a pair of vertices is a special case, thus providing a generalization of a well known result.

Theorem 1. *Given a graph $G = (V, E)$, a vertex $v \in V$ and $R \subseteq V \setminus N[v]$, we can enumerate all induced paths from v to a vertex of $N(R)$, with no intermediate vertex in $N[R]$, in time $O^*(3^{\frac{|V \setminus R|}{3}})$.*

We actually want the paths from v to R, but since these paths must have the second-to-last vertex in $N(R)$ we state the result as above. Theorem 1 will follow

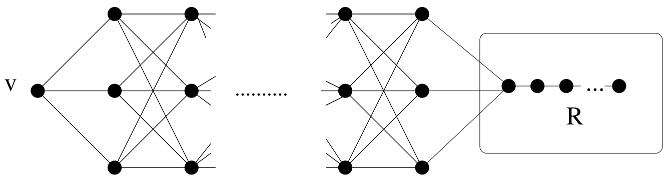


Fig. 1. The number of induced paths between vertex v and a vertex of $N(R)$ (the rightmost column of 3 vertices) is $3^{\frac{n-|R|-1}{3}}$. Since each such path P has branch depth $b(P) = n - |R| - 1$ this graph shows tightness of Lemma 2 when $b(P)$ is a multiple of 3. If $b(P) = 3i+1$ then replace one column of 3 vertices by 4 vertices and if $b(P) = 3i+2$ add a new column of 2 vertices. R induces a connected graph so the number of minimal $R \cup \{v\}$-connecting sets is also $3^{\frac{n-|R|-1}{3}}$.

from Lemma 2, which is stated in terms of branch depth of paths in order to be used for the branching algorithm in the next section. Since the branch depth of each induced path from v to $N(R)$, with no intermediate vertex in $N(R)$, is at most $|V \setminus R| - 1$, Theorem 1 will follow from Lemma 2 below and is tight up to polynomial factors, see Figure 1. We start with a combinatorial lemma.

Lemma 1. *Fix a non-negative integer t and let T be a rooted tree where any root-to-leaf path $v_1, v_2, ..., v_q$ has $\Sigma_{i=1}^{q}\, c(v_i) \leq t$, with $c(v)$ the number of children of node v. The maximum number of leaves that T can have is $l(t)$ with $l(1) = 1$ and for $t \neq 1$*

$$l(t) = \begin{cases} 3^i & \textit{if } t = 3i, \\ 4 \cdot 3^{i-1} & \textit{if } t = 3i+1, \\ 2 \cdot 3^i & \textit{if } t = 3i+2. \end{cases}$$

Proof. We first show that for any t there is a tree U_t achieving the maximum, where all nodes at the same level have the same number of children. For any t let T_t be any rooted tree achieving the maximum. Define $r(t)$ as the number of children of the root of T_t. In the tree U_t all nodes at level $i \geq 1$ will have $u(i)$ children, with $u(i)$ defined level-by-level as follows. The root of U_t will have the same number of children as the root of T_t, in other words we define $u(1) = r(t)$. The sum of the number of children of nodes on any path from a child of the root of U_t to a leaf of U_t should be $t - u(1)$, thus nodes at level 2 of U_t should have the same number of children as the root of $T_{t-u(1)}$, in other words we define $u(2) = r(t - u(1))$. Continuing like this we get that in general $u(i) = r(t - \Sigma_{1 \leq j < i}\, u(j))$. By induction on t it follows that U_t has as many leaves as T_t and any root-to-leaf path has t children.

Assume U_t has p levels. We then have that $u(1) + u(2) + ... + u(p-1) = t$ and that $u(1) \cdot u(2) ... \cdot u(p-1)$, the number of leaves of U_t, is maximized. Since the product of these integers is maximized we can assume that we have no integer $x \geq 4$ among them since then we could replace x by $2 \cdot (x-2) \geq x$ which does not decrease the product nor changes the sum of the integers. Also, if 2 appears then it appears at most twice since we could replace $2 \cdot 2 \cdot 2$ by $3 \cdot 3 > 2 \cdot 2 \cdot 2$. This implies that the number of leaves in U_t is $l(t)$ as stated in the Lemma.

Note that $l(t)$ is the maximum number of maximal independent sets in a graph on t vertices [8,5]. For the connection to the largest integer which is the product of positive integers with sum t see e.g. [13].

Lemma 2. *Given a graph $G = (V, E)$, a vertex $v_1 \in V$, $R \subseteq V \setminus N[v_1]$, and an integer t. Then there exist at most $l(t)$ induced paths $P = (v_1, v_2, ..., v_q)$ in G such that*

(a) $b(P) \leq t$,
(b) $v_i \notin N[R]$ for $1 \leq i \leq q - 1$, and
(c) $v_q \in N(R)$.

Furthermore all these paths can be enumerated in $O^*(3^{\frac{t}{3}})$ time.

Proof. The enumeration algorithm will be a standard backtracking algorithm starting in v_1 that checks all choices. At the first step the choices for v_2 are the vertices in $N(v_1)$. In general, when we have a subpath $P = (v_1, v_2, ..., v_i)$, if $v_i \notin N(R)$ the choices for v_{i+1} are the vertices in $N(v_i) \setminus N[\{v_1, v_2, ..., v_{i-1}\}]$, and if this latter set is empty then the subpath P will not be part of the rooted tree T of all possible choices. On the other hand, if $v_i \in N(R)$ then P is a root to leaf path in T. Thus, in the rooted tree T of all possible choices, if we label the nodes of T with the vertex chosen, the set of paths from the root to a leaf in T will be in 1-1 correspondence with the set of paths satisfying (b) and (c) in the Lemma.

Consider such a path $P = (v_1, v_2, ..., v_q)$. By definition the branch depth of P is $b(P) = |N[\{v_1, v_2, ..., v_{q-1}\}]| - 1$. Consider the root-to-leaf path P_T in T corresponding to P. For any $1 \leq i < q-1$ the children of the node in P_T labelled v_i have labels $N(v_i) \setminus N[\{v_1, v_2, ..., v_{i-1}\}]$, and the node labelled v_q is a leaf. Thus the children of all nodes of P_T have distinct labels and the union of all these labels is $N[\{v_1, v_2, ..., v_{q-1}\}] \setminus \{v_1\}$. Thus the sum of the number of children over all nodes on P_T is exactly $b(P)$.

Consider any rooted tree T having the property that for any root-to-leaf path the sum of the number of children of all nodes on this path is at most t. Lemma 1 bounds the number of leaves in such a tree to $l(t)$. By the above observations, and the fact that $l(t) \leq 3^{t/3}$ since $2 \leq 3^{2/3}$ and $4 \leq 3^{4/3}$, this proves the Lemma.

This enumeration algorithm is optimal to within polynomial factors, see Figure 1.

4 Enumeration of Minimal *T*-Connecting Sets

Theorem 1 with $|R| = 1$ can be viewed as an enumeration of all minimal T-connecting sets when $T = \{u, v\}$. We now generalize this approach to an arbitrary terminal set T by a branching algorithm. The following observation will be used to simplify our branching algorithm.

Lemma 3. *Given $G = (V, E)$, $T \subseteq V$, and two vertices $u, v \in T$ such that $uv \in E$. Let G' be the graph obtained by contracting edge uv into v. Then there is a one to one mapping between minimal T-connecting Sets in G and minimal $T \setminus \{u\}$-connecting Sets in G'.*

Proof. For every minimal T-connecting Set S in G we can contract edge uv and obtain a minimal $T \setminus \{u\}$-connecting Set $S' = S \setminus \{u\}$ in G'. For every minimal $T \setminus \{u\}$-connecting Set S' in G' we can observe that $G[S' \cup \{u\}]$ is a T-Connecting Set in G and it is also minimal as $u \in T$.

Consider **Algorithm Main Enumeration**. It will solve Enumeration of minimal T-connecting Sets for any graph $G = (V, E)$ and $T \subseteq V$. Let us first give the informal intuition for the algorithm. We fix a vertex $u \in T$ and using the algorithm of Lemma 2 we find all induced paths from u to $N(T \setminus \{u\})$. For each of these paths $P = (u, v_2, ..., v_q)$ we again call the algorithm of Lemma 2, but now on the graph G' obtained by recursively contracting the edge uv_i, in increasing order for $i = 2$ to q. In this graph G' the path P together with the vertices of T that P has in its neighborhood, have all been contracted into u. The path we find in G' will in G start at some vertex of P or a neighbor in T and we see that we start forming a tree of paths. We carry on recursively in this way until the collection of paths spans all of T, note however that the vertices of these paths may induce a graph containing cycles. To avoid repeating work we label vertices by a total order and use this ordering to guide the recursive calls.

Lemma 4. *Given $G = (V, E)$, $T \subseteq V$ and $|T| \leq n/3$ Algorithm Main Enumeration will:*

1. *output every minimal T-connecting Set of G,*
2. *output, for any integer $r \in [0..|V \setminus T|]$, at most $\binom{|V \setminus T|}{|T|-2} \cdot 3^{r/3}$ vertex sets $S \supseteq T$ such that $|N[S] \setminus T| \leq r$, and*
3. *run in $O^*(\binom{|V \setminus T|}{|T|-2} \cdot 3^{|V \setminus T|/3})$ time.*

Proof. 1.) Let us first argue that every minimal T-connecting vertex set is output by the algorithm. In the case where $|T| \leq 1$ the single vertex set T is output by the algorithm. In the remaining cases $|T| > 1$.

Let S be a minimal T-connecting vertex set. Our goal will be to show that there will be a call MCS(C, X) performed by the algorithm in which $T \cup C = S$. Initially $C = X = \emptyset$ so we trivially have $T \cup C \subseteq S$ and $X \cap S = \emptyset$. Consider a call MCS(C, X) where we have $|T \cup C|$ maximized under the constraint $T \cup C \subseteq S$ and $S \cap X = \emptyset$. We show by contradiction that $T \cup C = S$ for this call MCS(C, X). Assume, by sake of contradiction, that there is a terminal vertex not in C_u, i.e. not in the component of $G[T \cup C]$ containing u, i.e. that $T' \neq \emptyset$. Let v_2 be the lowest numbered vertex of $N(C_u) \cap S$. As S is minimal we have that $G[S]$ is connected but $G[S \setminus \{v_2\}]$ is not connected. By the minimality of S we have that $G'[S']$ is connected but $G'[S' \setminus \{v_2\}]$ is not connected for $S' = (S \setminus C_u) \cup \{u\}$. Vertex v_2 is not a vertex of $C \cup T$ and as S is minimal we have that each connected component of $G'[S' \setminus \{v_2\}]$ contains a vertex of T'. If this was not the case, this

```
Algorithm Main Enumeration
Input: A graph $G = (V, E)$ and terminal set $T \subseteq V$
Output: A family of sets containing all minimal $T$-connecting Sets
begin
    assign each vertex a unique label between 1 and $|V|$
    choose $u \in T$
    MCS$(\emptyset, \emptyset)$
end

Procedure MCS$(C, X)$
Parameter $C$: vertex set used to connect $T$
Parameter $X$: vertices not to explore in this call
begin
if $G[T \cup C]$ is connected then output $T \cup C$
else
    set $C_u \supseteq C$ as vertex set of connected component of $G[T \cup C]$ containing $u$
    set $T' = T \setminus C_u$ i.e. the terminals not yet connected to $u$ by $C$
    set $G'$ to be graph obtained from $G$ by contracting edges of $G[C_u]$ to $u$
    call the algorithm of Lemma 2 on $G'[V(G') \setminus X]$ with $v_1 = u$ and $R = T'$
    for every path $P = (v_1, v_2, \ldots, v_q)$ output by that call
        MCS$(C \cup \{v_2, \ldots, v_q\},\ X \cup \{w \in N(C_u) : label(w) < label(v_2)\})$
    end-for
end
```

component could simply be removed from S without changing the connectivity between vertices of T. Let B be a connected component of $G'[S' \setminus \{v_2\}]$ not containing u. By the previous arguments B contains a vertex of T'. Therefore the call of the algorithm in Lemma 2 on graph $G'[V(G') \setminus X]$ with $R = T'$ will find a path $P = (u, v_2, \ldots, v_q)$ with all vertices in S and with v_q a neighbor of a vertex of T' in B and containing only vertex v_2 from $N(C_u)$. This would lead to a recursive call where C would be updated to $C \cup \{v_2, \ldots, v_q\} \subseteq S$, and to X there would not be added any vertices of S as v_2 had lowest label among all vertices in $N(C_u) \cap S$, contradicting the maximality of $|T \cup C|$ under the constraint $T \cup C \subseteq S$ and $S \cap X = \emptyset$.

2.) We bound the number of recursive calls in the algorithm and thus also the number of vertex sets that is output. Our objective will be to prove that the number of recursive calls MCS(C, X) where $r = |N[C_u] \setminus T|$ and p is the number of times a path is added to C, is at most $\binom{|X|+p}{p-1} \cdot 3^{r/3}$. Note that p is equal to the depth of the recursion. Let $x = |X|$. Since the algorithm ensures that $X \subset N(C_u)$, $p \leq |T| - 1$, and at least one vertex is added to C for each found path so $p \leq |C|$, we have that $x + p \leq |N[C_u] \setminus T| = r$. Given that $|T| \leq n/3$ and thus $|V \setminus T| \geq 2|T|$ it is clear that $\binom{|V \setminus T|}{|T|-2} \geq \binom{x+p}{p-1}$ and the claim of the lemma follows.

The proof will be by induction on $s = x + p$. We assume without loss of generality that $|T| \geq 2$. The first call is MCS$(\emptyset, \emptyset)$ in which case $p = 0$, and this is in fact the only call where $x + p \leq 0$. The execution of MCS$(\emptyset, \emptyset)$ will call the algorithm of Lemma 2 on G' with $v_1 = u$ and $R = T'$ and make a recursive call MCS(C, X) for each path P output by the algorithm of Lemma 2. Consider such a call MCS(C, X) originating from path P. This call will have $x = |X| \geq 0$, $p = 1$, and it will have $r = |N[C_u] \setminus T| \geq b(P)$. The number of paths P with $b(P) \leq r$ output by the algorithm of Lemma 2 applied to the execution of MCS$(\emptyset, \emptyset)$ on G' with $v_1 = u$ and $R = T'$ is at most $3^{r/3}$. Since $3^{r/3} \leq \binom{x+p}{p-1} \cdot 3^{r/3}$ for $p = 1$ we have just established the base case $s = x+p \leq 1$ this also covers all cases where $p \leq 1$ in our induction.

In the induction step we consider the case where $s = x+p \geq 2$ and $p > 1$. Let MCS(C', X') be a call and let $x' = |X'|$, $r' = |N[C'_u] \setminus T|$, and p' be the number of paths added, or equivalently the depth of the recursion. By the induction hypothesis we assume that the bound holds for the number of calls MCS(C', X') where $x' + p' \leq x + p - 1$.

Every call MCS(C, X) where $x + p = s$ is created by a call MCS(C', X') and a path $P = (v_1, v_2, \ldots, v_q)$ such that $C = C' \cup \{v_2, \ldots, v_q\}$, $p' = p - 1$, and $X = X' \cup \{w \in N(C'_u) : \mathit{label}(w) < \mathit{label}(v_2)\}$. As each vertex from $N(C'_u) \setminus X'$ chosen as v_2 will create a unique size of the set $X = X' \cup \{w \in N(C'_u) : \mathit{label}(w) < \mathit{label}(v_2)\}$ for the next recursive call there is at most one choice for v_2 starting from a fixed MCS(C', X') when it should lead to a recursive call MCS(C, X) where $x+p = s$. However, there are choices for the sub-path vertices $(v_3, \ldots, v_q)$, but these vertices can be chosen only among $V \setminus (N[C'_u] \cup T)$, since v_2 is fixed in $N(C'_u)$ and the path P is induced. Note that any such sub-path has branch-depth at most $|N[C_u] \setminus (N[C'_u] \cup T)|$. We can use Lemma 2 to bound the number of such sub-paths, as follows. By applying Lemma 2 to the graph obtained from $G[V \setminus (N(C'_u) \setminus \{v_2\})]$ by contracting $C'_u \cup \{v_2\}$ to u with $v_1 = u$ and with $R = T \setminus C'_u$ we deduce that the number of such sub-paths is at most $3^{(|N[C_u] \setminus (N[C'_u] \cup T)|)/3}$.

This means that the number of calls MCS(C, X) where $x + p = s$ is at most the number of calls MCS(C', X') where $C' \subseteq C$, $X' \subseteq X$ thus $x' \leq x$, and $p' = p - 1$, times $3^{(|N[C_u] \setminus (N[C'_u] \cup T)|)/3}$. By the induction hypothesis we have that the number of calls MCS(C', X') where $x' + p' < s$ is at most $\binom{x'+p-1}{p-1-1} \cdot 3^{|N[C'_u] \setminus T|/3}$. Multiplying these two factors we get $\binom{x'+(p-1)}{(p-1)-1} \cdot 3^{|N[C'_u] \setminus T|/3} . 3^{(|N[C_u] \setminus (N[C'_u] \cup T)|)/3}$ which can be simplified to $\binom{x'+(p-1)}{(p-1)-1} \cdot 3^{(|N[C_u] \setminus T|)/3}$.

Thus it remains to bound the number of calls MCS(C', X') that can make a new recursive call MCS(C, X) where $x + p = s$ to be at most $\binom{x+p}{p-1}$. We know that each call MCS(C', X') can only make calls where $x+p = s$ when it uses the unique vertex $v_2 \in N(C'_u) \setminus X$ as the second vertex of the path. Thus it suffices to count these calls, and let y be the number of such calls. We have that

$$y \leq \sum_{i=0}^{x} \binom{i+p-1}{p-1-1} \cdot 3^{(|N[C_u] \setminus T|)/3}$$

Using the standard observation that $\sum_{k=0}^{n} \binom{k}{m} = \binom{n+1}{m+1}$ we can conclude that $y \leq \binom{x+p}{p-1} \cdot 3^{(|N[C_u] \setminus T|)/3}$ and the proof is completed.

3.) In the previous claim we bounded the number of recursive calls in the algorithm to $\binom{|V \setminus T|}{|T|-2} \cdot 3^{r/3}$ vertex sets $S \supseteq T$ such that $|N[S] \setminus T| \leq r$ and $|T| \leq n/3$, and as Lemma 2 ensures that all paths in a single call can be enumerated within a polynomial delay it follows that the polynomial bound holds.

Using Lemma 4 we can make the following conclusion.

Theorem 2. *For an n vertex graph $G = (V, E)$ and a terminal set $T \subseteq V$ where $|T| \leq n/3$ there is at most $\binom{n-|T|}{|T|-2} \cdot 3^{(n-|T|)/3}$ minimal T-connecting vertex sets and these can be enumerated in $O^*(\binom{n-|T|}{|T|-2} \cdot 3^{(n-|T|)/3})$ time.*

5 The 2-Disjoint Connected Subgraphs Problem

Let us now use Theorem 2 to solve the 2-Disjoint Connected Subgraphs problem. Recall that the problem is defined as follows:

2-Disjoint Connected Subgraphs
Input: A connected graph $G = (V, E)$ and two disjoint subsets of vertices $Z_1, Z_2 \subseteq V$.
Question: Does there exist two disjoint subsets A_1, A_2 of V, with $Z_1 \subseteq A_1, Z_2 \subseteq A_2$ and $G[A_1]$, $G[A_2]$ both connected?

Theorem 3. *There exists a polynomial space algorithm that solves the* 2-Disjoint Connected Subgraphs *problem in $O^*(1.7804^n)$ time.*

Proof. Let us assume without loss of generality that $|Z_1| \leq |Z_2|$ and let $\alpha = |Z_1|/n$; note that $0 < \alpha \leq 0.5$. The algorithm has a first stage that finds a list of potential candidates for A_1 and a second stage that checks each candidate to see if it can be used as a solution.

Consider first the case where $|Z_1 \cup Z_2| > 2(0.0839)$. In this case the algorithm simply loops over all subsets of $V \setminus (Z_1 \cup Z_2)$ to list every vertex subset $A \subseteq (V \setminus Z_2)$ where $Z_1 \subseteq A$. As $|Z_1 \cup Z_2| > 2(0.0839)$ we get that the number of such subsets is at most $2^{n-2(0.0839)} \leq 1.7804^n$ and they can be found in $O^*(1.7804^n)$ time.

In the remaining case $|Z_1 \cup Z_2| \leq 2(0.0839)$ and in particular $\alpha \leq 0.0839$ as $|Z_1| \leq |Z_2|$. Vertices of Z_2 are of no use when searching for a potential set A_1 so it suffices to consider the graph $G[V \setminus Z_2]$. As $|Z_1| \leq 0.0839n \leq n(1-2(0.0839))/3$ we know that by the algorithm for Enumeration of minimal T-connecting Sets of Theorem 2 all minimal Z_1-connecting sets of $G[V \setminus Z_2]$ can be enumerated in $O^*(\binom{n-|Z_1|-|Z_2|}{|Z_1|-2} \cdot 3^{(n-|Z_1|-|Z_2|)/3})$ time. As $|Z_1| \leq |Z_2|$ it is clear that $\alpha n \leq |Z_2|$. The number $|Z_2|$ only contributes negatively so we can observe that

$$\binom{n-|Z_1|-|Z_2|}{|Z_1|-2} \cdot 3^{(n-|Z_1|-|Z_2|)/3} \leq \binom{(1-2\alpha)n}{\alpha n - 2} \cdot 3^{(1-2\alpha)n/3}.$$

By using $\beta = (1-2\alpha)$ and Stirling approximation we get that $\binom{(1-2\alpha)n}{\alpha n-2} \leq \binom{\beta n}{\alpha n}$ is $O^*((\frac{\beta^\beta}{\alpha^\alpha \cdot (\beta-\alpha)^{(\beta-\alpha)}})^n)$ or $O^*((\frac{(1-2\alpha)^{(1-2\alpha)}}{\alpha^\alpha \cdot (1-3\alpha)^{(1-3\alpha)}})^n)$. It is not hard to verify by computer that the maximum value of $(\frac{(1-2\alpha)^{(1-2\alpha)}}{\alpha^\alpha\cdot(1-3\alpha)^{(1-3\alpha)}})^n \cdot 3^{(1-2\alpha)n/3}$ for $0 < \alpha \leq 0.0839$ occurs when $\alpha = 0.0839$ and that $\binom{(1-2\alpha)n}{\alpha n-2} \cdot 3^{(1-2\alpha)n/3} \leq 1.7804^n$ for $\alpha = 0.0839$. Thus, we can conclude that when $\alpha \leq 0.0839$ a list of all minimal Z_1-connecting sets can be found in time $O^*(1.7804^n)$.

For the second stage of the algorithm, for every listed set A, the algorithm tests if vertices of Z_2 are contained in the same connected component of $G \setminus A$ and if so the algorithm returns the solution with $A_1 = A$ and A_2 being the vertices of the connected component of $G \setminus A$ containing Z_2. This is clearly a solution to the problem. Conversely, if there is a solution A_1, A_2 to the problem, then there is clearly one where A_1 is a minimal Z_1-connecting set.

Finally, as a simple branching algorithm is used for both cases, the algorithm uses polynomial space.

6 Conclusion

The graph in Figure 1 shows that our algorithm for Enumeration of minimal T-connecting Sets given by Theorem 2 is optimal, up to polynomial factors, for the case $|T| = 2$. Is the algorithm optimal, up to polynomial factors, also for larger T, let us say $|T| \leq 0.1n$?

Let us remark that our algorithm for Enumeration of minimal T-connecting Sets can be used to give a $O^*(\binom{|V\setminus T|}{|T|-2} \cdot 3^{\frac{|V\setminus T|}{3}})$ algorithm for Steiner Tree with unit weights on terminal vertices T. This is upper bounded by $O^*(1.8778^n)$ when balanced with the standard brute force search, but will not beat the fastest algorithm for this problem, which is by Nederlof [9] and has runtime $O^*(1.3533^n)$ using polynomial space.

The algorithm given in this paper for Enumeration of minimal T-connecting Sets may have more applications in the future, apart from 2-Disjoint Connected Subgraphs, in particular for problems where the enumeration of all solutions is required.

References

1. Cygan, M., Pilipczuk, M., Pilipczuk, M., Wojtaszczyk, J.O.: Solving the 2-disjoint connected subgraphs problem faster than 2^n. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 195–206. Springer, Heidelberg (2012)
2. Fomin, F.V., Gaspers, S., Pyatkin, A.V., Razgon, I.: On the minimum feedback vertex set problem: Exact and enumeration algorithms. Algorithmica 52(2), 293–307 (2008)
3. Fomin, F.V., Grandoni, F., Pyatkin, A.V., Stepanov, A.A.: Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. ACM Transactions on Algorithms 5(1) (2008)
4. Fomin, F.V., Villanger, Y.: Treewidth computation and extremal combinatorics. Combinatorica 32(3), 289–308 (2012)
5. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms. Texts in Theoretical Computer Science. An EATCS Series. Springer (2011)

6. Gray, C., Kammer, F., Löffler, M., Silveira, R.I.: Removing local extrema from imprecise terrains. Comput. Geom. 45(7), 334–349 (2012)
7. Johnson, D.S., Papadimitriou, C.H., Yannakakis, M.: On generating all maximal independent sets. Inf. Process. Lett. 27(3), 119–123 (1988)
8. Moon, J., Moser, L.: On cliques in graphs. Israel Journal of Mathematics 3, 23–28 (1965), doi:10.1007/BF02760024
9. Nederlof, J.: Fast polynomial-space algorithms using möbius inversion: Improving on steiner tree and related problems. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 713–725. Springer, Heidelberg (2009)
10. Paulusma, D., van Rooij, J.M.M.: On partitioning a graph into two connected subgraphs. Theor. Comput. Sci. 412(48), 6761–6769 (2011)
11. Robertson, N., Seymour, P.D.: Graph minors. xiii. The disjoint paths problem. J. Comb. Theory, Ser. B 63(1), 65–110 (1995)
12. van't Hof, P., Paulusma, D., Woeginger, G.J.: Partitioning graphs into connected parts. Theor. Comput. Sci. 410(47-49), 4834–4843 (2009)
13. Vatter, V.: Maximal independent sets and separating covers. American Mathematical Monthly 118(5), 418–423 (2011)

Author Index