15th International Symposium on Parameterized and Exact Computation

IPEC 2020, December 14–18, 2020, Hong Kong, China (Virtual Conference)

Edited by

Yixin Cao

Marcin Pilipczuk

LIPIcs – Vol. 180 – IPEC 2020 www.dagstuhl.de/lipics

Editors

Yixin Cao
Hong Kong Polytechnic University, China
yixin.cao@polyu.edu.hk

Marcin Pilipczuk
University of Warsaw, Poland
malcin@mimuw.edu.pl

ACM Classification 2012
Theory of computation → Parameterized complexity and exact algorithms

ISBN 978-3-95977-172-6

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-172-6.

Publication date
December, 2020

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.IPEC.2020.0

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Regular Papers

15th International Symposium on Parameterized and Exact Computation (IPEC 2020).
Editors: Yixin Cao and Marcin Pilipczuk
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

PACE Solver Descriptions

Preface

This volume contains the papers presented at IPEC 2020: the 15th International Symposium on Parameterized and Exact Computation. IPEC 2020 was held virtually on 14–18 December 2020 and was organized, together with ISAAC 2020, by The Hong Kong Polytechnic University, China.

The International Symposium on Parameterized and Exact Computation (IPEC, formerly IWPEC) is a series of international symposia covering research in all aspects of parameterized and exact algorithms and complexity. Started in 2004 as a biennial workshop, it became an annual event in 2009.

Previous iterations of I(W)PEC	
2004	Bergen, Norway
2006	Zürich, Switzerland
2008	Victoria, Canada
2009	Copenhagen, Denmark
2010	Chennai, India
2011	Saarbrücken, Germany
2012	Ljubljana, Slovenia
2013	Sophia Antipolis, France
2014	Wrocław, Poland
2015	Patras, Greece
2016	Aarhus, Denmark
2017	Vienna, Austria
2018	Helsinki, Finland
2019	Münich, Germany

In response to the call for papers, 53 abstracts were submitted, which led to 51 papers considered by the program committee. Each considered submission received at least 3 reviews. The reviews came from the 14 members of the program committee and from 62 external reviewers, together contributing 160 reviews. The program committee held electronic meetings through the EasyChair platform. In the end, the program committee selected 26 of the submissions for presentation at the symposium and inclusion in these proceedings.

The Best Paper Award and Best Student Paper Award were given jointly to two co-winners:

- Tuukka Korhonen, for his paper *Finding Optimal Triangulations Parameterized by Edge Clique Cover*,
- Łukasz Bożyk, Jan Derbisz, Tomasz Krawczyk, Jana Novotná and Karolina Okrasa, for their paper *Vertex deletion into bipartite permutation graphs*.

IPEC 2020 hosted an award ceremony with a plenary talk for the 2020 EATCS-IPEC Nerode Prize for outstanding papers in the area of multivariate algorithmics. The Nerode prize committee consisted of Hans L. Bodlaender, Anuj Dawar, and Virginia Vassilevska Williams. They awarded the prize to the following series of two papers:

- Dániel Marx, *Parameterized graph separation problems*. Theoretical Computer Science 351(3), 394–406 (2006)
- Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, Igor Razgon, *A fixed-parameter algorithm for the directed feedback vertex set problem*. J. ACM 55(5) (2008)

IPEC 2020 also invited Yoichi Iwata to present a tutorial. Finally, IPEC 2020 hosted the award ceremony and poster session of the fifth *Parameterized Algorithms and Computational Experiments* challenge, PACE. This yearly challenge was conceived in Fall 2015 to deepen the relationship between parameterized algorithms and practice. These proceedings contain a report by Ł. Kowalik, M. Mucha, W. Nadara, M. Pilipczuk, M. Sorge, and P. Wygocki on the 2020 PACE challenge.

15th International Symposium on Parameterized and Exact Computation (IPEC 2020).
Editors: Yixin Cao and Marcin Pilipczuk
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We would like to thank the program committee, together with the external reviewers, for their commitment in the difficult paper selection process. We also thank all the authors who submitted their work for consideration. We are grateful to the local organizers of ISAAC 2020 and IPEC 2020, for their work on the local arrangements. Finally, we acknowledge the financial support of The Hongkong Polytechnic University, the University of Warsaw, and the European Research Council (ERC) via European Union's Horizon 2020 research and innovation programme Grant Agreement no. 714704.

Yixin Cao and Marcin Pilipczuk
Hongkong and Warsaw, October 2020

Program Committees

IPEC 2020 Program Committee

Marthe Bonamy	CNRS Bordeaux	France
Yixin Cao (co-chair)	Hong Kong Polytechnic University	China
Yijia Chen	Fudan University	China
David Eppstein	UC Irvine	USA
Eun Jung Kim	CNRS	France
Marvin Künnemann	MPII	Germany
Euiwoong Lee	New York University	USA
Pasin Manurangsi	Google Research	USA
Pranabendu Misra	MPII	Germany
Irene Muzi	TU Berlin	Germany
Marcin Pilipczuk (co-chair)	University of Warsaw	Poland
Marc Roth	University of Oxford	UK
R.B. Sandeep	IIT Dharwad	India
Darren Strash	Hamilton College	USA

PACE 2020 Program Committee

Łukasz Kowalik (chair)	University of Warsaw	Poland
Marcin Mucha	University of Warsaw	Poland
Wojciech Nadara	University of Warsaw	Poland
Marcin Pilipczuk	University of Warsaw	Poland
Manuel Sorge	University of Warsaw	Poland
Piotr Wygocki	University of Warsaw	Poland

15th International Symposium on Parameterized and Exact Computation (IPEC 2020).
Editors: Yixin Cao and Marcin Pilipczuk

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of External Reviewers

Akanksha Agrawal
Jungho Ahn
Saeed Akhoondian Amiri
Miriam Backens
Max Bannach
Rémy Belmonte
Andreas Björklund
Jannis Bulian
Dibyayan Chakraborty
Rajesh Chitnis
Syamantak Das
Eduard Eiben
Nick Fischer
Robert Ganian
Niels Grüttemeier
Sylvain Guillemot
Jiong Guo
Thekla Hamm
Meike Hatzel
Anjeneya Swami Kare
Fabian Klute
Dušan Knop
Yusuke Kobayashi
Sudeshna Kolay
Martin Koutecky
Bundit Laekhanukit
Michael Lampis
Monique Laurent
Bingkai Lin
Jan Marcinkowski
Neeldhara Misra
Matthias Mnich
Jesper Nederlof
Alantha Newman
Fahad Panolan
Astrid Pieterse
Marko Radovanovic
Bhaskar Ray Chaudhury
Daniel Rehfeldt
Lars Rohwedder
Frances Rosamond
Saket Saurabh
Chris Schwiegelshohn
Roohani Sharma
Sebastian Siebertz
Krzysztof Sornat
Giannos Stamoulis
Raphael Steiner
Torstein Strømme
Hisao Tamaki
Till Tantau
Ioan Todinca
Johan M. M. Van Rooij
Nicole Wein
Philip Wellnitz
Ryan Williams
Michal Wlodarczyk
Karol Węgrzycki
Mingyu Xiao
Chao Xu
Chao-Tung Yang
Chihao Zhang

15th International Symposium on Parameterized and Exact Computation (IPEC 2020).
Editors: Yixin Cao and Marcin Pilipczuk
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Akanksha Agrawal (1)
Indian Institute of Technology Madras, Chennai, India

Jørgen Bang-Jensen (2)
University of Southern Denmark, Odense, Denmark

Max Bannach (27, 28)
Institute for Theoretical Computer Science, Universität zu Lübeck, Germany

Benjamin Bergougnoux (3)
Department of Informatics, University of Bergen, Norway

Sebastian Berndt (27, 28)
Institute for IT Security, Universität zu Lübeck, Germany

Hans L. Bodlaender (4)
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Édouard Bonnet (3)
Université Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Łukasz Bożyk (5)
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Nick Brettell (3, 6)
School of Mathematics and Statistics, Victoria University of Wellington, New Zealand

Ruben Brokkelkamp (29)
Centrum Wiskunde & Informatica (CWI), The Netherlands

Karthekeyan Chandrasekaran (7)
University of Illinois, Urbana-Champaign, IL, USA

Steven Chaplick (8)
Maastricht University, The Netherlands

Jan Derbisz (5)
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

Louis Dublois (9)
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, Paris, France

Eduard Eiben (2, 10)
Royal Holloway, University of London, UK

Carl Einarson (11)
Royal Holloway, University of London, UK

Andreas Emil Feldmann (17)
Department of Applied Mathematics, Charles University, Prague, Czech Republic

Fedor V. Fomin (12)
Department of Informatics, University of Bergen, Norway

Marcelo Garlet Milani (13)
Technische Universität Berlin, Chair of Logic and Semantics, Germany

Petr A. Golovach (8, 12)
Department of Informatics, University of Bergen, Norway

Dishant Goyal (14)
Indian Institute of Technology Delhi, India

Elena Grigorescu (7)
Purdue University, West Lafayette, IN, USA

Gregory Gutin (2)
Royal Holloway, University of London, UK

Tim A. Hartmann (8)
RWTH Aachen, Germany

Tatsuhiko Hatanaka (15)
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Felix Hommelsheim (15)
Fakultät für Mathematik, TU Dortmund University, Germany

Jake Horsfield (6)
School of Computing, University of Leeds, UK

Radek Hušek (16)
Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Davis Issac (17)
Hasso Plattner Institute, Potsdam, Germany

Gabriel Istrate (7)
West University of Timişoara, Romania; e-Austria Research Institute, Timişoara, Romania

15th International Symposium on Parameterized and Exact Computation (IPEC 2020).
Editors: Yixin Cao and Marcin Pilipczuk
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Takehiro Ito (15)
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Ashwin Jacob (18, 19)
The Institute of Mathematical Sciences, HBNI, Chennai, India

Ragesh Jaiswal (14)
Indian Institute of Technology Delhi, India

Leon Kellerhals (20)
Technische Universität Berlin, Faculty IV, Algorithmics and Computational Complexity, Germany

Dušan Knop (8, 16)
Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Czech Republic

Tomohiro Koana (20)
Technische Universität Berlin, Faculty IV, Algorithmics and Computational Complexity, Germany

Yasuaki Kobayashi (21)
Kyoto University, Japan

Yusuke Kobayashi (15)
Research Institute for Mathematical Sciences, Kyoto University, Japan

Tuukka Korhonen (22, 30)
Department of Computer Science, University of Helsinki, Finland

Łukasz Kowalik (23, 37)
Institute of Informatics, University of Warsaw, Poland

Tomasz Krawczyk (5)
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

Shubhang Kulkarni (7)
University of Illinois, Urbana-Champaign, IL, USA

Amit Kumar (14)
Indian Institute of Technology Delhi, India

O-joung Kwon (3)
Department of Mathematics, Incheon National University, South Korea; Discrete Mathematics Group, Institute for Basic Science (IBS), Daejeon, South Korea

Michael Lampis (9)
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, Paris, France

Young-San Lin (7)
Purdue University, West Lafayette, IN, USA

William Lochet (10, 12)
Department of Informatics, University of Bergen, Norway

Daniel Lokshtanov (24)
University of California Santa Barbara, CA, USA

Konrad Majewski (23)
Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw, Poland

Diptapriyo Majumdar (18)
Royal Holloway, University of London, UK

Dejun Mao (31)
The University of Tokyo, Japan

Tomáš Masařk (16)
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland; Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Czech Republic

Pranabendu Misra (12)
Max Planck Institute for Informatics, Saarbrücken, Germany

Amer E. Mouawad (24)
Department of Computer Science, American University of Beirut, Lebanon

Marcin Mucha (37)
Institute of Informatics, University of Warsaw, Poland

Andrea Munaro (6)
School of Mathematics and Physics, Queen's University Belfast, UK

Moritz Mühlenthaler (15)
Laboratoire G-SCOP, Grenoble INP, Université Grenoble Alpes, France

Wojciech Nadara (37)
Institute of Informatics, University of Warsaw, Poland

Jesper Nederlof (25)
Utrecht University, Algorithms and Complexity Group, The Netherlands

Jana Novotná (5)
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland; Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Karolina Okrasa (5)
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland; Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Yota Otachi (21)
Nagoya University, Japan

Giacomo Paesani (6)
Department of Computer Science, Durham University, UK

Fahad Panolan (19, 24)
Department of Computer Science and Engineering, IIT Hyderabad, India

Vangelis Th. Paschos (9)
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, Paris, France

Daniël Paulusma (6)
Department of Computer Science, Durham University, UK

Marcin Pilipczuk (37)
Institute of Informatics, University of Warsaw, Poland

Ashutosh Rai (17)
Department of Applied Mathematics, Charles University, Prague, Czech Republic

Venkatesh Raman (18, 19)
The Institute of Mathematical Sciences, HBNI, Chennai, India

M. S. Ramanujan (1)
University of Warwick, Coventry, UK

Felix Reidl (11)
Birkbeck, University of London, UK

Vibha Sahlot (19)
The Institute of Mathematical Sciences, HBNI, Chennai, India

Saket Saurabh (10, 12, 26)
Institute of Mathematical Sciences, Chennai, India; Department of Informatics, University of Bergen, Norway

Martin Schuster (27, 28)
Institute for Epidemiology, Kiel University, Germany

Roohani Sharma (12)
Max Planck Institute for Informatics, Saarbrücken, Germany

Sebastian Siebertz (24)
University of Bremen, Germany

Manuel Sorge (37)
Institute of Informatics, University of Warsaw, Poland

Ben Strasser (32)
Independent Researcher, Germany

Vorapong Suppakitpaisarn (31)
The University of Tokyo, Japan

Akira Suzuki (15)
Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Sylwester Swat (33)
Institute of Computing Science, Poznan University of Technology, Poland

Céline M. F. Swennenhuis (25)
Eindhoven University of Technology, Combinatorial Optimization Group, The Netherlands

Prafullkumar Tale (26)
CISPA - Helmholtz Center for Information Security, Saarbrücken, Germany

James Trimble (34, 35)
School of Computing Science, University of Glasgow, Scotland, UK

Raymond van Venetië (29)
Korteweg-de Vries Institute, University of Amsterdam, The Netherlands

Mees de Vries (29)
University of Amsterdam, The Netherlands

Magnus Wahlström (2)
Royal Holloway, University of London, UK

Marieke van der Wegen (4)
Department of Information and Computing Sciences, Utrecht University, The Netherlands; Mathematical Institute, Utrecht University, The Netherlands

Jan Westerdiep (29)
Korteweg-de Vries Institute, University of Amsterdam, The Netherlands

Marcel Wienöbst (27, 28)
Institute for Theoretical Computer Science, Universität zu Lübeck, Germany

Marcin Wrochna (36)
University of Oxford, UK

Piotr Wygocki (37)
Institute of Informatics, University of Warsaw, Poland

Zijian Xu (31)
The University of Tokyo, Japan

Anders Yeo (2)
University of Southern Denmark, Odense, Denmark

Minshen Zhu (7)
Purdue University, West Lafayette, IN, USA

On the Parameterized Complexity of Clique Elimination Distance

Akanksha Agrawal
Indian Institute of Technology Madras, Chennai, India
akanksha@iitm.ac.in

M. S. Ramanujan
University of Warwick, Coventry, UK
R.Maadapuzhi-Sridharan@warwick.ac.uk

Abstract

Bulian and Dawar [Algorithmica, 2016] introduced the notion of elimination distance in an effort to define new tractable parameterizations for graph problems and showed that deciding whether a given graph has elimination distance at most k to any minor-closed class of graphs is fixed-parameter tractable parameterized by k [Algorithmica, 2017].

In this paper, we consider the problem of computing the elimination distance of a given graph to the class of cluster graphs and initiate the study of the parameterized complexity of a more general version – that of obtaining a modulator to such graphs. That is, we study the $(\eta, \textsc{Clq})$-Elimination Deletion problem ($(\eta, \textsc{Clq})$-ED Deletion) where, for a fixed η, one is given a graph G and $k \in \mathbb{N}$ and the objective is to determine whether there is a set $S \subseteq V(G)$ such that the graph $G - S$ has elimination distance at most η to the class of cluster graphs.

Our main result is a polynomial kernelization (parameterized by k) for this problem. As components in the proof of our main result, we develop a $k^{\mathcal{O}(\eta k + \eta^2)} n^{\mathcal{O}(1)}$-time fixed-parameter algorithm for $(\eta, \textsc{Clq})$-ED Deletion and a polynomial-time factor-$\mathsf{min}\{\mathcal{O}(\eta \cdot \mathsf{opt} \cdot \log^2 n), \mathsf{opt}^{\mathcal{O}(1)}\}$ approximation algorithm for the same problem.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Elimination Distance, Cluster Graphs, Kernelization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.1

Funding *Akanksha Agrawal*: supported by the PBC Program of Fellowships for Outstanding Post-doctoral Researchers from China and India when a part of the work was carried out.

1 Introduction

A popular methodology for studying the parameterized complexity of problems is to consider *parameterization by distance from triviality* [19]. In this methodology, the idea is to try and lift the tractability of special cases of generally hard computational problems, to tractability of instances that are "close" to these special cases (i.e., close to triviality) for appropriate notions of "distance from triviality". With some exceptions (see, for example, [13, 21]), this approach typically has two components – (i) recognition algorithms that determine whether the input is indeed close to triviality and possibly compute a (approximate) witness, and (ii) solution algorithms that exploit the structure expressed by the witness in order to solve computational problems. In graph problems, a standard measure of distance from triviality is the size of a vertex modulator to a specific graph class, i.e., a set of vertices whose deletion results in a graph belonging to a specific graph class. This way of parameterizing graph problems has led to a rich collection of sophisticated algorithmic and lower bound machinery over the last two decades. Of particular interest to us in this work are two specific strands of research that fall under this framework.

In one strand, the goal is to enhance existing notions of distance from triviality by exploiting some form of *structure* underlying vertex modulators rather than just the size bound. This line of exploration has led to the development of several new notions of distance from triviality [11, 5, 6, 17, 16, 10]. Of special interest to us in this line of research is the notion of *elimination distance* introduced in [5]. Bulian and Dawar [5] introduced the notion of elimination distance in an effort to define tractable parameterizations that are more general than the modulator size for graph problems. We refer the reader to Section 2 for a formal definition of this parameter. In their work, they focused on the Graph Isomorphism (GI) problem and showed that GI is fixed-parameter tractable (FPT) when parameterized by the elimination distance to graphs of bounded degree. In follow-up work, Bulian and Dawar [6] showed that deciding whether a given graph has elimination distance at most k to any minor-closed class of graphs is fixed-parameter tractable parameterized by k (i.e., can be solved in time $f(k)n^{\mathcal{O}(1)}$). The second strand focuses on enhancing the set of "base classes" that capture the notion of triviality and study algorithms that compute or use small modulators to these classes. For instance, the computation and use of modulators into various hereditary graph classes (see, e.g., [14, 18, 3, 15, 24] for a partial list relevant to this paper) has been extensively explored.

In more recent years, efforts have been made to simultaneously build upon these two strands by retaining the notion of small modulators as the measure of distance from triviality, but enhancing the notion of triviality to include graphs that have bounded elimination to a second, well-understood graph class. Hols et al. [20] recently presented a comprehensive study of the classic Vertex Cover problem parameterized by the size of a smallest modulator to graphs that have bounded elimination distance to specific hereditary graph classes. They provided an elegant (partial) characterization of parameterizations that permit polynomial kernelizations for Vertex Cover. However, their focus is on solution (utilising the modulator) rather than recognition, and so they do not focus on computing the modulators.

In our current work, we draw from both strands of research described above and study the parameterized complexity of the $(\eta, \textsc{Clq})$-Elimination Deletion problem, where one is given a graph G and $k \in \mathbb{N}$ with the objective of determining whether there is a set $S \subseteq V(G)$ such that the graph $G - S$ has elimination distance at most η to the class of cluster graphs (disjoint union of cliques). We call graphs with elimination distance at most η to the class of cluster graphs, (η, Clq)-graphs. Our parameter is the size of the modulator k and we note that even for $k = 0$ (i.e., deciding whether the given graph has elimination distance at most η to cluster graphs), this problem is quite non-trivial and even an algorithm with running time $n^{f(\eta)}$ is far from obvious. Indeed, Bulian and Dawar [6] ask about the possibility of extending their approach to obtain a fixed-parameter algorithm for this very problem.

The following is the formal definition of our main problem.

$(\eta, \textsc{Clq})$-Elimination Deletion($(\eta, \textsc{Clq})$-ED Deletion)

Input: A graph G and an integer k.
Parameter: k.
Question: Is there a set $S \subseteq V(G)$ of size at most k, such that $G - S$ is an (η, Clq)-graph?

The central result of this paper is Theorem 1.

▶ **Theorem 1.** $(\eta, \textsc{Clq})$-Elimination Deletion *admits a kernelization algorithm running in time* $\eta^{O(\eta^2)} \cdot n^{\mathcal{O}(1)}$*, that outputs an equivalent instance with* $2^{\mathcal{O}(\eta)} \cdot k^{\mathcal{O}(1)}$ *vertices, where* n *is the number of vertices in the input graph.*

Theorem 1 is in fact a *polynomial kernelization* for $(\eta, \textsc{Clq})$-ED Deletion as η is a constant that is part of the problem description. We have explicitly stated the dependence on η in all our results.

A simple corollary of Theorem 1 (obtained by setting $k = 0$) is an algorithm with running time $\eta^{\mathcal{O}(\eta^2)} \cdot n^{\mathcal{O}(1)}$ that determines whether a given graph is an (η, Clq)-graph (equivalently, we say that the Clq-elimination distance of the given graph is at most η). That is, it is a fixed-parameter algorithm for recognising (η, Clq)-graphs parameterized by η. Towards the proof of Theorem 1, we prove the following two results of independent interest. The first of these results gives an approximation algorithm for $(\eta, \textsc{Clq})$-ED Deletion. The second result gives a moderately exponential-time fixed-parameter algorithm for $(\eta, \textsc{Clq})$-ED Deletion.

▶ **Theorem 2.** *There is an algorithm that, given a graph G on n vertices, runs in time $\eta^{\mathcal{O}(\eta^2)} \cdot n^{\mathcal{O}(1)}$ and outputs a set $S \subseteq V(G)$ of size $\mathcal{O}(\eta \cdot \mathsf{opt}^2 \cdot \log^2 n)$, such that $G - S$ has Clq-elimination distance at most η.*

▶ **Theorem 3.** $(\eta, \textsc{Clq})$-Elimination Deletion *can be solved in time $(k+\eta)^{O(\eta k+\eta^2)} n^{\mathcal{O}(1)}$.*

An important consequence of Theorem 2 and Theorem 3 is a polynomial-time $\mathsf{opt}^{\mathcal{O}(1)}$-approximation algorithm for $(\eta, \textsc{Clq})$-ED Deletion. We remark that the moderately exponential dependence on k in the running time of Theorem 3 is crucially used in obtaining this approximation algorithm and hence, we avoid resorting to meta-theorems in the proof of Theorem 3. This $\mathsf{opt}^{\mathcal{O}(1)}$-approximation algorithm for $(\eta, \textsc{Clq})$-ED Deletion is the starting point of our proof of Theorem 1. Such $\mathsf{opt}^{\mathcal{O}(1)}$-approximation algorithms play a crucial role in bootstrapping kernelization algorithms (see, for example, [22, 2]). They also allow for further consequences when studying kernelization of problems parameterized by the size of the smallest modulator to (η, Clq)-graphs since we do not need to assume that the modulator is given as part of the input. In the literature on such structural parameterizations, it is generally assumed that the modulator is included in the input. Often, such an assumption can be made without loss of generality because the modulator can be $\mathsf{opt}^{\mathcal{O}(1)}$-approximated in polynomial time. However, there are situations where the assumption is necessary due to the lack of such approximations. We refer the reader to [12] for a detailed discussion on formalizing structural parameterizations.

Finally, as part of the proof of Theorem 3, we obtain a constant factor fixed-parameter approximation algorithm for the problem of determining the Clq-elimination distance of a given graph. Moreover, we show that by invoking a result of Czerwiński et al. [8], one can speed up this algorithm at the cost of a worse approximation. The formal statement is given below (we refer the reader to Section 2 for the definition of an (η, Clq)-decomposition).

▶ **Theorem 4.** *There are algorithms $\mathcal{A}_1$, $\mathcal{A}_2$ such that, given a graph G and an integer η, the following hold:*

1. *$\mathcal{A}_1$ runs in time $2^{\mathcal{O}(\eta^2)} \cdot n^{\mathcal{O}(1)}$ and either correctly reports that the Clq-elimination distance of G is more than η, or computes a $(5\eta, \mathsf{Clq})$-decomposition of G.*
2. *$\mathcal{A}_2$ runs in time $2^{\mathcal{O}(\eta)} \cdot n^{\mathcal{O}(1)}$ and either correctly reports that the Clq-elimination distance of G is more than η, or computes an $(\mathcal{O}(\eta^2 \log^{3/2} \eta), \mathsf{Clq})$-decomposition of G.*

Related work

Recently, Lindermayr et al. [27] showed that computing elimination distance to bounded degree graphs is fixed-parameter tractable when the input is planar. Bougeret et al. [2] introduced a measure called bridge-depth and showed that a minor-closed family of graphs

$\mathcal{F}$ has bounded bridge-depth precisely when Vertex Cover admits a polynomial kernel parameterized by the size of a modulator to $\mathcal{F}$ (subject to standard complexity theoretic hypotheses). The notion of elimination distance [5] generalizes the notion of *generalized treedepth* introduced by Bouland et al [4] in an effort to combine the treedepth and max-leaf number parameters. Building on [5] and extending the approach of combining width parameters (treedepth in the case of [5]) and modulator size, Ganian et al. [17] proposed a measure of distance to triviality for CSP that depended on the treewidth of an appropriate graph defined on backdoor sets (these can be thought of as a version of vertex modulators appropriate for use in solving ILP and CSP instances). That is, they introduced a way of combining *treewidth* and modulator size into a single parameter that is stronger than elimination distance. More recently, Eiben et al. [10] continued the line of research into combining modulators and width parameters by studying this parameter in the context of graph problems, where triviality is expressed in terms of bounded rankwidth.

2 Preliminaries

We refer to the book of Diestel [9] for standard graph terminology. Whenever the context is clear, we use n and m to denote the number of vertices and the number of edges in the input graph, respectively. A *vertex cover* in G is a set $S \subseteq V(G)$, such that $G - S$ has no edges. By $\mathsf{vc}(G)$ we denote the size of a minimum sized vertex cover in G. We say that a set $S \subseteq V(G)$ is a *clique* in G if for every distinct $u, v \in S$, we have $\{u, v\} \in E(G)$. We let $\mathsf{Clq}(G)$ denote the set of all cliques in G.

▶ **Proposition 5** ([23, 28]). *We can generate all maximal cliques of a graph with $\mathcal{O}(n^\omega)$ time delay, where ω is the exponent in the running time of matrix multiplication.*[1]

The set $\mathcal{C}(G)$ denotes the set of connected components of G. Consider a graph G. For sets $X, Y \subseteq V(G)$, an X-Y *separator* in G is a set $S \subseteq V(G)$, such that $G - S$ has no x-y path, where $x \in X \setminus S$ and $y \in Y \setminus S$. By $\mathsf{sep}_G(X, Y)$ we denote the size of a minimum sized X-Y separator in G. Our algorithm(s) will rely on existence of balanced separators, which is defined next.

▶ **Definition 6.** For a graph H, a set $Z \subseteq V(H)$ is a *balanced separator* of H, if the connected components of $H - Z$ can be partitioned into two sets, $\mathcal{C}_1$ and $\mathcal{C}_2$, such that $|\cup_{C \in \mathcal{C}_1} V(C)| \leq 2|V(H)|/3$ and $|\cup_{C \in \mathcal{C}_2} V(C)| \leq 2|V(H)|/3$.

For a tree T and vertices $u, v \in V(T)$, we denote the unique path between u and v by $\mathsf{Pth}_T(u, v)$. A *rooted tree* is a tree with a special vertex called the *root* of the tree. Consider a rooted tree T with root r. A vertex $t \in V(T) \setminus \{r\}$ is a *leaf* of T if it is a vertex of degree exactly one in T. Moreover, if $V(T) = \{r\}$, then r is the leaf (as well as the root) of T. A vertex which is not a leaf, is a *non-leaf* vertex. Consider a node $t \in V(T)$. We say that t' is a *child* of t, if $\{t, t'\} \in E(T)$ and t' does not belong to the unique $t - r$ path in T. Furthermore, we say that $t = \mathsf{par}_T(t')$ is the *parent* of t'. A vertex $t' \in V(T)$ (t' can possibly be the same as t) is a *descendant* of t, if in $T - \{\mathsf{par}_T(t)\}$, where $\mathsf{par}_T(t)$ is the parent of t, there is a $t - t'$ path. Note that when $t = r$, then $T - \{\mathsf{par}_T(t)\} = T$, as the parent of r does not exist. (Every vertex in T is a descendant of r.) By $\mathsf{desc}_T(t)$, we denote the set of all descendants of t in T. We drop the subscript T from $\mathsf{par}_T(\cdot)$ and $\mathsf{desc}_T(\cdot)$, when the context is clear. A *rooted forest* is a forest where each of its connected component is a rooted tree. The set of

[1] The current best value of ω is less than 2.3727 [31].

leaves of a rooted forest is the union of the sets of leaves of the trees in this forest. The set of leaves in a rooted forest F is denoted by $\mathsf{Lf}(F)$. The *depth*, denoted by $\mathsf{depth}(T)$ of a rooted tree T is the maximum number vertices in a root to leaf path in T. The *depth*, denoted by $\mathsf{depth}(F)$ of a rooted forest is the maximum over the depths of its rooted trees. We now define the notion of tree decompositions.

▶ **Definition 7.** A *tree decomposition* of a graph G is a pair (T, β), where T is a tree rooted at r and $\beta : V(T) \to 2^{V(G)}$ that satisfies the following properties:

1. $\bigcup_{t \in V(T)} \beta(t) = V(G)$,
2. for every edge $\{u, v\} \in E(G)$ there is a node $t \in V(T)$, such that $u, v \in \beta(t)$, and
3. for every $v \in V(G)$, the graph $T[X_v]$ is a subtree of T, where $X_v = \{t \in V(T) \mid v \in \beta(t)\}$.

For $t \in V(T)$, we call $\beta(t)$ the *bag* of t. The sets in $\{\beta(t) \mid t \in V(T)\}$ are called *bags* of (T, β). We refer to the vertices in $V(T)$ as *nodes*, to distinguish it from the vertices of G. The *width* of the tree decomposition (T, β) is $\max_{t \in V(T)} |\beta(t)| - 1$. The *treewidth* of G, denoted by $\mathsf{tw}(G)$, is the minimum over the widths over all possible tree decompositions of G.

A tree decomposition (T, β) of a graph G is called a *path decomposition* if T is a path. Moreover, *pathwidth* of G, denoted by $\mathsf{pw}(G)$, is the minimum over the widths over all possible path decompositions of G. In the following we state some folklore properties about bounded treewidth graphs, that will be useful later.

▶ **Proposition 8** (Exercise 7.6 [7])**.** *Consider a graph G, a tree decomposition (T, β) of G, and a clique $S \subseteq V(G)$ in G. Then, there is $t \in V(T)$, such that $S \subseteq \beta(t)$.*

▶ **Proposition 9.** *For a graph G, the number of distinct cliques (not necessarily maximal) in G is bounded by $\mathcal{O}(2^{\mathsf{tw}(G)} \cdot n)$.*

▶ **Definition 10** ([25, 7])**.** For a graph H, a path decomposition $\mathcal{P} = (P = (p_1, p_2, \cdots, p_t), \beta : V(P) \to 2^{V(H)})$ of H is a *nice path decomposition* if $\beta(p_1) = \beta(p_t) = \emptyset$, P is rooted at p_t, and every node p_i, for $i \in [t] \setminus \{1, t\}$ is of exactly one of the following types:

1. **Insert Vertex Node.** We have $\beta(p_i) = \beta(p_{i-1}) \cup \{v\}$, for some $v \in V(H) \setminus \beta(p_{i-1})$.
2. **Forget Vertex Node.** We have $\beta(p_i) = \beta(p_{i-1}) \setminus \{v\}$, for some $v \in \beta(p_{i-1})$.

We now state a result regarding computation of a nice path decomposition of a graph, which follows from Lemma 7.2 of [7] and Proposition 8.

▶ **Proposition 11.** *Any graph H that admits a path decomposition of width at most p, also admits a nice path decomposition of width at most p. Moreover, given a path decomposition $\mathcal{P} = (P = (p_1, p_2, \cdots, p_t), \beta)$ of H of width at most p, one can compute a nice path decomposition $\mathcal{P}' = (P' = (p'_1, p'_2, \cdots, p'_{t'}), \beta')$ of H of width at most p in time $\mathcal{O}(p^2 \cdot \max(|V(P)|, |V(H)|))$, such that the root node, p_t, is a forget node and for each $i \in [t]$, there is some $j \in [t']$, with $\beta(p_i) = \beta'(p'_j)$.*

▶ **Definition 12** (Forest embedding)**.** A *forest embedding* of a graph G is a pair (F, f), where F is a rooted forest and $f : V(G) \to V(F)$ is a bijective function, such that for each $\{u, v\} \in E(G)$, either $f(u)$ is a descendant of $f(v)$, or $f(v)$ is a descendant of $f(u)$. The *depth* of the forest embedding (F, f) is the depth of the rooted forest F.[2] The *treedepth* of G, denoted by $\mathsf{td}(G)$, is the minimum over the depths over all possible forest embeddings of G.

[2] Sometimes we slightly abuse the notation for simplicity, and say that, for every $\beta \geq \alpha$, (F, f) is a forest embedding of depth β, where α is the depth of F.

▶ **Definition 13** (Induced forest embedding). *Let (F, f) be a forest embedding of G and let $W \subseteq V(G)$. Define the pair (F', f') as follows:*

- *$V(F') = V(F) \cap f(W)$ and $f' : W \to V(F')$ is defined as $f|_W$, where $f(W) = \bigcup_{w \in W} f(w)$.*
- *For every $u \in V(F')$, if no ancestor of u in F is in $f(W)$, then make u a root.*
- *For every $u, v \in V(F')$ such that u is an ancestor of v in the rooted forest F, we add an edge between u and v (making v a child of u in F') if and only if W is disjoint from $f^{-1}(X)$, where X is the set of internal vertices on the unique u-v path in F.*

We say that (F', f') is the forest embedding induced *by (F, f) on the set W.*

In the following we state some known results regarding treedepth of a graph.

▶ **Proposition 14** (see Excercise 7.54 [7]). *For every graph G, $\mathsf{tw}(G) \leq \mathsf{td}(G)$.*

The next observation states that for every clique S in G, and every forest embedding of G, there is a root-to-leaf path in this forest embedding that contains all the vertices of S.

▶ **Observation 15.** *Consider a graph G, a forest embedding $f : V(G) \to V(F)$ of G into the rooted forest F, and a clique $S \subseteq V(G)$ in G. Then, there is a rooted tree $T \in \mathcal{C}(F)$[3] with root r and a leaf $t \in V(T)$, such that for each $s \in S$, we have $f(s) \in V(\mathsf{Pth}_T(r, t))$.*

Next, we recall the notion of *elimination-distance* introduced by Bulian and Dawar [5]. We rephrase their definition and introduce notation that will facilitate our presentation.

▶ **Definition 16** (Elimination Distance and $(\eta, \mathcal{H})$-decompositions). Consider a family $\mathcal{H}$ of graphs and an integer $\eta \in \mathbb{N}$. An $(\eta, \mathcal{H})$-decomposition of a graph G is a tuple $(X, Y, F, f : X \to V(F), g : \mathcal{C}(G[Y]) \to \mathsf{Lf}(F) \cup \{\bot\})$, where (X, Y) is a partition of $V(G)$ and F is a rooted forest of depth η, such that the following conditions are satisfied:

1. (F, f) is a forest embedding of $G[X]$,
2. each connected component of $G[Y]$ belongs to $\mathcal{H}$, and
3. for a connected component C of $G[Y]$, a vertex $v \in V(C)$, and an edge $\{u, v\} \in E(G)$, either $u \in Y$ or $f(u)$ is a vertex in the unique path in F from r to $g(C)$, where r is the root of the connected component in F containing the vertex $g(C)$.[4]

We say that G admits an $(\eta', \mathcal{H})$-decomposition if there is some $\eta \leq \eta'$, for which there is an $(\eta, \mathcal{H})$-decomposition of G. The *elimination distance* of G to $\mathcal{H}$ (or the *$\mathcal{H}$-elimination distance* of G) is the smallest integer η^* for which G admits an $(\eta^*, \mathcal{H})$-decomposition.

Consider an $(\eta, \mathcal{H})$-decomposition $\mathbb{D} = (X, Y, F, f, g)$ of a graph G. We say that X is the *interior part* of $\mathbb{D}$ and Y is the *exterior part* of $\mathbb{D}$. For a leaf $u \in \mathsf{Lf}(F)$, by $\widehat{P}^{\mathbb{D}}_u$ we denote the the path from u to r in the tree T, where T is the tree rooted at r in F, containing u. Moreover, by $P^{\mathbb{D}}_u$, we denote the graph $G[\{f^{-1}(w) \mid w \in V(\widehat{P}^{\mathbb{D}}_u)\}]$. For a connected component $C \in \mathcal{C}(G[Y])$, by $C^{\mathbb{D}}_{\mathsf{ext}}$ we denote the graph $G[V(C) \cup \{f^{-1}(w) \mid w \in V(\mathsf{Pth}_F(g(C), r))\}]$, where r is the root of the component of F containing $g(C)$. (For the above notations we drop the superscript $\mathbb{D}$, when the context is clear.) The following observation directly follows from the definition of $C^{\mathbb{D}}_{\mathsf{ext}}$ and item 3 of Definition 16.

▶ **Observation 17.** *Consider a graph G with an (η, Clq)-decomposition $\mathbb{D} = (X, Y, F, f, g)$. For every clique S in G the following holds: i) if $S \cap Y = \emptyset$, then there is $u \in \mathsf{Lf}(F)$, such that $S \subseteq V(P_u)$, otherwise, ii) $S \cap Y \neq \emptyset$, and there is $C \in \mathcal{C}(G[Y])$, such that $S \subseteq V(C_{\mathsf{ext}})$.*

[3] Recall that $\mathcal{C}(F)$ denotes the set of connected components of F,
[4] If $g(C) = \bot$, then u must belong to Y.

For a graph G and integer η, by $\mathsf{opt}_\eta(G)$, we denote the size of a minimum sized set $S \subseteq V(G)$, such that $G - S$ admits an (η, Clq)-decomposition. We drop the argument G and the subscript η, whenever the context is clear. For a set $S \subseteq V(G)$, we say that S is *solution*, if $G - S$ has Clq-elimination distance at most η. Furthermore, we say that S is a t-solution if $|S| \leq t$.

3 Overview of our algorithms

In this section, we give high level summaries of the proofs behind our results. For all our algorithms, we assume that the input graph is connected and not a clique.

3.1 Polynomial kernelization for (η, Clq)-ED Deletion (Theorem 1)

We first outline our polynomial kernelization assuming Theorem 2 and Theorem 3.

Our kernelization is heavily inspired by the work of Fomin et al. [14] and Giannopoulou et al. [18]. Fomin et al. gave a polynomial kernelization for the η-Treewidth Deletion problem (and more generally, for the Planar $\mathcal{F}$-deletion problem). Their kernel has size $k^{f(\eta)}$ and subsequently, Giannopoulou et al. [18] developed specialised reduction rules that result in a kernel of size $f(\eta) \cdot k^6$ for the special case of η-Treedepth Deletion. In both these kernelizations, the starting point is a constant-factor approximate modulator to η-treewidth graphs (respectively, η-treedepth graphs). While an approximate η-treedepth modulator can be obtained by repeatedly taking the vertex set of a sufficiently long path in the graph into the modulator, such an approach will not help in our case and we rely on Theorem 2 and Theorem 3 to obtain a polynomial-time $\mathsf{opt}^{\mathcal{O}(1)}$-approximation algorithm for (η, Clq)-ED Deletion.

Indeed, suppose that $n \leq (k+\eta)^{d(\eta k+\eta^2)}$ (d is the constant hidden in the $\mathcal{O}(\cdot)$ notation in Theorem 3). Then, Theorem 2 implies a $\mathsf{opt}^{\mathcal{O}(1)}$-approximation in polynomial time. On the other hand, if $n > (k+\eta)^{d(\eta k+\eta^2)}$, then the algorithm of Theorem 3 runs in polynomial time. This approximation algorithm is the starting point of our proof of Theorem 1.

We now describe the rest of our kernelization algorithm. Using Theorem 4, we obtain a $(5\eta+1, \mathsf{Clq})$-decomposition, $\widetilde{\mathbb{D}} = (X, Y, T, f, g)$, of $G - S$, where T is a rooted tree. The two main objectives of the algorithm are to i) bound the size of a connected component of $G[Y]$ by $(k+\eta)^{\mathcal{O}(1)}$, and ii) bound the degree of a vertex in T (and also the number of connected components of $G[Y]$ associated with a leaf in T via the function g) by $2^{\mathcal{O}(\eta)} \cdot k^{\mathcal{O}(1)}$. Once we achieve the above two, using the fact that T has bounded depth, we can obtain our kernel of the desired size. Each of our reduction rules either decreases the number of non-edges in the input, or deletes a vertex. Thus in total, our algorithm will apply at most $n^2 + n$ reduction rules. We will next give an intuitive description of our reduction rules (and their workings).

Our first reduction rule helps us bound the size of a connected component of $G[Y]$, and we briefly explain the working of this reduction rule, below. Consider a connected component C of $G[Y]$, and let $a = g(C)$. Furthermore, let X_a be the set of vertices from X, that are mapped to the vertices in the path from a to the root of T. For each $u \in X_a \cup S$, we mark $\mathcal{O}(k+\eta)$ neighbors (and non-neighbors) of u in C. After this, our reduction rules remove all unmarked vertices from C. The idea behind the correctness of the above reduction rule is that if u has large neighborhood in C, then any solution S^* to G can delete at most k of these neighbors. Moreover, at most η of these neighbors of u can belong to the interior part of the decomposition for $G - S^*$. Thus, if we mark $\mathcal{O}(k+\eta)$ neighbors of u in C, then we will be able to preserve the information that some neighbors of u in C must belong to

the exterior part of the decomposition. Once we have the above property, we may add the deleted vertex from C to the exterior of (η, Clq)-decomposition for the reduced instance, to obtain such a decomposition for the original instance.

Towards designing our reduction rule for bounding the degree of vertices in T, we first devise another reduction rule that, very roughly speaking, helps up ensure that for the neighborhood of the (sub-)graph below a vertex $a \in V(T)$, in S, induces a clique. This clique-neighborhood property helps us to avoid considering the exact set of neighborhood in S, as the vertices from a clique must belong to a root to leaf path, in any (η, Clq)-decomposition. To achieve the clique-neighborhood property, we add edges between non-adjacent pair of vertices in S that have $(k+\eta)^{\Omega(1)}$-flow between them (the flow value is proportional to bound on the size of a connected components in $G[Y]$). On the other hand, for pairs of vertices in S that do not have $(k+\eta)^{\Omega(1)}$-flow between them, we mark the vertices of a minimum separator for these vertices contained in $G - S$. As the number of separator vertices thus marked is bounded by $(k+\eta)^{\mathcal{O}(1)}$, they do not contribute excessively to the degree of a vertex in T. Fix a vertex a in T that has an unbounded number of children and let $a_1, \ldots, a_\ell$ be the children of a in T. For $i \in [\ell]$, let G_i be the graph $G[V_i]$, where V_i contains all i) $v \in X$, for which $f(v)$ is a descendant of a_i, and ii) all vertices of $C \in \mathcal{C}(G[Y])$, where $g(C)$ is a descendant of a_i. A child a_i of a is *relevant*, if G_i does not contain a marked separator vertex. At this point, for a relevant child a_i of a in T, $N(V(G_i)) \cap S$ is a clique. Let $X_a \subseteq X$ be the set of vertices that are mapped to vertices in the path from a to the root of T. For each $B \subseteq X_a$, we mark $\mathcal{O}(k+\eta)$ relevant children a_i of a, for which $N(V(G_i)) \cap X_a$ is B. Also, for each $s \in S$, we mark $\mathcal{O}(k+\eta)$ relevant child a_i of a, for which $V(G_i)$ has s as a neighbor (resp. non-neighbor). While marking relevant children in this way, we not only consider their neighborhoods as described, but we also do the following: for every possible $\widehat{\eta} \in [\eta]_0$, mark $\mathcal{O}(k+\eta)$ relevant children a_i (satisfying the aforementioned constraints on their neighborhood) that have elimination distance exactly $\widehat{\eta}$. Following this series of markings, if a child a_i of a is unmarked, then we delete all the vertices in $V(G_i)$, from G. Since we have preserved $\mathcal{O}(k+\eta)$ representatives for the neighborhood and the elimination distance, given a solution $\widehat{S}$ for $G - V(G_i)$, and an (η, Clq)-decomposition for $(G - V(G_i)) - S$, we will be able to show that G_i can be appended in an identical fashion to one of preserved representatives, thus giving an (η, Clq)-decomposition for $G - S$. Summing up, when no reduction rules are applicable, we can bound the number of vertices in the graph by $2^{\mathcal{O}(\eta)} \cdot k^{\mathcal{O}(1)}$.

3.2 FPT-algorithm for (η, Clq)-ED Deletion (Theorem 3)

We now give a summary of the proof of Theorem 3 assuming Theorem 4. The first ingredient of our FPT algorithm is the well-known technique of iterative compression, introduced by Reed, Smith, and Vetta [29]. Roughly speaking, using the above, we can reduce our goal to solving a variant of $(\eta, \textsc{Clq})$-Elimination Deletion, where we have a $(k+1)$-solution, call it Z, for G, and the objective is to find a k-solution, $S \subseteq V(G) \setminus Z$ for G. As Z is a solution for G, $G - Z$ admits an (η, Clq)-decomposition. We compute a $(5\eta, \mathsf{Clq})$-decomposition, $\mathbb{D} = (X, Y, F, f, g)$, of $G - Z$ using Theorem 4. Using this $(5\eta, \mathsf{Clq})$-decomposition $\mathbb{D}$, we compute a path decomposition $\mathcal{P} = (P, \beta : V(P) \to 2^{X \cup Z})$ of $G[X \cup Z]$. We then compute a "pathlike decomposition", $\mathcal{T}$ of G, by attaching connected components of $G[Y]$ (which induce cliques), as bags, to $\mathcal{P}$. We remark that $\mathcal{T}$ may not be a tree-decomposition of G, as the edges between vertices in Y and $X \cup Z$ may not be contained in any bag of $\mathcal{T}$. We ensure that each bag of $\mathcal{P}$ is attached to at most one connected component of $G[Y]$ (which is achieved by repeating some bags of $\mathcal{P}$, when necessary), and for any connected component C of $G[Y]$, the neighborhood of C is contained in the bag of $\mathcal{P}$, to which it is attached. We then execute

a dynamic programming algorithm over this pathlike decomposition. Roughly speaking, we define states only for bags that are present in $\mathcal{P}$ and only preserve polynomial-size information regarding the cliques (from $G[Y]$), attached to it. Remembering only polynomial sized information regarding the attached cliques is possible due to the following two properties: i) at most one connected component of $G[Y]$ is attached to a bag from $\mathcal{P}$, which allows us to use "vertex cover" like property to identify vertices that must go to the interior part of the decomposition. ii) As vertices of the connected component C of $G[Y]$ induces a clique, the vertices from C that go to the interior, must belong to one root to leaf path.

We will now intuitively discuss the states for our dynamic programming algorithm. Let $P = (p_1, p_2 \cdots, p_\ell)$, where p_ℓ is the root of P. For $i \in [\ell]$, let G_i be the graph induced on vertices that appear in $\beta(p_{i'})$ and the clique attached to it (if any), for $i' \in \{1, 2, \cdots, i\}$. Suppose that S is a solution for G, that we are looking for, and $\mathbb{D}' = (X', Y', F', f' : V(X') \to V(F'), g' : \mathcal{C}(G[Y']) \to V(F') \cup \{\perp\})$ is an (η, Clq)-decomposition for $G - S$.[1] For each $i \in [\ell]$, we will maintain a guess for the partition, $(\widehat{X}, \widehat{Y}, \widehat{W})$ of $\beta(p_i)$, where we want $\widehat{W} = S \cap \beta(p_i)$, $\widehat{X} = X' \cap \beta(p_i)$, and $\widehat{Y} = Y' \cap \beta(p_i)$. We will also maintain the information regarding the "structure" of (F', f', g'), when restricted to the vertices in $\beta(p_i) - S$. Although we cannot maintain the whole of F', we will maintain a tuple $\mathcal{F} = (\widehat{F}, \widehat{f} : \widehat{X} \to \widehat{F}, \mathsf{fill} : V(\widehat{F}) \to \{\mathsf{cur}, \mathsf{pbl}, \mathsf{ump}\}, \mathsf{load} : V(\widehat{F}) \to [\eta])$ that will give us the following information. We would like $\widehat{F}$ to correspond to the "truncation" of F', when restricted to: i) the vertices in F' to which vertices in $\widehat{X}$ are mapped, and ii) the vertices in F' that are associated with connected components of $G_i[\widehat{Y}]$ (via g'). In the above not only we will preserve the mappings of $\widehat{X}$ and $\mathcal{C}(G_i[\widehat{Y}])$, but will also maintain the paths of these vertices to the root of the tree in F', containing them. As the depth of F' is at most η, it will be enough to have at most $\mathcal{O}(\eta(k + \eta))$ vertices in our guess $\widehat{F}$. The function $\widehat{f}$ will correspond to f' restricted to the vertices in $\widehat{X}$. Some vertices in F' are filled from above (vertices in $G - V(G_i)$), and thus must remain unmapped (ump, for short), when restricted to G_i. Thus, the vertices in $\widehat{F}$ (if any) of the above type, will be marked ump by the function fill. The vertices of F' to which vertices in $\widehat{X}$ are mapped, will be assigned cur (short for current) by fill, indicating that the these vertices are already used by the current bag $\beta(p_i)$, under consideration. Finally, the remaining vertices in $\widehat{F}$ are free to be potentially used by vertices that appear strictly below, and they are marked pbl (short for potential below) by fill. The function load will indicate the depth of the sub-tree that may be to a vertex in $\widehat{F}$. We will now discuss the properties that we would like to maintain corresponding to the function g'. We will maintain a tuple $\mathcal{G} = (\widehat{g} : \mathcal{C}(G[\widehat{Y}]) \to V(\widehat{F}) \cup \{\perp\}, \mathsf{ext} : \mathcal{C}(G[\widehat{Y}]) \to \{0, 1\})$, which will give us the following information: the function $\widehat{g}$ correspond to the restriction of g' to the connected components containing vertices from $\widehat{Y}$, and ext will indicate whether we can "extend" the connected component by adding vertices to it, that appear strictly below. Finally, we will maintain the guess, $k' \in \mathbb{N}$, for the size of the solution restricted to G_i. We can bound the number of states in the dynamic programming table by $(k + \eta)^{O(\eta k + \eta^2)} n^{\mathcal{O}(1)}$, and we can (recursively) compute each of the table entry in time bounded by $(k + \eta)^{O(\eta k + \eta^2)} n^{\mathcal{O}(1)}$. This gives us an algorithm for the problem, running in time $(k + \eta)^{O(\eta k + \eta^2)} n^{\mathcal{O}(1)}$.

[1] For the section, we will be using a modified (but equivalent) definition of (η, Clq)-decomposition, which will simplify some of the technicalities and arguments.

3.3 Polynomial-time $\mathcal{O}(\mathsf{opt}\log^2 n)$-approximation for (η, Clq)-ED Deletion (Theorem 2)

Our algorithm follows the recursive scheme of [1], for designing polylogarithmic approximation algorithms, that in turn builds upon the classic technique of finding balanced separators in a graph for designing approximation algorithms [26]. Our algorithm has two crucial ingredients: (i) If the graph G has a clique of size $\Omega(n)$, then it also has a clique of size $\Omega(n)$ with a neighborhood of size $\mathcal{O}(\mathsf{opt}+\eta)$. Moreover, in polynomial time, either we can find such a large clique with a small neighborhood, or conclude that G has no large cliques. (ii) If G has no cliques of size $n/3$ and it satisfies certain simple constraints relating opt, k and n then G has a balanced separator (see Definition 6) of size $\mathcal{O}((\mathsf{opt}+\eta)\log n)$ that can be computed efficiently. In the case where G does not satisfy these constraints on opt, k and n, we will have arrived at the base case of our recursion we can obtain a straightfoward approximation.

Using these two ingredients, our algorithm proceeds as follows. We try to compute a clique of size $\Omega(n)$, say Q, with a neighborhood of size $\mathcal{O}(\mathsf{opt}+\eta)$. If we succeed, then we add the neighbors of Q to the solution, remove Q from G, and recurse on the smaller instance. If we fail to find Q, then we may conclude that G has no large cliques. In this case, we compute an $\mathcal{O}(\log n)$-approximate balanced separator [26], add these separator vertices to the solution, and recursively approximate the solutions in the two smaller instances. A standard induction argument on the number of vertices gives the stated bound on the approximation ratio.

3.4 Approximating (η, Clq)-decompositions (Theorem 4)

Consider an instance (G, η) of Clq-Elimination Distance, and let $\mathbb{D} = (X, Y, F, f, g)$ be an (η, Clq)-decomposition of G. The high level idea is to identify an efficiently computable set Z of vertices such that for some F', f', g', $\mathbb{D}' = (Z, V(G) \setminus Z, F', f', g')$ is a $(5\eta, \mathsf{Clq})$-decomposition.

The first step of our algorithm iteratively computes a grouping of vertices of G. Let $\mathcal{S}$ be the set of all maximal cliques in G. From Observation 17, every clique in G either lies on a root-to-leaf path in the interior of $\mathbb{D}$ or in the set $V(C_{\mathsf{ext}})$ for some $C \in \mathcal{C}(G[Y])$. In particular, Observation 17 holds for every set in $\mathcal{S}$. The goal of our grouping is to repeatedly combine pairs of sets in this collection while ensuring that each set in the collection obtained at every step also satisfies the statement of Observation 17.

Specifically, we begin by computing the collection $\mathcal{S}$ of all maximal cliques in the input graph G and then repeatedly do the following: as long as there is a pair of sets in the collection such that either (i) they intersect in at least $\eta+1$ vertices or (ii) the minimum set of vertices that must be deleted to separate them is at least $\eta+1$ or (iii) both sets have size at least $2\eta+1$ and their union can be partitioned into a clique plus at most η vertices, then we remove both sets from the collection and add their union to the collection. We then show that every set in the collection obtained at every step in this procedure also satisfies the statement of Observation 17. Moreover, we show that for every pair of large sets (of size at least $2\eta+1$) that remain in the collection at the end of this iterative procedure, every vertex in their intersection must be contained in the interior of any (η, Clq)-decomposition of G. Therefore, we may safely "push" these vertices (denoted by Z_{nec}) to the interior part of the approximate decomposition that we are constructing. Now, we consider the remaining vertices of the graph and argue that vertices that appear only in small sets (sets of size at most 2η) in our grouping can be safely pushed to the interior part of our approximate (η, Clq)-decomposition without increasing the depth of the interior by more than a constant factor. Denote the vertices pushed to the interior in this way by Z_{sml}. We then need to deal

with vertices that appear in exactly one large set in our grouping. For this, we show that in each large set, the set of vertices that appear only in this large set can be covered by a clique plus a set of at most η vertices. We show that pushing these at most η vertices from each large set into the interior of our decomposition also does not blow up the depth of the interior by more than a constant factor. Once we have pushed these vertices (denoted by Z_{mch}) to the interior, we are left with a disjoint union of cliques. We argue that every set $S \in \mathcal{S}$ can be separated from its out-neighborhood by at most η vertices and moreover, pushing an arbitrary set of such vertices into the interior for each set $S \in \mathcal{S}$ (cumulatively denoted by Z_{sep}) also does not increase the depth of the interior by more than η. Finally, we show that the remaining vertices in X that are not explicitly pushed into the interior by one of our steps can be simply removed from the interior with the result being a $(5\eta, \mathsf{Clq})$-decomposition of G. That is, we prove the following lemma, where $\overset{\rightsquigarrow}{X} = Z_{\mathsf{nec}} \cup Z_{\mathsf{sml}} \cup Z_{\mathsf{mch}} \cup Z_{\mathsf{sep}}$ denotes the set of vertices that we have explicitly pushed into the interior of the new decomposition.

▶ **Lemma 18.** *If there is an (η, Clq)-decomposition $\mathbb{D} = (X, Y, F, f, g)$ of G, then there is an $(\overset{\rightsquigarrow}{\eta}, \mathsf{Clq})$-decomposition $\overset{\rightsquigarrow}{\mathbb{D}} = (\overset{\rightsquigarrow}{X}, \overset{\rightsquigarrow}{Y}, \overset{\rightsquigarrow}{F}, \overset{\rightsquigarrow}{f}, \overset{\rightsquigarrow}{g})$ of G such that $\overset{\rightsquigarrow}{\eta} \leq 5\eta$.*

In order to compute the 5-approximate (η, Clq)-decomposition, we create an appropriate graph, call it G_{full}, on the vertex set $\overset{\rightsquigarrow}{X}$ and show that computing the treedepth exactly (and a forest embedding) of G_{full} is sufficient to obtain a $(5\eta, \mathsf{Clq})$-decomposition of G.

▶ **Lemma 19.** *There is a polynomial-time algorithm that, given a forest embedding $(F_{\mathsf{full}}, f_{\mathsf{full}})$ of G_{full} with depth η', outputs an (η', Clq)-decomposition of G.*

When we use the $2^{\mathcal{O}(\mathsf{td}(G_{\mathsf{full}}) \cdot \mathsf{tw}(G_{\mathsf{full}}))} n^{\mathcal{O}(1)}$ algorithm of Reidl et al. [30] to compute the treedepth of G_{full} and an optimal forest embedding, we obtain a $(5\eta, \mathsf{Clq})$-decomposition of G in time $2^{\mathcal{O}(\eta^2)} n^{\mathcal{O}(1)}$. If we use the polynomial-time algorithm of Czerwinski et al. [8] to approximate the treedepth of G_{full}, we obtain a $(\mathcal{O}(\eta^2 \log^{3/2} \eta), \mathsf{Clq})$-decomposition of G in time $2^{\mathcal{O}(\eta)} n^{\mathcal{O}(1)}$.

4 Discussions and conclusions

We studied the parameterized complexity of detecting a small modulator to graphs that have a constant elimination distance to cluster graphs. For this problem, we obtained a polynomial kernelization and in the process, developed a $k^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$-time fixed-parameter algorithm and a polynomial-time factor-$\mathsf{min}\{\mathcal{O}(\eta \cdot \mathsf{opt} \cdot \log^2 n), \mathsf{opt}^{\mathcal{O}(1)}\}$ approximation algorithm for this problem. Since our focus was on analyzing the kernelization complexity of this problem, we have not attempted to optimize the running time of our algorithm or the exponent of k in the size-bound of the kernel. Moreover, since our focus was on the "recognition" problem for such graphs, we leave for future work the design of fixed-parameter algorithms and kernelization algorithms for other problems, when parameterized by the size of the smallest modulator of the given graph to the class of (η, Clq)-graphs. We remark that even parameterization by the elimination distance of the input graph to cluster graphs has not been explored.

Between $\mathcal{A}_1, \mathcal{A}_2$ and Theorem 3, for the problem of computing the eliminating distance of a graph to cluster graphs, we obtained an exact algorithm (i.e., a 1-approximation) running in time $\eta^{\mathcal{O}(\eta^2)} \cdot n^{\mathcal{O}(1)}$, a 5-approximation algorithm running in time $2^{\mathcal{O}(\eta^2)} \cdot n^{\mathcal{O}(1)}$ and an $\mathcal{O}(\eta \log^{3/2} \eta)$-approximation algorithm running in time $2^{\mathcal{O}(\eta)} \cdot n^{\mathcal{O}(1)}$. An interesting direction for future research is to identify the best possible tradeoffs between approximation factor and running time for the problem. Naturally, exploring the algorithmic utility of (small modulators to) bounded elimination distance to other graph classes remains an interesting line of research.

References

1 Akanksha Agrawal, Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Polylogarithmic approximation algorithms for weighted-f-deletion problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 1:1–1:15, 2018.

2 Marin Bougeret, Bart M. P. Jansen, and Ignasi Sau. Bridge-Depth Characterizes which Structural Parameterizations of Vertex Cover Admit a Polynomial Kernel. *arXiv e-prints*, 2020. `arXiv:2004.12865`.

3 Marin Bougeret and Ignasi Sau. How much does a treedepth modulator help to obtain polynomial kernels beyond sparse graphs? *Algorithmica*, 81(10):4043–4068, 2019.

4 Adam Bouland, Anuj Dawar, and Eryk Kopczynski. On tractable parameterizations of graph isomorphism. In Dimitrios M. Thilikos and Gerhard J. Woeginger, editors, *Parameterized and Exact Computation - 7th International Symposium (IPEC)*, volume 7535, pages 218–230, 2012.

5 Jannis Bulian and Anuj Dawar. Graph isomorphism parameterized by elimination distance to bounded degree. *Algorithmica*, 75(2):363–382, 2016.

6 Jannis Bulian and Anuj Dawar. Fixed-parameter tractable distances to sparse graph classes. *Algorithmica*, 79(1):139–158, 2017.

7 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Science & Business Media, 2015.

8 Wojciech Czerwinski, Wojciech Nadara, and Marcin Pilipczuk. Improved bounds for the excluded-minor approximation of treedepth. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms (ESA)*, volume 144, pages 34:1–34:13, 2019.

9 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

10 Eduard Eiben, Robert Ganian, Thekla Hamm, and O-joung Kwon. Measuring what matters: A hybrid approach to dynamic programming with treewidth. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 138, pages 42:1–42:15, 2019.

11 Eduard Eiben, Robert Ganian, and Stefan Szeider. Meta-kernelization using well-structured modulators. In Thore Husfeldt and Iyad A. Kanj, editors, *10th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 43 of *LIPIcs*, pages 114–126, 2015.

12 Michael R. Fellows, Bart M. P. Jansen, and Frances A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *Eur. J. Comb.*, 34(3):541–566, 2013.

13 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, M. S. Ramanujan, and Saket Saurabh. Solving d-sat via backdoors to small treewidth. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 630–641, 2015.

14 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar $\mathcal{F}$-deletion: Approximation, kernelization and optimal FPT algorithms. In *53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 470–479, 2012.

15 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. *Journal of Computer and System Sciences*, 84:219–242, 2017.

16 Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Going beyond primal treewidth for (M)ILP. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 815–821, 2017.

17 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Combining treewidth and backdoors for CSP. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 66, pages 36:1–36:17, 2017.

18 Archontia C. Giannopoulou, Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. Uniform kernelization complexity of hitting forbidden minors. *ACM Transactions on Algorithms*, 13(3):35:1–35:35, 2017.

19 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne, editors, *Parameterized and Exact Computation, First International Workshop (IWPEC)*, volume 3162, pages 162–173. Springer, 2004.

20 Eva-Maria C. Hols, Stefan Kratsch, and Astrid Pieterse. Elimination distances, blocking sets, and kernels for vertex cover. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 154, pages 36:1–36:14, 2020.

21 Ashwin Jacob, Fahad Panolan, Venkatesh Raman, and Vibha Sahlot. Structural parameterizations with modulator oblivion, 2020. `arXiv:2002.09972`.

22 Bart M. P. Jansen and Marcin Pilipczuk. Approximation and kernelization for chordal vertex deletion. *SIAM J. Discrete Math.*, 32(3):2258–2301, 2018.

23 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

24 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Transactions on Algorithms*, 12(2):21:1–21:41, 2016.

25 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

26 Frank Thomson Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

27 Alexander Lindermayr, Sebastian Siebertz, and Alexandre Vigny. Elimination distance to bounded degree on planar graphs. In *45th International Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 170 of *LIPIcs*, pages 65:1–65:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

28 Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. In *9th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 260–272, 2004.

29 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

30 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In *Automata, Languages, and Programming - 41st International Colloquium (ICALP)*, pages 931–942, 2014.

31 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, (STOC)*, pages 887–898, 2012.

Component Order Connectivity in Directed Graphs

Jørgen Bang-Jensen
University of Southern Denmark, Odense, Denmark
jbj@imada.sdu.dk

Eduard Eiben
Royal Holloway, University of London, UK
eduard.eiben@rhul.ac.uk

Gregory Gutin
Royal Holloway, University of London, UK
g.gutin@rhul.ac.uk

Magnus Wahlström
Royal Holloway, University of London, UK
Magnus.Wahlstrom@rhul.ac.uk

Anders Yeo
University of Southern Denmark, Odense, Denmark
andersyeo@gmail.com

Abstract

A directed graph D is semicomplete if for every pair x, y of vertices of D, there is at least one arc between x and y. Thus, a tournament is a semicomplete digraph. In the Directed Component Order Connectivity (DCOC) problem, given a digraph $D = (V, A)$ and a pair of natural numbers k and ℓ, we are to decide whether there is a subset X of V of size k such that the largest strong connectivity component in $D - X$ has at most ℓ vertices. Note that DCOC reduces to the Directed Feedback Vertex Set problem for $\ell = 1$. We study parameterized complexity of DCOC for general and semicomplete digraphs with the following parameters: $k, \ell, \ell + k$ and $n - \ell$. In particular, we prove that DCOC with parameter k on semicomplete digraphs can be solved in time $O^*(2^{16k})$ but not in time $O^*(2^{o(k)})$ unless the Exponential Time Hypothesis (ETH) fails. The upper bound $O^*(2^{16k})$ implies the upper bound $O^*(2^{16(n-\ell)})$ for the parameter $n - \ell$. We complement the latter by showing that there is no algorithm of time complexity $O^*(2^{o(n-\ell)})$ unless ETH fails. Finally, we improve (in dependency on ℓ) the upper bound of Göke, Marx and Mnich (2019) for the time complexity of DCOC with parameter $\ell + k$ on general digraphs from $O^*(2^{O(k\ell \log(k\ell))})$ to $O^*(2^{O(k \log(k\ell))})$. Note that Drange, Dregi and van 't Hof (2016) proved that even for the undirected version of DCOC on split graphs there is no algorithm of running time $O^*(2^{o(k \log \ell)})$ unless ETH fails and it is a long-standing problem to decide whether Directed Feedback Vertex Set admits an algorithm of time complexity $O^*(2^{o(k \log k)})$.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized Algorithms, component order connectivity, directed graphs, semicomplete digraphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.2

Funding *Jørgen Bang-Jensen*: Research supported by the Independent Research Fund Denmark under grant number DFF 7014-00037B.
Gregory Gutin: Research supported by the Leverhulme Trust under grant number RPG-2018-161.

1 Introduction

Motivated by various practical network applications, many different vulnerability measures of undirected graphs have been introduced and studied in the literature. The two most studied of such measures are vertex and edge connectivity of an undirected graph. However, these two measures often do not capture the more subtle vulnerability properties of networks that one might wish to consider, such as the number of vertices in the largest remaining connected component.

While both undirected and directed graphs are of great interest in graph theory, algorithms and applications, undirected graphs have been studied much more than their directed counterparts arguably due to simpler structure of undirected graphs. In this paper, we study a number of parameterizations of a problem of interest from both theory and applications which was mainly studied for undirected graphs so far.

In many networks, the underlying graph is directed rather than undirected and the aim of this paper is to study an extension to directed graphs of the **ℓ-component order connectivity** of an undirected graph G, which is the size of a minimum set $X \subseteq V(G)$ such that $\mathrm{mco}(G - X) \leq \ell$, where $\mathrm{mco}(G - X)$ is the number of vertices in the largest connected component of $G - X$ (mco stands for maximum component order). By Component Order Connectivity we will denote the following decision problem:

component order connectivity
Input: A graph $G = (V, E)$ and a pair $\ell, k \in \mathbb{N}$ of natural numbers
Question: Is there a subset X of V of size k such that $\mathrm{mco}(G - X) \leq \ell$?

For a survey on Component Order Connectivity, see Gross et al. [13]; for more recent research on the problem, see e.g. [11, 15, 16].

For a directed graph D, we define the **ℓ-component order connectivity** as the size of a minimum set $X \subseteq V(D)$ such that $\mathrm{mco}(D - X) \leq \ell$, where $\mathrm{mco}(D - X)$ is the number of vertices in the largest strongly connected component of $D - X$. Using this definition of $\mathrm{mco}(D - X)$, we can state the following directed version of Component Order Connectivity.

directed component order connectivity
Input: A digraph $D = (V, A)$ and a pair $\ell, k \in \mathbb{N}$ of natural numbers
Question: Is there a subset X of V of size k such that $\mathrm{mco}(D - X) \leq \ell$?

In what follows, we will assume without loss of generality that $k + \ell < n = |V|$ (or, $k < n - \ell$). Indeed, if $k + \ell \geq n$ then our instance is a YES-instance since deleting any set X of k vertices implies $\mathrm{mco}(D - X) \leq \ell$.

Clearly, Directed Component Order Connectivity is a generalization of Component Order Connectivity (each instance (G, ℓ, k) of Component Order Connectivity corresponds to an equivalent instance (D, ℓ, k) of Directed Component Order Connectivity, where D is obtained from G by replacing every edge of G by a directed 2-cycle). For $\ell = 1$, while Component Order Connectivity is equivalent to the Vertex Cover problem, Directed Component Order Connectivity is equivalent to the Directed Feedback Vertex Set problem. Unlike Vertex Cover whose fixed-parameter tractability is very easy to show, a fact that was known very early on in parameterized algorithmics [9], fixed-parameter tractability of Directed Feedback Vertex Set was a long-standing open problem until Chen et al. [7] in 2008 proved its fixed-parameter tractability by designing a $4^k k! n^{\mathcal{O}(1)}$-time algorithm. (We provide basics on parameterized algorithms and complexity in the next section.)

Since Component Order Connectivity is NP-complete (it remains NP-complete even for split, co-bipartite and chordal undirected graphs [11]), a number of researchers studied Component Order Connectivity using the framework of parameterized algorithmics, see e.g. [11, 15, 16]. Göke, Marx and Mnich [12] were the first to study the Directed Component Order Connectivity problem from the viewpoint of parameterized algorithms and complexity. They obtained an algorithm of running time $4^k(k\ell+k+\ell)!n^{\mathcal{O}(1)}$, which is close to the complexity of the algorithm of Chen et al. [7] when $\ell = 1$. Thus, Directed Component Order Connectivity parameterized by $k+\ell$ is fixed-parameter tractable (FPT).

We will continue the study of Directed Component Order Connectivity using parameterized algorithms and complexity. In particular, as in papers [11, 15, 16] which studied Component Order Connectivity, we study Directed Component Order Connectivity parameterized by three parameters: ℓ, k and $\ell+k$. We will denote the corresponding parameterized problems by Directed Component Order Connectivity$[\ell]$, Directed Component Order Connectivity$[k]$ and Directed Component Order Connectivity$[\ell+k]$, respectively.

Moreover, we introduce and study a new parameterization of Directed Component Order Connectivity: parameter $n-\ell$, where n is the number of vertices in D. One reason to introduce Directed Component Order Connectivity$[n-\ell]$ is that normally one requires the parameters to be relatively small compaired to the size of the problem under consideration. However, if k is small it is possible that for every $X \subseteq V(D)$ of size k, $\mathrm{mco}(D-X)$ is not much smaller than $n-k$. Then $n-\ell$ can be much smaller than ℓ.

Since Component Order Connectivity is equivalent to the Vertex Cover problem for $\ell = 1$, Component Order Connectivity$[\ell]$ is para-NP-complete. Drange et al. [11, Theorem 8] proved that Component Order Connectivity$[k]$ is W[1]-hard even on split graphs. In their construction, $n-\ell = \mathcal{O}(k^2)$. Hence, Component Order Connectivity$[n-\ell]$ is also W[1]-hard. They also showed that Component Order Connectivity$[\ell+k]$ is FPT by obtaining an algorithm of running time $2^{\mathcal{O}(k\log\ell)}n$. The above mentioned results are written in the undirected graphs row of Table 1.

A directed graph D is **semicomplete** if for every pair x, y of distinct vertices of D, there is an arc between x and y. When we require that there is only one arc between x and y then we obtain a definition of a **tournament**. Clearly, the hardness results for the directed graphs row of Table 1 follow from the corresponding results in the undirected graphs row for columns $n-\ell$ and k. Directed Component Order Connectivity$[\ell]$ is para-NP-complete for semicomplete digraphs as Directed Component Order Connectivity on semicomplete digraphs is NP-complete for $\ell = 1$. This follows from the fact that Directed Feedback Vertex Set is NP-complete even for tournaments, as proved by Bang-Jensen and Thomassen [3] and Speckenmeyer [18].

The FPT result in the directed graphs row of Table 1 is first obtained by Göke et al. [12] as discussed above. The running time of their algorithm is $4^k(k\ell+k+\ell)!n^{\mathcal{O}(1)} = 2^{\mathcal{O}(k\ell\log(k\ell))}n^{\mathcal{O}(1)}$. By modifying their algorithm, we obtained an algorithm of complexity $2^{\mathcal{O}(k)}\ell^k k!n^{\mathcal{O}(1)} = 2^{\mathcal{O}(k\log(k\ell))}n^{\mathcal{O}(1)}$, which decreases asymptotic dependence of the running time on ℓ.[1] Our modification consists of replacing a branching algorithm in [12] with a randomized algorithm which can be derandomized without increasing the complexity upper bound. Note that Drange et al. [11, Theorem 14] proved that even for Component Order

[1] The same result was also obtained in [17]. We obtained this result independently and our approach is different from that in [17].

Connectivity on split graphs there is no algorithm of running time $O^*(2^{o(k \log \ell)})$ (here we assume that $\ell = O(k^{O(1)})$) unless the Exponential Time Hypothesis (ETH) [14] fails and it is a long-standing problem to decide whether Directed Feedback Vertex Set admits an algorithm of time complexity $O^*(2^{o(k \log k)})$.

Table 1 Parameterized Complexity of (Directed) Component Order Connectivity.

class of graphs	$n - \ell$	k	ℓ	$\ell + k$
semicomplete digraphs	FPT	FPT	para-NP-c.	FPT
undirected graphs	W[1]-hard	W[1]-hard	para-NP-c.	FPT
directed graphs	W[1]-hard	W[1]-hard	para-NP-c.	FPT

The most interesting entry in the semicomplete digraphs row is a non-trivial result that Directed Component Order Connectivity$[k]$ on semicomplete digraphs is FPT. This FPT algorithm boils down to finding a shortest path in a suitably defined auxiliary weighted acyclic digraph. The running time of the algorithm is $\mathcal{O}(2^{16k}kn^2)$. The other two FPT entries in this row follow from this result (for the parameter $n - \ell$ this is due to our assumption that $k < n - \ell$). We also prove the following lower bounds: no algorithm for Directed Component Order Connectivity$[k]$ on semicomplete digraphs can have time complexity $2^{o(k)}n^{\mathcal{O}(1)}$ unless ETH fails[2] and no such deterministic algorithm can run in time $o(n^2)$.

Our paper is organised as follows. The next section is devoted to terminology and notation on directed and undirected graphs, and basics on parameterized algorithms and complexity. In Section 3, we describe our improvement on the algorithm of Göke et al. [12]. In Section 4, we prove that Directed Component Order Connectivity$[k]$ on semicomplete digraphs admits an algorithm of running time $\mathcal{O}^*(2^{16k})$ and show the lower bounds on running time with parameters k and $n - \ell$. We conclude the paper in Section 5.

2 Preliminaries

2.1 Directed and Undirected Graph Terminology and Notation

In this paper, all directed and undirected graphs are finite, without loops or parallel edges. As often the case in the directed graph theory, an edge of a digraph will be called an **arc** and the vertex and arc sets of a digraph D will be denoted by $V(D)$ and $A(D)$, respectively. The **out-neighbourhood** and **in-neighbourhood** of a vertex x of a digraph D are denoted by $N_D^+(x) = \{y \in V(D) : \ xy \in A(D)\}$ and $N_D^-(x) = \{y \in V(D) : \ yx \in A(D)\}$, respectively, and the subscript D will be omitted if D is clear from the context. The **out-degree** and **in-degree** of a vertex x of D is $d_D^+(x) = |N_D^+(x)|$ and $d_D^-(x) = |N_D^-(x)|$, respectively.

In this paper all paths and cycles in digraphs are directed, so we will omit the adjective "directed" when referring to paths and cycles in digraphs. If $D = (V, A)$ is a digraph and $S \subseteq V$, then we denote by $D[S]$ the subdigraph induced by the vertices in S. A digraph D is **strongly connected** (or, just **strong**) if there is a path from x to y for every ordered pair x, y of distinct vertices. A **strong component** of a digraph D is a maximal strong induced subgraph of D. Strong components of D do not share vertices and can be ordered $D_1, D_2, \ldots, D_p$ such that there is no arc in D from $V(D_j)$ to $V(D_i)$ when $j > i$. Such an

[2] Similarly, no algorithm for Directed Component Order Connectivity$[n - \ell]$ on semicomplete digraphs can have running time $2^{o(n-\ell)}n^{\mathcal{O}(1)}$, unless ETH fails.

ordering is called an **acyclic ordering**. Note that if D is a semicomplete digraph, then the strong components of D have a unique acyclic ordering $D_1, D_2, \ldots, D_p$ and we have $xy \in A(D)$ for every $x \in V(D_i),\ y \in V(D_j),\ i < j$.

Basic digraph terminology not introduced in this section can be found in [1, 2].

2.2 Parameterized Complexity

An instance of a parameterized problem Π is a pair (I, k) where I is the **main part** and k is the **parameter**; the latter is usually a non-negative integer. A parameterized problem is **fixed-parameter tractable** (FPT) if there exists a computable function f such that instances (I, k) can be solved in time $\mathcal{O}(f(k)|I|^c)$ where $|I|$ denotes the size of I and c is an absolute constant. The class of all fixed-parameter tractable decision problems is called FPT and algorithms which run in the time specified above are called FPT algorithms. As in other literature on FPT algorithms, we will sometimes omit the polynomial factor in $\mathcal{O}(f(k)|I|^c)$ and write $\mathcal{O}^*(f(k))$ instead.

While FPT is a parameterized complexity analog of P in classic complexity, there are many hardness classes in parameterized complexity and they form a nested sequence starting from W[1]. It is well known that if the Exponential Time Hypothesis holds then FPT$\neq$W[1]. Due to this and other complexity results, it is widely believed that FPT$\neq$W[1] and hence W[1] is viewed as a parameterized analog of NP in classical complexity.

para-NP is the class of parameterized problems which can be solved by a nondeterministic algorithm in time $\mathcal{O}(f(k)|I|^c)$, where f is a computable function and c is an absolute constant. It is well-known that if a problem Π with parameter κ is NP-hard when κ equals to some constant, then Π is para-NP-hard. It is also well known that FPT=para-NP if and only if P=NP.

For more information on parameterized algorithms and complexity, see recent books [8, 10].

3 Directed Component Order Connectivity[$\ell + k$] on General Digraphs

Göke, Marx and Mnich [12] showed that Directed Component Order Connectivity[$\ell + k$] is FPT with a running time given as

$$4^k(k\ell + k + \ell)!n^{\mathcal{O}(1)} = 2^{\mathcal{O}(k\ell \log(k\ell))}n^{\mathcal{O}(1)}.$$

The core of their algorithm is as follows. Begin with the *iterative compression* version of the problem, where in addition to (D, ℓ, k) the input also contains a solution X_0 with $|X_0| = k+1$, which can be used to guide the search for a smaller solution. This is a standard ingredient in FPT algorithms; see, e.g., [8]. At the cost of a simple branching step, we may also assume that we are looking for a solution X with $X \cap X_0 = \emptyset$. Next, they observe that *if* we knew the strongly connected components of $D - X$ that the vertices of X_0 are contained in, then the problem reduces to a previously studied, simpler problem known as Skew Separator [7], which occurs in the design of the FPT algorithm for Directed Feedback Vertex Set (DFVS) of Chen et al. [7]. Indeed, if the precise strong components containing the vertices of X_0 are known, then the problem can be solved in time $O^*(4^k k!)$ using a strategy much like that for DFVS [7, 12]. Hence the bottleneck in Directed Component Order Connectivity[$\ell + k$] is the guessing of the strong components of X_0 in $D - X$.

Göke et al. [12] solve this via a branching algorithm that they analyse as taking time at most $(k\ell + k + \ell)!$. We show a simpler randomized method solving this problem with an improved time bound of

$$\binom{\ell(k+1)+k}{k} \leq (e(\ell+1+\ell/k))^k \leq (3e\ell)^k = 2^{\mathcal{O}(k)} \cdot \ell^k. \tag{1}$$

The method can be derandomized by standard means.

▶ **Lemma 1.** *Let (D, ℓ, k) be an instance of* Directed Component Order Connectivity$[\ell + k]$*, and let X_0 be a solution with $|X_0| = k + 1$. Let X be an unknown solution with $|X| \leq k$ such that $X \cap X_0 = \emptyset$. There is a randomized procedure that with success probability at least*

$$\left((\ell+k)^{\mathcal{O}(1)}\binom{\ell k+\ell+k}{k}\right)^{-1}$$

computes a set $S \subset V(D)$ such that for every $x \in X_0$, the strong components containing x in $D - X$ and in $D[S]$ are identical.

Proof. Initialize $S = X_0$, then for every vertex $v \in V(D) \setminus X_0$ place v in S independently at random with probability $p = 1 - 1/(\ell + 1)$. We declare a guess a *success* if the following conditions apply:

1. For every $x \in X_0$ we have $V_x \subseteq S$, where $V_x \subseteq V$ is the strong component of $D - X$ containing x
2. $X \cap S = \emptyset$

Let $Y = \bigcup_{x \in X_0} V_x$. Our guess is successful if and only if $v \in S$ for every $v \in Y$, and $v \notin S$ for every $v \in X$. Since these are independent events, this clearly happens with probability precisely

$$p^{|Y|}(1-p)^{|X|} \geq p^{\ell(k+1)}(1-p)^k,$$

hence the worst case occurs when all sets V_x are disjoint and have $|V_x| = \ell$, and $|X| = k$, i.e., $|Y| = \ell(k + 1)$ and $|X| = k$. Let $S_0 = X \cup Y$. We bound the probability of success carefully in two steps:

1. We estimate the probability that $|S \cap S_0| = |Y|$, *without* caring about the precise intersection (i.e., success in this stage includes cases where $X \cap S \neq \emptyset$).
2. We estimate the probability of success, conditional on the previous event.

Note $|S_0| = \ell(k + 1) + k$ by assumption.

For the first step, note that the expected number of vertices of S_0 *not* in S is

$$(1-p)|S_0| = (1/(\ell+1))(\ell k + \ell + k) = k + \frac{\ell}{\ell+1}.$$

Also note that in a successful guess, this value is precisely k. Hence the expected value differs from the intended value by less than 1. Since $|S \cap S_0|$ is a binomial distribution, due to the guesses being independent, this clearly happens with probability at least inverse polynomial in $k + \ell$.

Subject to this event, the set $S_0 \setminus S$ is uniformly distributed among all subsets of S_0 of size k by independence, hence the conditional probability of success is one in $\binom{\ell k+\ell+k}{k}$. We conclude that the success probability matches the bound in the lemma.

Finally, assume that the guess was successful for some set S and consider the strong component of x in $D[S]$ for some $x \in X_0$. Let V'_x be this strong component. Since $D[V_x]$ is strongly connected and $V_x \subseteq S$, we have $V_x \subseteq V'_x$. On the other hand, by assumption $D[S]$ is an induced subgraph of $D - X$, and since V_x is a strongly connected component in $D - X$ we must have $V'_x \subseteq V_x$. We conclude $V_x = V'_x$ for each $x \in X_0$, as required. ◀

For the derandomization, we employ a **cover-free family** construction of Bshouty and Gabizon [4]. We get the following:

▶ **Lemma 2.** *There is a deterministic procedure that produces a set $\mathcal{F} \subseteq 2^V$ with*

$$|\mathcal{F}| = \binom{\ell k + \ell + k}{k}^{1+o(1)} \log |V|$$

in time $\mathcal{O}(|\mathcal{F}|n)$, such that there is a set $S \in \mathcal{F}$ such that for every $x \in X_0$, the strong components containing x in $D - X$ and in $D[S]$ are identical.

Proof. Let $r \leq s < n$ be integers. Bshouty and Gabizon (in a slightly non-standard definition) define an $(n, (r, s))$**-cover free family** as a set $\mathcal{F} \subseteq \{0, 1\}^n$ such that for every disjoint pair of sets $A, B \subseteq [n]$ with $|A| = r$ and $|B| = s$ there is a set $S \in \mathcal{F}$ such that $A \subseteq S$ and $B \cap S = \emptyset$. Bshouty and Gabizon [4] show how to compute an $(n, (r, s))$-cover free family $\mathcal{F}$ of size

$$|\mathcal{F}| = \binom{r + s}{r}^{1+o(1)} \log n$$

in time $\mathcal{O}(|\mathcal{F}|n)$.

By Lemma 1, it suffices to construct a cover-free family with parameters $n = |V(D)|$, $r = \ell(k + 1)$ and $s = k$. Here $r > s$, but we can simply compute an $(n, (s, r))$-cover free family and take the complement of every member. Hence we get a family of size

$$\binom{\ell k + \ell + k}{k}^{1+o(1)} \log n$$

computed in output-linear time. ◀

The two lemmas of this section and (1) imply the following:

▶ **Theorem 3.** *There is a randomized FPT algorithm that solves* Directed Component Order Connectivity$[\ell + k]$ *in time $2^{\mathcal{O}(k)}\ell^k k! n^{\mathcal{O}(1)}$ with probability at least $\Omega(1)$. The algorithm can be derandomized in the same time, up to a lower-order overhead factor.*

4 Directed Component Order Connectivity on Semicomplete Digraphs

Let us first summarize the main ideas behind our FPT algorithm, before providing more technical details. Let $D = (V, A)$ be a semicomplete digraph, $k, \ell \in \mathbb{N}$ and let $X \subseteq V$ of size k such that $\text{mco}(D - X) \leq \ell$. The vertices of $D - X$ can be partitioned into $C_1, \ldots, C_q$ such that each C_i is the vertex set of a strong component of $D - X$ and

1. for every $i \in [q]$ is $|C_i| \leq \ell$, and
2. for every $i, j \in [q]$ with $i < j$ and every $x \in C_i$, $y \in C_j$ we have $xy \in A$ and $yx \notin A$.

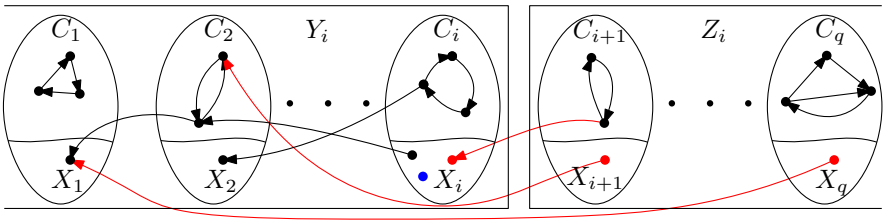


Figure 1 An example of a valid triple (Y_i, Z_i, S_i). A semicomplete digraph D, the set $X = \bigcup_{i\in[q]} X_i$ is such that $\mathrm{mco}(D - X) = 3$ and $C_1, \ldots, C_q$ are strong components of $D - X$. $Y_i = C'_1 \cup C'_2 \cup \cdots \cup C'_i$ and $Z_i = C'_{i+1} \cup C'_{i+2} \cup \cdots \cup C'_q$, where $C'_i = C_i \cup X_i$, $i \in [q]$. The arcs uv, $u \in C'_i$, $v \in C'_j$ for $i < j$ are omitted as well as the arcs within X between X_t and C_t, $t \in [q]$. The set S_i is the set of the three red vertices, one in each of X_i, X_{i+1}, and X_q, is a minimal vertex cover of the red arcs from Z_i to Y_i. Note that the vertex in X_1 is not in S_i as the arc incident to it with the tail in Z_i is already covered by S_i. Note also the blue vertex in X_i, the only reason it is in X is to reduce the size of C_i and as such it will not appear in any S_j, $j \in [q]$, in the set of q valid triples defining these components.

In our algorithm, we would like to discover the strong components one by one in the ascending order from C_1 to C_q. Now let $X_1, \ldots, X_q$ be a partition of X into q (possibly empty) parts and let, for each $i \in [q]$, $Y_i = C'_1 \cup C'_2 \cup \cdots \cup C'_i$ and $Z_i = C'_{i+1} \cup C'_{i+2} \cup \cdots \cup C'_q$, where $C'_i = C_i \cup X_i$, $i \in [q]$. Moreover, let S_i be a subset of X such that for each $y \in Y_i \setminus S_i$ and $z \in Z_i \setminus S_i$ we have $yz \in A$ and $zy \notin A$. See also Figure 1. Note that, given S_i, it suffice to solve our problem in subgraphs $D[Y_i \setminus S_i]$ and $D[Z_i \setminus S_i]$ separately. Moreover, the set $(Y_{i+1} \setminus Y_i) \setminus (S_{i+1} \cup S_i)$ is basically the strong component C_{i+1} up to few vertices in X_{i+1} that are not incident to any arc with tail in $Z_{i+1} \setminus S_{i+1}$ or head in $Y_i \setminus S_i$. Such vertices can actually be replaced in X by any vertex in C_{i+1}. It follows, that if we are given $(Y_1, Z_1, S_1), \ldots, (Y_q, Z_q, S_q)$, then we can easily reconstruct a solution of size $|X|$ as $\bigcup_{i\in[q]} S_i$ plus some arbitrary vertices of $(Y_{i+1} \setminus Y_i) \setminus (S_{i+1} \cup S_i)$ to have at most ℓ vertices in each strong component of $D - X$.

Therefore, our goal will be to search for triples (Y_i, Z_i, S_i), $i \in [q]$, where $\{Y_i, Z_i\}$ is a partition of V and S_i is a minimal subset of X such that there is no arc zy in A with $z \in Z_i \setminus S_i$ and $y \in Y_i \setminus S_i$. The first step of our proof is to show that there are at most $2^{8k+2}n$ triples we need to consider (Lemma 7). We will call these important triples **valid** and we postpone the precise definition for later. The main reason for the bound is that we only need to consider triples (Y_i, Z_i, S_i) for which $|S_i| \leq k$ and that if we fix $|Y_i|$ (and hence also $|Z_i|$), then vertices with out-degree at least $|Z_i| + |S_i| + 1$ (resp. in-degree at least $|Y_i| + |S_i| + 1$) have to be in Y_i (resp. in Z_i) or in S_i and we can fix these vertices in Y_i (resp. in Z_i). Once we bound the number of the triples we need to consider, we can define **compatible** pairs of triples $\big((Y^1, Z^1, S^1), (Y^2, Z^2, S^2)\big)$, for which $Y^1 \subset Y^2$ and these triples, loosely speaking can define a strong component of $D - X$ with at most ℓ vertices as $(Y^2 \setminus Y^1) \setminus (S^1 \cup S^2)$ and the arcs from Z_2 to Y_1 are all hit by a vertex in $S^1 \cap S^2$. This allows us to create an auxiliary acyclic "state" digraph whose vertices are valid triples and arcs are the compatible pairs of triples. The paths from $(\emptyset, V, \emptyset)$ to $(V, \emptyset, \emptyset)$ in this graph then define a solution for (D, ℓ, k). Note that our algorithm can be equivalently seen as a dynamic programming which computes for each valid triple (Y, Z, S) a minimum size set X such that $\mathrm{mco}(D[Y] - (X \cup S)) \leq \ell$.

The following lemma allows us to show that if we fix $|Y|$ in a triple (Y, Z, S), then only $\mathcal{O}(k)$ vertices of D could potentially be in both Y and Z and all other vertices are fixed. The lemma is an easy consequence of the fact that every semicomplete digraph on at least $2p + 2$, $p \in \mathbb{N}$, vertices has a vertex of out-degree at least $p + 1$. We give the proof here for the convenience of the reader.

▶ **Lemma 4.** *Let $D = (V, A)$ be a semicomplete digraph and let Y, Z be a partition of V such that for every $y \in Y$ and every $z \in Z$, we have $yz \in A$. Then for every $p \in \mathbb{N}$ (1) there are at most $2p + 1$ vertices in Y with $d^+_D(y) \leq |Z| + p$ and (2) there are at most $2p + 1$ vertices in Z with $d^-_D(z) \leq |Y| + p$.*

Proof. We will first prove Part (1). Let $Y_\leq$ be the set of vertices in Y with out-degree at most $|Z| + p$ in D. Since for every $y \in Y$ and every $z \in Z$, we have $yz \in A$, it follows that all vertices in $Y_\leq$ have out-degree at most p in $D[Y_\leq]$. Hence $\sum_{y \in Y_\leq} d^+_{D[Y_\leq]}(y)$, *i.e.*, the sum of out-degrees of vertices in $Y_\leq$ in $D[Y_\leq]$, is at most $p|Y_\leq|$. Hence,

$$\sum_{y \in Y_\leq} d^+_{D[Y_\leq]}(y) = |A(D[Y_\leq])| \leq p|Y_\leq|.$$

Since D is a semicomplete digraph,

$$\frac{|Y_\leq| \cdot (|Y_\leq| - 1)}{2} \leq |A(D[Y_\leq])| \leq p|Y_\leq|.$$

It follows that $|Y_\leq| \leq 2p + 1$. Part (2) follows directly from Part (1) applied to a digraph $D' = (V, A')$ obtained from D by reversing all the arcs i.e. $A' = \{yx \mid xy \in A\}$. ◀

Let $D = (V, A)$ be a semicomplete digraph and $t \in [n]$. We will call a triple (Y, Z, S) **t-valid** if

1. Y, Z is a partition of $V(D)$ with $|Y| = t$,
2. $S \subseteq V(D)$ is a minimal (w.r.t. inclusion) set such that for all $y \in Y$ and $z \in Z$, if $zy \in A(D)$, then $|\{y, z\} \cap S| \geq 1$,
3. $|S| \leq k$,
4. for all $x \in S$, if $d^+_D(x) > n - t + k$, then $x \in Y$,
5. for all $x \in S$, if $d^-_D(x) > t + k$, then $x \in Z$.

We will say a triple (Y, Z, S) is **valid**, if it is t-valid for some $t \in \mathbb{N}$. The following simple observation will help us bound the number of partitions (Y, Z) that could lead to a t-valid triple (Y, Z, S).

▶ **Lemma 5.** *For any t-valid triple (Y, Z, S), all vertices v with $d^+_D(v) > n - t + k$ are in Y and all vertices v with $d^-_D(v) > t + k$ are in Z.*

Proof. If $v \in S$, the lemma follows directly from the definition of a t-valid triple. If $v \in V(D) \setminus S$ and $d^+_D(v) > n - t + k$, then v has an out-neighbour in $Y \setminus S$, because $|Z \cup S| \leq n - t + k$, and $v \in Y$ follows by property 2. Similarly, if $v \in V(D) \setminus S$ and $d^-_D(v) > t + k$, then v has an in-neighbour in $Z \setminus S$ and $v \in Z$ by property 2. ◀

▶ **Lemma 6.** *Let $D = (V, A)$ be a semicomplete digraph, $n = |V|$, and let $t \in [n]$. If there exists a t-valid triple, then there are at most $7k + 2$ vertices v in $V(D)$ with $d^+_D(v) \leq n - t + k$ and $d^-_D(v) \leq t + k$.*

Proof. Let us assume that there is at least one t-valid triple and let us denote it (Y, Z, S). Note that for all $y \in Y \setminus S$ and $z \in Z \setminus S$ it holds that $zy \notin A(D)$. Since D is a semicomplete digraph, it follows that $yz \in A(D)$. Due to Lemma 4 applied to $D - S$, there are at most $2(k + |Z \cap S|) + 1$ vertices in $Y \setminus S$ with $d^+_{D-S}(y) \leq |Z \setminus S| + k + |Z \cap S| = n - t + k$ and there are at most $2(k + |Y \cap S|) + 1$ vertices in $Z \setminus S$ with $d^-_{D-S}(z) \leq |Y \setminus S| + k + |Y \cap S| = t + k$. Let $F = \{v \in V(D) : d^+_D(v) \leq n - t + k \text{ and } d^-_D(v) \leq t + k\}$. By the above,

$$\begin{aligned} |F \setminus S| &\leq 2(k + |Z \cap S|) + 1 + 2(k + |Y \cap S|) + 1 \\ &\leq 4k + 2 + 2|S| \leq 6k + 2. \end{aligned}$$

Thus, $|F| \leq 7k + 2$. ◀

▶ **Lemma 7.** *Let $D = (V, A)$ be a semicomplete digraph, $n = |V|$, and let $t \in [n]$. There are at most 2^{8k+2} t-valid triples (Y, Z, S). Moreover, if we are given the in- and out-degrees of all vertices in D on the input, then we can enumerate all such triples in time $\mathcal{O}(2^{8k}kn)$.*

Proof. Let $F = \{v \in V(D) : d^+_D(v) \leq n - t + k \text{ and } d^-_D(v) \leq t + k\}$. By Lemma 6, $|F| \leq 7k + 2$. If the out- and in-degrees of all vertices in D are given on the input, we can construct the set F in time $\mathcal{O}(n)$. By Lemma 5, there are at most 2^{7k+2} possible partitions (Y', Z') that could lead to a t-valid triple (Y', Z', S') for some S', each such partition is uniquely determinate by fixing $Y' \cap F$.

For the rest of the proof, we assume that we computed the set F of vertices v in $V(D)$ with $d^+_D(v) \leq n - t + k$ and $d^-_D(v) \leq t + k$, $|F| \leq 7k + 2$. Let (Y', Z') be one of 2^{7k+2} partitions that could lead to a t-valid triple. We show that we can enumerate all minimal sets S', $|S'| \leq k$, such that for all $y \in Y'$ and $z \in Z'$, if $zy \in A(D)$, then $|\{y, z\} \cap S'| \geq 1$. Let G be an undirected bipartite graph such that $V(G) = V(D)$, the partite sets of G are Y' and Z', and for every $y \in Y'$, $z \in Z'$, it holds $yz \in V(G)$ if and only if $zy \in A(D)$. Then S' is a minimal vertex cover in G. Moreover, every minimal vertex cover S' in G leads to a t-valid triple (Y', Z', S'). It is well known and easy to show that we can enumerate all minimal vertex covers of size at most k in G in time $\mathcal{O}(2^k k^2 + kn)$. This is done by including all vertices with degree at least $k + 1$ in every vertex coverand removing every vertex they cover. If the resulting graph has more than k^2 edges, then there is no vertex cover of size at most k [5]. Then we can enumerate all vertex covers, by using simple search-tree algorithm that picks an edge, say uv, and recursively enumerate all minimal vertex covers of size at most $k - 1$ that include u or v, respectively. Given the algorithm, it is also easy to see that there are at most 2^k distinct minimal vertex covers of size at most k.

It follows that there are at most $2^{7k+2} \cdot 2^k = 2^{8k+2}$ t-valid triples and we can enumerate all of them in time $\mathcal{O}(n + 2^{8k}k^2 + kn) = \mathcal{O}(2^{8k}kn)$. ◀

We are now ready to present our algorithm.

▶ **Theorem 8.** *There is an FPT algorithm that solves* Directed Component Order Connectivity$[k]$ *on semicomplete digraphs in time $\mathcal{O}(2^{16k}kn^2)$.*

Proof. Let $D = (V, A)$ be a semicomplete digraph and let (D, ℓ, k) be an instance of Directed Component Order Connectivity$[k]$.

Algorithm. Our algorithm boils down to finding a shortest path in an auxiliary weighted acyclic digraph whose vertex set consists of all the valid triples. The main idea is to find a sequence of valid triples $(Y_1, Z_1, S_1), \ldots, (Y_q, Z_q, S_q)$ such that $S = \bigcup_{i \in [q]} S_i$ is a solution for (D, ℓ, k) and the strongly connected components of $D - S$ are subsets of $C_i = Y_{i+1} \setminus (Y_i \cup S)$, where $|C_i| \leq \ell$ and for all $i < j$, $x_i \in C_i$, $x_j \in C_j$ it holds that $x_j x_i \notin A$.

We define the weighted directed acyclic state graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ as follows. The set of vertices $\mathcal{V}$ is the set of all t-valid triples for all $t \in \{0, 1, \ldots, n\}$. The set of arcs $\mathcal{A}$ contains an arc from a t_1-valid triple (Y_1, Z_1, S_1) to a t_2-valid triple (Y_2, Z_2, S_2) if and only if the following conditions holds:

- $Y_1 \subset Y_2$ (and $Z_2 \subseteq Z_1$),
- if $x \in S_1 \cap Z_1$ and $x \in Z_2$, then $x \in S_2$,
- if $x \in Y_1 \setminus S_1$, then $x \in Y_2 \setminus S_2$, and
- $|S_1 \setminus S_2| + \max(0, |Z_1 \cap Y_2 \setminus (S_1 \cup S_2)| - \ell) \leq k$.

We let the weight of an arc from (Y_1, Z_1, S_1) to (Y_2, Z_2, S_2) be

$$|S_1 \setminus S_2| + \max(0, |Z_1 \cap Y_2 \setminus (S_1 \cup S_2)| - \ell).$$

This finishes the description of the auxiliary weighted acyclic digraph. In the remainder of the proof we first show that (D, ℓ, k) is a YES-instance if and only if the cost of the shortest path in $\mathcal{D}$ from $(\emptyset, V(D), \emptyset)$ to $(V(D), \emptyset, \emptyset)$ is at most k. Afterwards, we bound $|\mathcal{V}| + |\mathcal{A}|$ by $\mathcal{O}(2^{16k}n^2)$ and prove that we can construct the auxiliary digraph in $\mathcal{O}(2^{16k}kn^2)$ time. We can then find a shortest path from $(\emptyset, V(D), \emptyset)$ to $(V(D), \emptyset, \emptyset)$ in linear time, that is, in time $\mathcal{O}(2^{16k}n^2)$ since $\mathcal{D}$ is acyclic (by dynamic programming using an acyclic ordering of the vertices), which finishes the proof.

Correctness of the Algorithm. Suppose first that (D, ℓ, k) is a YES-instance of Directed Component Order Connectivity$[k]$ such that D is a semicomplete digraph. Let X be a minimum size solution for (D, ℓ, k), that is, a minimum size set such that $\mathrm{mco}(D - X) \leq \ell$. Since (D, ℓ, k) is YES-instance, $|X| \leq k$ and $\mathrm{mco}(D - X) \leq \ell$, the vertices of $D - X$ can be partitioned in sets $C_1, \ldots, C_q$ such that

1. for every $i \in [q]$ is $|C_i| \leq \ell$, and
2. for every $i, j \in [q]$ with $i < j$ and every $x \in C_i$, $y \in C_j$ we have $xy \in A$ and $yx \notin A$.

Our goal is to define a sequence of valid triples (Y_i, Z_i, S_i), $i \in [q]$, such that the arc $((Y_i, Z_i, S_i), (Y_{i+1}, Z_{i+1}, S_{i+1}))$ is in $\mathcal{A}$ and the cost of the path in $\mathcal{D}$ defined by this sequence is $|X|$. We will construct these triples from X and $C_1, \ldots, C_q$ with some additional restrictions that makes it easier to show that they indeed define a path in $\mathcal{D}$ of cost at most $|X|$. Namely, we will define them such that for all $i, j \in [q]$, $i < j$ the triples satisfy the following properties:

1. (Y_i, Z_i, S_i) is t_i-valid for some $t_i \in [n]$,
2. $C_1 \cup \cdots \cup C_i \subseteq Y_i$,
3. $C_{i+1} \cup \cdots \cup C_q \subseteq Z_i$,
4. $S_i \subseteq X$,
5. $Y_i \subset Y_j$ and $Z_j \subseteq Z_i$,
6. if $x \in S_i \cap Z_i$ and $x \in Z_j$, then $x \in S_j$,
7. if $x \in Y_i \setminus S_i$, then $x \in Y_j \setminus S_j$.

It is straightforward to verify that, given the above properties, the arc

$$((Y_i, Z_i, S_i), (Y_{i+1}, Z_{i+1}, S_{i+1})) \in \mathcal{A}.$$

We first show that a sequence with the above properties indeed exists and defer the computation of the cost of the path defined by this sequence to later.

To obtain this sequence, we need to discuss how to distribute the vertices of X in the sets Y_i's and Z_i's and how to compute S_i, S_j (Note that the partition of the vertices in $V \setminus X$ is fixed by properties 2 and 3).

To distribute the vertices of X between Y_i and Z_i, we put all $x \in X$ with $d_D^+(x) \geq n - t_i + k$ in Y_i and all $x \in X$ with $d_D^-(x) \geq t_i + k$ in Z_i. The remaining vertices in X we can distribute arbitrarily, we only have to make sure that for all $i, j \in [q]$, $i < j$, it holds that $Y_i \subset Y_j$ and $Z_j \subseteq Z_i$. Now $|X| \leq k$ and for all $y \in Y_i \setminus X = C_1 \cup \cdots \cup C_i$ and $z \in Z_i \setminus X = C_{i+1} \cup \cdots \cup C_q$ we have $zy \notin A(D)$. The set S_i is defined to be those vertices $x \in X$ such that one of the following holds:

1. $x \in Y_i$ and there exists $z \in Z_i \setminus X$ such that $zx \in A(D)$,
2. $x \in Z_i$ and there is an arc $xy \in A(D)$, $y \in Y_i$ such that $y \notin S_i$.

Note that all arcs from Z_i to Y_i are covered by S_i and for each $x \in X$ there is an arc zy from Z_i to Y_i with $\{y, z\} \cap X = \{x\}$. Note that if $x \in Y_i \setminus S_i$, then $x \in Y_j \setminus S_j$ for all $j > i$. On the other hand, if $x \in Z_i \cap S_i$, then there is $y \in Y_i \setminus S_i$ such that $xy \in A(D)$. Moreover, for all $j > i$, $y \in Y_j \setminus S_j$. Therefore, if $x \in Z_j$, then $x \in S_j$. From the above two properties it follows that if $x \in S_i \setminus S_j$, then $x \notin S_{j+1} \cup \cdots \cup S_q$. This finishes the proof of the existence of a sequence of valid triples $(Y_1, Z_1, S_1), \ldots, (Y_q, Z_q, S_q)$ with properties 1-7.

We claim that the cost of path following this sequence is $|X| \leq k$. First note that if $x \in S_i \setminus S_{i+1}$, then $x \in Y_{i+1}$ and for all $j \geq i + 1$ it holds $x \notin S_j$, hence every vertex in X is counted in at most one of the sets $S_i \setminus S_{i+1}$. Now the set C_i is precisely $(Z_{i-1} \cap Y_i) \setminus X$. If $x \in Z_{i-1} \cap Y_i \cap X$ is in some set S_j, then from the properties 5, 6 and 7 of the sequence of triples it follows that x is in $S_{i-1} \cup S_i$. Hence $|(Z_{i-1} \cap Y_i) \setminus (S_{i-1} \cup S_i)| - |C_i|$ is precisely the number of vertices in X that are in $Z_{i-1} \cap Y_i$ and in none of the sets S_j, $j \in [q]$. Note that for such vertex $x \in (Z_{i-1} \cap Y_i) \setminus \bigcup_{j \in [q]} S_j$ and a vertex $y \in Y_j \setminus S_j$, for some $j \in [q]$ with $j < i$, it holds $xy \notin A(D)$ (else by definition of a valid triple $|\{x, y\} \cap S_j| \geq 1$). Similarly for $z \in Z_j \setminus S_j$, $j > i$, $zx \notin A(D)$. Hence, if $|C_i| < \ell$, then $X \setminus \{x\}$ would be a smaller solution for the instance (D, ℓ, k) and because of minimality of X, $(|Z_{i-1} \cap Y_i \setminus (S_{i-1} \cup S_i)| - \ell)$ is precisely the number of vertices in X that are in $Z_{i-1} \cap Y_i$ and in none of the sets S_j. It follows that each vertex in X is counted on precisely one arc on the path and the shortest path from $(\emptyset, V(D), \emptyset)$ to $(V(D), \emptyset, \emptyset)$ in $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ has length precisely $|X|$.

For the other direction, let some shortest path in $\mathcal{D}$ from $(\emptyset, V(D), \emptyset)$ to $(V(D), \emptyset, \emptyset)$ be defined by the sequence (Y_i, Z_i, S_i), $i \in \{0, \ldots, q\}$, and assume that the cost of the path is at most k. For every $i \in [q]$, let T_i be an arbitrary set consisting of $(|(Z_{i-1} \cap Y_i) \setminus (S_{i-1} \cup S_i)| - \ell)$ vertices from $(Z_{i-1} \cap Y_i) \setminus (S_{i-1} \cup S_i)$ and let $X = \bigcup_{i \in [q]} (T_i \cup S_i)$. Because the pair $((Y_{i-1}, Z_{i-1}, S_{i-1}), (Y_i, Z_i, S_i))$ is an arc in $\mathcal{D}$ for every $i \in [q]$, we have $Y_{i-1} \subseteq Y_i$ and $Z_i \subseteq Z_{i-1}$. Moreover, $(Y_{i-1}, Z_{i-1}, S_{i-1})$ and $(Y_i, Z_i, S_i))$ are t_{i-1}-valid and t_i-valid triples, for some $t_{i-1}, t_i \in [n]$, respectively. Therefore, there is no arc from $Z_j \setminus X$ to $Y_i \setminus X$ for any $i \leq j \in [q]$. It follows that each strongly connected component of $D - X$ is a subset of $(Z_{i-1} \cap Y_i) \setminus X$ for some $i \in [q]$. In particular note that $(Z_{i-1} \cap Y_i) \cap X = (Z_{i-1} \cap Y_i) \cap (S_{i-1} \cup S_i \cup T_i)$, $(S_{i-1} \cup S_i) \cap T_i = \emptyset$ and $T_i \subseteq (Z_{i-1} \cap Y_i)$. Hence the size of each connected component is at most $\max_{i \in [q]} |(Z_{i-1} \cap Y_i) \setminus (S_{i-1} \cup S_i \cup T_i)| = \max_{i \in [q]} |((Z_{i-1} \cap Y_i) \setminus (S_{i-1} \cup S_i)) \setminus T_i| = \max_{i \in [q]} (|((Z_{i-1} \cap Y_i) \setminus (S_{i-1} \cup S_i))| - |T_i|) \leq \ell$. Since $S_0 = S_q = \emptyset$, every vertex that appears in S_i for some $i \in [q]$ is counted in some $|S_j \setminus S_{j+1}|$, where $j \geq i$ and every vertex that appears in T_i for some $i \in [q]$ is counted in $\max(0, |Z_i \cap Y_{i+1} \setminus (S_i \cup S_{i+1})| - \ell)$ and the final set X has at most k vertices.

Construction of the Auxiliary Weighted Digraph. Note that by Lemma 7, $|\mathcal{V}| \leq 2^{8k+2} n$ and, since we can compute the out- and in-degrees of all vertices in D in time $\mathcal{O}(n^2)$, we can enumerate all vertices in $\mathcal{D}$ in time $\mathcal{O}(2^{8k} k n^2)$. It follows that $|\mathcal{A}| \leq |\mathcal{V}|^2 \leq 2^{16k+4} n^2$ and $|\mathcal{V}| + |\mathcal{A}| = \mathcal{O}(2^{16k} n^2)$. It remains to show that for a pair of triples (Y_1, Z_1, S_1) and

(Y_2, Z_2, S_2), we can check whether $((Y_1, Z_1, S_1), (Y_2, Z_2, S_2))$ is an arc and compute its weight in $\mathcal{O}(k)$ amortized time. First note that if $|Y_1| \geq |Y_2|$, then the arc is not there. We will only check if $((Y_1, Z_1, S_1), (Y_2, Z_2, S_2))$ is an arc if $|Y_1| < |Y_2|$. This can be done without computing the sizes of Y_1 and Y_2, respectively, if we enumerate the t-valid triples in $\mathcal{D}$ in levels in the order increasing t (i.e., we invoke Lemma 7 for t only after we added all t'-valid triples, for all $t' < t$, to $\mathcal{V}$.) and compute all in-neighbours of a vertex when it is added to $\mathcal{V}$. Moreover, when adding the triple (Y, Z, S) in $\mathcal{V}$, we will in $\mathcal{O}(n)$ time compute maps $\alpha_{(Y,Z,S)} : V(D) \to \{0, 1\}$ such that $\alpha_{(Y,Z,S)}(x) = 0$ if and only if $x \in Y$ and $\beta_{(Y,Z,S)} : V(D) \to \{0, 1\}$ such that $\beta_{(Y,Z,S)}(x) = 0$ if and only if $x \in S$. We also compute the set $\Delta_{Y,Z} = \{x \mid x \in V(D), d^+_D(x) \leq |Z| + k, d^-_D(x) \leq |Y| + k\}$. By Lemma 6, $|\Delta_{Y,Z}| \leq 7k + 2$. Now we can describe the $\mathcal{O}(k)$ algorithm that determines whether $((Y_1, Z_1, S_1), (Y_2, Z_2, S_2))$ is an arc.

First, for every $x \in S_1$ we can in constant time check that $x \in S_1 \cap Z_1$ (*i.e.*, $\alpha_{(Y_1,Z_1,S_1)}(x) = 1$ and $\beta_{(Y_1,Z_1,S_1)}(x) = 0$) and $x \in Z_2$ ($\alpha_{(Y_2,Z_2,S_2)}(x) = 1$) implies $x \in S_2$ ($\beta_{(Y_2,Z_2,S_2)} = 0$). Similarly we can check in constant time that if $x \in Y_1$, then $x \in Y_2 \setminus S_2$.

Second, by Lemma 5 and since $|Y_1| < |Y_2|$ and $|Z_1| > |Z_2|$, we get that to check that $Y_1 \subset Y_2$ and $Z_2 \subseteq Z_1$, we only need to check for every $x \in \Delta_{Y_1,Z_1} \cup \Delta_{Y_2,Z_2}$ that $\alpha_{(Y_1,Z_1,S_1)}(y) = 0$ implies $\alpha_{(Y_2,Z_2,S_2)}(x) = 0$. This check can done in $\mathcal{O}(|\Delta_{Y_1,Z_1} \cup \Delta_{Y_2,Z_2}|) = \mathcal{O}(k)$ time. Finally, to compute the weight of the arc, we note that $|Z_1 \cap Y_2|$ is precisely $|Y_2| - |Y_1|$, because $Y_1 \subset Y_2$ and $Z_1 = V(D) \setminus Y_1$, so we only need to check how many of the vertices in $S_1 \cup S_2$ are in $Z_1 \cap Y_2$ and how many of the vertices in S_1 are also in S_2. Moreover, we only need to compute $|(Z_1 \cap Y_2) \setminus (S_1 \cup S_2)| - \ell$ if $\ell < |Y_2| - |Y_1| \leq \ell + 2k$. Else either the weight of the arc is precisely $|S_1 \setminus S_2|$ or it would be more than k and hence it is not an arc. Hence, we end up spending $\mathcal{O}(k + \log n)$ time on the computation of the weight of each of at most $\mathcal{O}(2^{16k}kn)$ many arcs (for which $\ell < |Y_2| - |Y_1| \leq \ell + 2k$) and $\mathcal{O}(k)$ on all of at most $\mathcal{O}(2^{16k}n^2)$ remaining arcs. Since $k \leq n$, we can construct $\mathcal{D}$ in $\mathcal{O}(2^{16k}kn^2)$ time. ◀

In the rest of the section, we will show that the dependency on both k and n cannot be significantly improved. More precisely, we will show an unconditional lower-bound of $\Omega(n^2)$ even if $k = 0$, as we show that we need to read at least $\Omega(n^2)$ arcs of the input instance in the worst case to distinguish between $k = 0$ and $k = 1$. Furthermore, we show that any $2^{o(k)}n^{\mathcal{O}(1)}$ algorithm would imply that Exponential Time Hypothesis fails.

▶ **Theorem 9.** *There is no deterministic sequential algorithm that outputs the correct answer for every instance $(D, \ell, 0)$ of* Directed Component Order Connectivity *when D is a tournament in $o(n^2)$ time.*

Proof. For $i \in \mathbb{N}$, let H_i be an arbitrary but fixed strongly connected tournament on i vertices. If $\frac{n}{2} \leq \ell < n$, then let us consider the graph D obtained by taking disjoint union of $H_{\lfloor \frac{n}{2} \rfloor}$ and $H_{\lceil \frac{n}{2} \rceil}$ and orienting arcs between $H_{\lfloor \frac{n}{2} \rfloor}$ and $H_{\lceil \frac{n}{2} \rceil}$ from $H_{\lfloor \frac{n}{2} \rfloor}$ to $H_{\lceil \frac{n}{2} \rceil}$. Clearly, $\mathrm{mco}(D) = \lceil \frac{n}{2} \rceil \leq \ell$ and $(D, \ell, 0)$ is YES-instance of Directed Component Order Connectivity. Note there are $\lfloor \frac{n}{2} \rfloor \cdot \lceil \frac{n}{2} \rceil = \Theta(n^2)$ arcs between $H_{\lfloor \frac{n}{2} \rfloor}$ and $H_{\lceil \frac{n}{2} \rceil}$. Now let $\mathbb{A}$ be a deterministic sequential algorithm that solves Directed Component Order Connectivity$[k]$ in $o(n^2)$ time if $k = 0$. If we run $\mathbb{A}$ on D, then there is an arc from $H_{\lfloor \frac{n}{2} \rfloor}$ to $H_{\lceil \frac{n}{2} \rceil}$ that $\mathbb{A}$ did not read. Let this arc be xy and let D' be a graph obtained from D by replacing the arc xy by the arc yx. It follows that D' is strongly connected and hence $(D', \ell, 0)$ is NO-instance of Directed Component Order Connectivity. However, because the algorithm $\mathbb{A}$ decided that $(D, \ell, 0)$ is YES-instance without considering the orientation of the arc between x and y on the instance $(D, \ell, 0)$ and the only difference between $(D, \ell, 0)$ and $(D', \ell, 0)$ is the orientation of the arc between x and y, it follows that $\mathbb{A}$ outputs that

$(D', \ell, 0)$ is YES-instance, which contradicts the assumption that $\mathbb{A}$ outputs correct answer for every instance $(D, \ell, 0)$ of Directed Component Order Connectivity such that D is a tournament.

If $\ell < \frac{n}{2}$, the proof is very similar to the above, the only difference is the construction of the digraph D. To construct D we first take the disjoint union of $q = \lfloor \frac{n}{\ell} \rfloor$ copies of H_ℓ, denoted $H_\ell^1, \ldots, H_\ell^q$, and one copy of $H_{n-q\ell}$. We add the arc xy to D if $x \in H_\ell^i$ and $y \in H_\ell^j$ such that $1 \leq i < j \leq q$ or if $x \in H_\ell^i$, $i \in [q]$, and $y \in H_{n-q\ell}$. It follows that D is a tournament and $\mathrm{mco}(D) = \ell$, that is $(D, \ell, 0)$ is a YES-instance. Now let $Y = \bigcup_{i \in [\lfloor \frac{q}{2} \rfloor]} V(H_\ell^i)$ and $Z = V(D) \setminus Y$. It is easy to see that $\frac{n}{4} \leq |Y| \leq \frac{n}{2}$ and there are $\Theta(n^2)$ arcs from Y to Z in D. Moreover if $yz \in A(D)$ is an arc such that $y \in Y$ and $z \in Z$, then $D_{yz} = (V(D), (A(D) \setminus \{yz\}) \cup \{zy\})$ contains a strongly connected components of size at least 2ℓ that includes all vertices in $V(H_\ell^{\lfloor \frac{q}{2} \rfloor}) \cup V(H_\ell^{\lfloor \frac{q}{2} \rfloor + 1})$. The proof follows by analogous arguments to the case $n - \ell < \ell$, as for any algorithm $\mathbb{A}$ that solves (D, ℓ, k) in $o(n^2)$, there is an arc yz such that $\mathbb{A}$ outputs incorrectly that (D_{yz}, ℓ, k) is YES-instance. ◀

Finally, we will present our $\mathcal{O}^*(2^{o(k)})$ lower bound result, based on the well-established Exponential Time Hypothesis (ETH).

Our result uses the fact that the classical Vertex Cover problem cannot be solved in subexponential time under ETH.

▶ **Theorem 10** (Cai and Juedes [6]). *There is no $2^{o(k)} \cdot |V(G)|^{\mathcal{O}(1)}$ algorithm for Vertex Cover, unless ETH fails.*

Given the above result by Cai and Juedes, the lower bound then directly follows from the proof of NP-hardness of Directed Feedback Vertex Set by Speckenmeyer [18]. In fact, given a graph G, Speckenmeyer constructs in $O(|V(G)|^2)$ time a tournament T with $3|V(G)| - 2$ vertices such that for every k the graph G has a vertex cover of size at most k if and only if T has a directed feedback vertex set of size at most k (see Theorem 6 in [18]). Hence, we obtain the following:

▶ **Theorem 11.** *There is no algorithm solving* Directed Component Order Connectivity$[k]$ *on tournaments in time $2^{o(k)} n^{\mathcal{O}(1)}$, unless ETH fails.*

In Theorem 8 we saw that there is an FPT algorithm for Directed Component Order Connectivity$[n - \ell]$ that runs in $\mathcal{O}^*(2^{16(n-\ell)})$ time, as we may assume that $k \leq n - \ell$. By the construction explained before Theorem 11 we can replace k by $n - \ell$ in $2^{o(k)}$ in Theorem 11 and thus obtain a matching lower bound for the upper bound $\mathcal{O}^*(2^{16(n-\ell)})$.

▶ **Theorem 12.** *There is no $2^{o(n-\ell)} n^{\mathcal{O}(1)}$-time algorithm for solving* Directed Component Order Connectivity$[n - \ell]$ *on semicomplete digraphs, unless ETH fails.*

5 Conclusions

Since Directed Component Order Connectivity generalizes Directed Feedback Vertex Set, it would likely be hard to improve our upper bound and obtain a tight lower bound for the time complexity of Directed Component Order Connectivity$[\ell + k]$ on general digraphs. It seems easier to improve our upper and lower bounds on the time complexity of Directed Component Order Connectivity$[k]$ on semicomplete digraphs.

It would be interesting to consider the time complexity of the problem on well-studied generalizations of semicomplete digraphs: (i) *semicomplete multipartite digraphs* which are digraphs that can be obtained from complete multipartite graphs by replacing every edge by an arc with the same end-vertices or a pair of opposite arcs with the same end-vertices, (ii) *quasi-transitive digraphs* which are digraphs in which if xy and yz are arcs such that x, y, z are distinct vertices then either xz or zx or both are arcs, too (in particular, a transitive digraph is quasi-transitive), (iii) *locally semicomplete digraphs* which are digraphs in which the out- and in-neighborhood of every vertex induce semicomplete digraphs (a directed cycle is an example of a locally semicomplete digraph). Chapters 7, 8 and 5 of [2], respectively, provide extensive surveys on these classes of digraphs.

References

1 J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer-Verlag, London, 2nd edition, 2009.

2 J. Bang-Jensen and G.Z. Gutin, editors. *Classes of Directed Graphs*. Springer Monographs in Mathematics. Springer, 2018.

3 J. Bang-Jensen and C. Thomassen. A polynomial algorithm for the 2-path problem for semicomplete digraphs. *SIAM J. Discrete Math.*, 5(3):366–376, 1992. doi:10.1137/0405027.

4 N.H. Bshouty and A. Gabizon. Almost optimal cover-free families. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, volume 10236 of *Lecture Notes in Computer Science*, pages 140–151, 2017.

5 Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM J. Comput.*, 22(3):560–572, 1993.

6 Liming Cai and David Juedes. On the existence of subexponential parameterized algorithms. *J. Comput. System Sci.*, 67(4):789–807, 2003.

7 J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. Assoc. Comput. Mach.*, 55(5):21:1–21:19, 2008.

8 M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

9 R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

10 R.G. Downey and M.R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

11 P.G. Drange, M.S. Dregi, and P. van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016.

12 A. Göke, D. Marx, and M. Mnich. Parameterized algorithms for generalizations of directed feedback vertex set. In P. Heggernes, editor, *Algorithms and Complexity. CIAC 2019*, volume 11485 of *Lecture Notes in Computer Science*. Springer, Cham, 2019. doi:10.1007/978-3-030-17402-6_21.

13 D. Gross, M. Heinig, L. Iswara, L. W. Kazmierczak, K. Luttrell, J. T. Saccoman, and C. Suffel. A survey of component order connectivity models of graph theoretic networks. *WSEAS Transactions on Mathematics*, 12:895–910, 2013.

14 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001. Special issue on FOCS 98 (Palo Alto, CA).

15 M. Kumar and D. Lokshtanov. A $2lk$ kernel for l-component order connectivity. In J. Guo and D. Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

16 E. Lee. Partitioning a graph into small pieces with applications to path transversal. *Math. Program.*, 177(1-2):1–19, 2019.

17 Rian Neogi, M. S. Ramanujan, Saket Saurabh, and Roohani Sharma. On the parameterized complexity of deletion to -free strong components. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 75:1–75:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.75.

18 E. Speckenmeyer. On feedback problems in digraphs. In M. Nagl, editor, *Graph-Theoretic Concepts in Computer Science, 15th International Workshop, WG '89, 1989, Proceedings*, volume 411 of *Lecture Notes in Computer Science*, pages 218–231. Springer, 1989. doi:10.1007/3-540-52292-1.

Close Relatives of Feedback Vertex Set Without Single-Exponential Algorithms Parameterized by Treewidth

Benjamin Bergougnoux
Department of Informatics, University of Bergen, Norway
benjamin.bergougnoux@uib.no

Édouard Bonnet
Université Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France
edouard.bonnet@ens-lyon.fr

Nick Brettell
School of Mathematics and Statistics, Victoria University of Wellington, New Zealand
nick.brettell@vuw.ac.nz

O-joung Kwon
Department of Mathematics, Incheon National University, South Korea
Discrete Mathematics Group, Institute for Basic Science (IBS), Daejeon, South Korea
ojoungkwon@gmail.com

Abstract

The Cut & Count technique and the rank-based approach have lead to single-exponential FPT algorithms parameterized by treewidth, that is, running in time $2^{\mathcal{O}(\mathsf{tw})}n^{\mathcal{O}(1)}$, for Feedback Vertex Set and connected versions of the classical graph problems (such as Vertex Cover and Dominating Set). We show that Subset Feedback Vertex Set, Subset Odd Cycle Transversal, Restricted Edge-Subset Feedback Edge Set, Node Multiway Cut, and Multiway Cut are unlikely to have such running times. More precisely, we match algorithms running in time $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}n^{\mathcal{O}(1)}$ with tight lower bounds under the Exponential Time Hypothesis, ruling out $2^{o(\mathsf{tw}\log\mathsf{tw})}n^{\mathcal{O}(1)}$, where n is the number of vertices and tw is the treewidth of the input graph. Our algorithms extend to the weighted case, while our lower bounds also hold for the larger parameter pathwidth and do not require weights. We also show that, in contrast to Odd Cycle Transversal, there is no $2^{o(\mathsf{tw}\log\mathsf{tw})}n^{\mathcal{O}(1)}$-time algorithm for Even Cycle Transversal.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases Subset Feedback Vertex Set, Multiway Cut, Parameterized Algorithms, Treewidth, Graph Modification, Vertex Deletion Problems

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.3

Related Version A full version of the paper is available at `https://arxiv.org/abs/2007.14179`.

Acknowledgements This work was initiated while the authors attended the "2019 IBS Summer research program on Algorithms and Complexity in Discrete Structures", hosted by the IBS discrete mathematics group.

1 Introduction

Courcelle's Theorem [8] states that any problem definable in MSO_2 logic can be solved in linear time on graphs of bounded treewidth. However, the algorithms obtained through Courcelle's meta-theorem have a huge dependency on treewidth. For certain problems, efforts have been spent to find the "smallest" function f for which we can obtain an algorithm that, given a graph with treewidth tw, has running time $f(\mathsf{tw})n^{\mathcal{O}(1)}$. For Feedback Vertex

Set, standard dynamic programming techniques can be used to obtain an algorithm running in $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}n^{\mathcal{O}(1)}$ time, and for a while many believed this to be, in a sense, best possible. However, this changed in 2011 when Cygan et al. [11] developed the Cut&Count technique, by which they obtained a *single-exponential* $3^{\mathsf{tw}}n^{\mathcal{O}(1)}$-time randomized algorithm. Following this, Bodlaender et al. [3] showed there is a deterministic $2^{\mathcal{O}(\mathsf{tw})}n^{\mathcal{O}(1)}$-time algorithm, using a rank-based approach and the concept of representative sets. Moreover, also in 2011, Lokshtanov et al. [17] developed a framework yielding $2^{\Omega(\mathsf{tw}\log\mathsf{tw})}n^{\mathcal{O}(1)}$-time lower bounds under the Exponential Time Hypothesis (ETH) [16]. Recall that the ETH asserts that there is a real number $\delta > 0$ such that 3-SAT cannot be solved in time $2^{\delta n}$ on n-variable formulas. Lokshtanov et al.'s paper prompted several authors to investigate the exact time-dependency on treewidth for a variety of graph modification problems.

For a *vertex-deletion problem*, the task is to delete at most k vertices so that the resulting graph is in some target class. Feedback Vertex Set can be viewed as a vertex-deletion problem where the graphs in the target class consist of blocks with at most two vertices (a *block* is a maximal subgraph H such that H has no cut vertices). Bonnet et al. [6] considered the class of problems, generalizing Feedback Vertex Set, where the target graphs consist of blocks each of which has a bounded number of vertices, and is in some fixed hereditary, polynomial-time recognizable class $\mathcal{P}$. They showed that such a problem is solvable in time $2^{\mathcal{O}(\mathsf{tw})}n^{\mathcal{O}(1)}$ precisely when each graph in $\mathcal{P}$ is chordal (when $\mathcal{P}$ does not satisfy this condition, an algorithm with running time $2^{o(\mathsf{tw}\log\mathsf{tw})}n^{\mathcal{O}(1)}$ would refute the ETH). Baste et al. [2] studied another generalization of Feedback Vertex Set: the vertex-deletion problem where the target graphs are those having no minor isomorphic to a fixed graph H. They showed a single-exponential parameterized algorithm in treewidth is possible precisely when H is a minor of the banner (the cycle on four vertices with a degree-1 vertex attached to it), but H is not P_5 (the path graph on five vertices), assuming the ETH holds.

Slightly superexponential parameterized algorithms, running in time $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}n^{\mathcal{O}(1)}$, are by no means a formality for problems that are FPT in treewidth. For instance, Pilipczuk [19] showed that deciding if a graph has a transversal of size at most k hitting all cycles of length exactly ℓ (or length at most ℓ) for a fixed value ℓ cannot be solved in time $2^{o(\mathsf{tw}^2)}n^{\mathcal{O}(1)}$, unless the ETH fails. This lower bound matches a dynamic-programming based algorithm running in time $2^{\mathcal{O}(\mathsf{tw}^2)}n^{\mathcal{O}(1)}$. Cygan et al. [9] investigated the more general problem of hitting all subgraphs H of a given graph G, for a fixed pattern graph H, again parameterized by treewidth. For various H, they found algorithms running in time $2^{\mathcal{O}(\mathsf{tw}^{u(H)})}n^{\mathcal{O}(1)}$, and proved ETH lower bounds in $2^{\Omega(\mathsf{tw}^{\ell(H)})}n^{\mathcal{O}(1)}$, for values $1 < \ell(H) \leqslant u(H)$ depending on H. Another recent example is provided by Sau and Uéverton [20] who prove similar results for the analogous problem where "subgraphs" is replaced by "induced subgraphs". Finally, for the vertex-deletion problem where the target class is a proper minor-closed class given by the non-empty list of forbidden minors, it is still open if the double-exponential dependence on treewidth is asymptotically best possible [1].

Sometimes, only a seemingly slight generalization of Feedback Vertex Set can result in problems with no single-exponential algorithm parameterized by treewidth. Bonamy et al. [5] showed that Directed Feedback Vertex Set can be solved in time $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}n^{\mathcal{O}(1)}$ but not faster under the ETH, where tw is the treewidth of the underlying undirected graph. In this paper, we consider another collection of problems that generalize Feedback Vertex Set, and that do not have single-exponential algorithms parameterized by treewidth. An equivalent formulation of FVS is to find a transversal of *all* cycles in a given graph. We consider problems where the goal is to find a transversal of *some subset* of the cycles of a given graph. If this subset of cycles is the set of cycles intersecting some fixed set of vertices S, we obtain the following problem:

Subset Feedback Vertex Set (Subset FVS)	**Parameter:** tw(G)
Input: A graph G, a subset of vertices $S \subseteq V(G)$, and an integer k.	
Question: Is there a set of at most k vertices hitting all the cycles containing a vertex in S?	

If we further restrict this set of cycles to those that are odd, we obtain the next problem:

Subset Odd Cycle Transversal (Subset OCT)	**Parameter:** tw(G)
Input: A graph G, a subset of vertices $S \subseteq V(G)$, and an integer k.	
Question: Is there a set of at most k vertices hitting all the odd cycles going through a vertex in S?	

Both of these problems are NP-complete. By setting $S = V(G)$, one sees that the latter problem generalises Odd Cycle Transversal, for which Fiorini et al. [14] presented a $2^{\mathcal{O}(\mathsf{tw})}n^{\mathcal{O}(1)}$-time algorithm.

Alternatively, one can require a transversal of even cycles. We first consider the problem of finding a transversal of *all* even cycles since, to the best of our knowledge, the fine-grained complexity of this problem parameterized by treewidth has not previously been studied.

Even Cycle Transversal (ECT)	**Parameter:** tw(G)
Input: A graph G and an integer k.	
Question: Is there a set of at most k vertices hitting all the even cycles of G?	

We now move to edge variants of FVS. Note that Feedback Edge Set, where the goal is to find a set of edges of weight at most k that hits the cycles, can be solved in linear time, since it is equivalent to finding a maximum-weight spanning forest. Xiao and Nagamochi showed that the subset variants Vertex-Subset Feedback Edge Set and Edge-Subset Feedback Edge Set, where the deletion set only needs to hit cycles containing a vertex or an edge (respectively) of a given set S, can also be solved in linear time [21]. On the other hand, the latter problem becomes NP-complete when the deletion set cannot intersect S. This problem is known as Restricted Edge-Subset Feedback Edge Set.

Restricted Edge-Subset Feedback Edge Set (RESFES)	**Parameter:** tw(G)
Input: A graph G, a subset of edges $S \subseteq E(G)$, and an integer k.	
Question: Is there a set of at most k edges of $E(G) \setminus S$ whose removal yields a graph without any cycle containing at least one edge of S?	

The final two NP-complete problems we consider are closely related to Subset Feedback Vertex Set and Restricted Edge-Subset Feedback Edge Set. They are well-known problems with an abundant literature of approximation and parameterized algorithms.

Node Multiway Cut	**Parameter:** tw(G)
Input: A graph G, a subset of vertices $T \subseteq V(G)$, called *terminals*, and an integer k.	
Question: Is there a set of at most k vertices of $V(G) \setminus T$ hitting every path between a pair of terminals?	

Multiway Cut	**Parameter:** tw(G)
Input: A graph G, a subset of vertices $T \subseteq V(G)$, called *terminals*, and an integer k.	
Question: Is there a set of at most k edges hitting every path between a pair of terminals?	

The look-alike problem Multicut, where the task is to separate each pair of terminals in a given set of pairs (rather than all the pairs in a given set), is NP-complete on trees [15]. Therefore a parameterization by treewidth cannot help here. In the language of parameterized complexity, Multicut parameterized by treewidth is paraNP-complete.

1.1 Our contribution

With the exception of Even Cycle Transversal, for which we provide only a lower bound, we show that all the problems formally defined so far admit a slightly superexponential parameterized algorithm, and that this running time cannot be improved, unless the ETH fails. We leave as an open problem the existence of a slightly superexponential algorithm for (Subset) Even Cycle Transversal parameterized by treewidth. We note that Deng et al. [12] have already shown that Multiway Cut can be solved in time $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}n^{\mathcal{O}(1)}$. Our algorithms work for treewidth and weights, while our lower bounds hold for the larger parameter pathwidth and do not require weights.

On the algorithmic side we show the following:

▶ **Theorem 1.** *The following problems can be solved in time* $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}n^{\mathcal{O}(1)}$ *on* n*-vertex graphs with treewidth* tw:

- *Subset Feedback Vertex Set,*
- *Subset Odd Cycle Transversal,*
- *Restricted Edge-Subset Feedback Edge Set, and*
- *Node Multiway Cut.*

We provide algorithms having the claimed running time for the weighted versions of each of the four problems in Theorem 1. In these weighted versions, the input graph is given with a weight function w on the vertices when the problem is to find a set of vertices, or on the edges when the problem is to find a set of edges. Furthermore, in the weighted versions, the problem asks for a solution of weight at most k.

On the complexity side, the main conceptual contribution of the paper is to show that problems seemingly quite close to Feedback Vertex Set do not admit a single-exponential algorithm parameterized by treewidth, under the ETH.

▶ **Theorem 2.** *Unless the ETH fails, the following problems cannot be solved in time* $2^{o(\mathsf{pw}\log\mathsf{pw})}n^{\mathcal{O}(1)}$ *on* n*-vertex graphs with pathwidth* pw:

- *Subset Feedback Vertex Set,*
- *Subset Odd Cycle Transversal,*
- *Even Cycle Transversal,*
- *Restricted Edge-Subset Feedback Edge Set,*
- *Node Multiway Cut, and*
- *Multiway Cut.*

For the last two problems, our reductions build instances where the number of terminals $|T|$ is $\Theta(\mathsf{pw})$. Thus we also rule out a running time of $|T|^{o(\mathsf{pw})}$. All the reductions are from $k \times k$-(Permutation) Independent Set/Clique following a strategy suggested by Lokshtanov et al. [18] (see for instance, [2,5–7,13]). These problems cannot be solved in time $2^{o(k\log k)}$, unless the ETH fails.

$k \times k$-Independent Set	**Parameter:** k
Input: A graph H with vertex set $V(H) = [k]^2$ for some integer k.	
Question: An independent set of size k hitting each column exactly once.	

$k \times k$-Permutation Independent Set	**Parameter:** k
Input: A graph H with vertex set $V(H) = [k]^2$ for some integer k.	
Question: An independent set of size k hitting each column and each row exactly once.	

A *row* is a set of vertices of the form $\{(i,1),(i,2),\ldots,(i,k)\} \subset V(H)$ for some $i \in [k]$, while a column is a set $\{(1,j),(2,j),\ldots,(k,j)\} \subset V(H)$ for some $j \in [k]$. The problem $k \times k$-(Permutation) Clique is defined analogously, where the solution is required to be a clique rather than an independent set.[1]

Roadmap for the lower bounds. To prove Theorem 2, we start by designing a gadget specification for generic vertex-deletion problems. We show that any such problem, allowing for gadgets respecting the specification, has the lower bound given in Theorem 2. This is achieved by a meta-reduction from $k \times k$-Permutation Independent Set. We give gadgets for Subset FVS, Subset OCT, and ECT that comply with the specification. We thus obtain the first three items of the theorem in a unified way, with simple and reusable gadgets. This mini-framework may in principle be useful for other vertex-deletion problems.

In order to show a stronger lower bound for Node Multiway Cut, with the number of terminals in $\Theta(k)$, we depart from the previous specification slightly, although we still use some shared notation and arguments to bound the pathwidth, where convenient. This reduction is from $k \times k$-Independent Set.

Finally, the reduction to Multiway Cut is more intricate. For this problem it is surprisingly challenging to discourage the undesirable solutions "cutting close" to every terminal but one, where the deletion set yields a very large connected component for one terminal, and small components for the rest of the terminals. In particular, the trick used for the Node Multiway Cut lower bound cannot be replicated. We overcome this issue by designing a somewhat counter-intuitive edge gadget which encourages the retention of as many pairs of endpoints linked to two (distinct) terminals as possible. This uses the simple fact that, in a Δ-regular graph, a clique of size k minimizes the number of edges covered by k vertices: $\Delta k - \binom{k}{2}$ vs Δk for an independent set of size k. We then reduce from $k \times k$-Permutation Clique. We discuss why getting the same lower bound for a regular variant of $k \times k$-Permutation Clique is technical, and bypass that difficulty by encoding a *degree-equalizer* gadget directly in the Multiway Cut instance. As a side note, we nevertheless prove that a semi-regular variant of $k \times k$-Clique also has the slightly superexponential lower bound. This proof uses a constructive version of the Hajnal-Szemerédi theorem on equitable colorings.

Roadmap for the algorithms. To prove Theorem 1, we first present a $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}n^3$-time algorithm for the weighted variant of Subset OCT. With a few modifications, it can solve the weighted variant of Subset FVS. We obtain algorithms for the other problems in Theorem 1 by reducing these problems to the weighted variant of Subset FVS.

Let us explain our approach for Subset OCT on a graph G with $S \subseteq V(G)$. We solve Subset OCT indirectly by finding a set $X \subseteq V(G)$ of maximum weight that induces a graph with no odd cycles traversing S (we call such a graph S-bipartite). We prove that a graph has no odd cycle traversing S if and only if for each block C, either C is bipartite or C has no vertex in S. From this characterization, we prove that it is enough to store $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}$ partial solutions at each bag B of a tree decomposition.

[1] Observe that we switch the columns and the rows compared to the original definition of $k \times k$-Clique [18]. While this is of course equivalent, it will make the representation of some gadgets slightly more conducive to the page layout.

Let B be a bag of the tree decomposition of G and G_B be the graph induced by the vertices in B and its descendant bags in the tree decomposition. A partial solution of G_B is a set $X \subseteq V(G_B)$ that induces an S-bipartite graph. We design an equivalence relation $\equiv_B$ on the partial solutions of G_B such that for every $X \equiv_B Y$ and $W \subseteq V(G) \setminus V(G_B)$, $G[X \cup W]$ is S-bipartite if and only if $G[Y \cup W]$ is S-bipartite. Consequently, it is enough to keep a partial solution of maximum weight for each equivalence class of $\equiv_B$. Intuitively, the equivalence relation $\equiv_B$ is based on the information: (1) how the blocks of $G[X]$ intersecting B are connected, (2) whether important blocks (that have the possibility to create an S-traversing odd cycle later) contain a vertex of S, and (3) the parity of the paths between the vertices in B. Since $\equiv_B$ has $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}$ equivalence classes, we deduce from this equivalence relation a $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}n^3$-time algorithm with standard dynamic-programming techniques. The polynomial factor n^3 appears because we can test $X \equiv_B Y$ in time $\mathcal{O}(n^2)$.

For the weighted variant of Subset FVS, we can use the same equivalence relation without (3). We reduce the weighted variant of Node Multiway Cut to Subset FVS by adding a vertex v of infinite weight adjacent to the set of terminals, setting $S = \{v\}$, and also giving infinite weights to the terminals. Furthermore, we reduce the weighted variant of Restricted Edge-Subset Feedback Edge Set to the weighted variant of Subset FVS by subdividing each edge, setting S as the set of subdivided vertices corresponding to the given subset of edges, and giving infinite weights to the original vertices and the vertices in S. These two reductions show that both problems admit $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}n^3$-time algorithms.

Organization. The rest of the paper is organized as follows. In Section 3 we prove all the ETH lower bounds of Theorem 2. More precisely, in Section 3.1 we introduce a gadget specification for a generic vertex-deletion problem, and we show the slightly superexponential lower bound for any problem complying with the gadget specification. In Section 3.2 we design gadgets for Subset FVS, Subset OCT, ECT, and thus obtain the first three items of Theorem 2. In Sections 3.3 and 3.4 we present specific reductions for Node Multiway Cut and Multiway Cut, respectively. In Section 4 we prove that the weighted variants of Subset OCT, Subset FVS, Restricted Edge-Subset Feedback Edge Set, and Node Multiway Cut admit $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}n^3$-time algorithms. The statements marked with a $\star$ have their proof deferred to the full version.

2 Preliminaries

Our graph-theoretic terminology is standard; any terminology undefined here is deferred to the long version. A set $X \subseteq V(G)$ is a *clique* if G has an edge between every pair of vertices in X. A graph with vertex set $X \cup Y$ that has an edge between every vertex $x \in X$ and $y \in Y$ is called a *biclique*, and is denoted $K_{|X|,|Y|}$. For $u, v \in V(G)$, we say that u and v are *twins* if $N(u) = N(v)$. If u and v are adjacent, then we say that u and v are *true* twins; whereas when u and v are non-adjacent twins, we say that u and v are *false* twins.

A vertex v of G is a *cut vertex* if the deletion of v from G increases the number of connected components. We say G is *2-connected* if it is connected and has no cut vertices. Note that every connected graph on at most two vertices is 2-connected. A *block* of G is a maximal 2-connected subgraph of G.

A *tree decomposition* of a graph G is a pair $(T, \mathcal{B})$ consisting of a tree T and a family $\mathcal{B} = \{B_t\}_{t \in V(T)}$ of sets $B_t \subseteq V(G)$, called *bags*, satisfying the following three conditions:

1. $V(G) = \bigcup_{t \in V(T)} B_t$,
2. for every edge uv of G, there exists a node t of T such that $u, v \in B_t$, and
3. for $t_1, t_2, t_3 \in V(T)$, $B_{t_1} \cap B_{t_3} \subseteq B_{t_2}$ whenever t_2 is on the path from t_1 to t_3 in T.

The *width* of a tree decomposition $(T, \mathcal{B})$ is $\max\{|B_t| - 1 : t \in V(T)\}$. The *treewidth* of G is the minimum width over all tree decompositions of G. A *path decomposition* is a tree decomposition $(P, \mathcal{B})$ where P is a path. The *pathwidth* of G is the minimum width over all path decompositions of G. We denote a path decomposition $(P, \mathcal{B})$ as $(B_{v_1}, \ldots, B_{v_t})$, where P is a path $v_1 v_2 \cdots v_t$.

3 Superexponential lower bounds parameterized by treewidth

Our reductions for Subset FVS, Subset OCT, and ECT, in Section 3.2, will have the same skeleton. In order to avoid repeating the same arguments, we show in Section 3.1 the lower bound of Theorem 2 for a meta-problem. We prove the lower bound for Node Multiway Cut in Section 3.3, and the lower bounds for Multiway Cut and Restricted Edge-Subset Feedback Edge Set in Section 3.4.

3.1 Lower bound for a generic vertex-deletion problem

The scope of application of Theorem 2 is any *hereditary* vertex-deletion problem Π; that is, if $G - X$ satisfies a problem instance $P(\Pi)$, then $G - X'$ also satifies $P(\Pi)$ for every $X' \supseteq X$. The main part of the input is a graph G and a non-negative integer k'. In addition, we allow any sort of labelings of G, be it subsets of vertices $S_1, S_2, \ldots \subseteq V(G)$, of edges $E_1, E_2, \ldots, \subseteq E(G)$, pairs of vertices $P_1, P_2, \ldots \subseteq \binom{V(G)}{2}$, etc. The goal is to find a subset $X \subseteq V(G)$ of k' vertices such that a property $P(\Pi)$, dependent on Π, is satisfied on $G - X$ with its induced labeling. A subset of vertices $A \subseteq V(G)$ is a Π-*obstruction* if $G[A]$ does not satisfy $P(\Pi)$. A set $X \subseteq V(G)$ is Π-*legal* if $G - X$ satisfies $P(\Pi)$ (in particular, solutions are Π-legal sets of size k'). As $P(\Pi)$ is assumed hereditary, a Π-legal set intersects every Π-*obstruction*. Finally a Π-*legal s-deletion within Y* is a set $X \subseteq Y$ of size at most s such that $G[Y \setminus X]$ satisfies $P(\Pi)$.

Common base

The meta-result of Theorem 3 concerns hereditary vertex-deletion problems admitting four types of gadgets. These gadgets, which will eventually depend on Π, are attached to a common problem-independent base. We first describe the common base. $H_\bullet$ is a set of $2k^2$ vertices, for some implicit positive integer k. We denote these vertices by $v_\bullet(i, j, z)$ for each $i \in [k]$, $j \in [k]$, and $z \in [2]$. We imagine the vertices of $H_\bullet$ being displayed in a k-by-k grid with $v_\bullet(i, j, 1)$ and $v_\bullet(i, j, 2)$ side by side in the i-th row and j-th column.

The *base* consists of copies of $H_\bullet$ that we denote by $H_1, H_2, \ldots$ and typically index by p. The vertices of H_p are denoted by $v_p(i, j, z)$. The vertices $v_p(i, j, 1)$ and $v_p(i, j, 2)$ are said to be *homologous*. We set $C_{p,j} := \bigcup_{i \in [k], z \in [2]} \{v_p(i, j, z)\}$ and refer to it as the *j-th column* of H_p. Similarly $R_{p,i} := \bigcup_{j \in [k], z \in [2]} \{v_p(i, j, z)\}$ is called the *i-th row* of H_p. We can attach to the base a list of gadgets as detailed now. The vertices added to the base are called *additional* or *new*.

Column selector gadget

A *k-column selector* gadget has the following specification. Its vertex set is a single column $C_{p,j}$ plus $\mathcal{O}(k)$ additional vertices $\mathcal{C}_{\text{sel}}(p, j)$. The only restriction on the edge set of the gadget is that homologous vertices should remain non-adjacent. Other than that, any edge can be added within $C_{p,j}$. However the open neighborhood of $\mathcal{C}_{\text{sel}}(p, j)$ has to be contained in $C_{p,j}$.

A problem Π *admits a column selector gadget* if, for every positive integer k, one can build in time $k^{\mathcal{O}(1)}$ a k-column selector such that the only Π-legal $(2k-2)$-deletions within $C_{p,j} \cup \mathcal{C}_{\text{sel}}(p,j)$ are one of the k sets: $C_{p,j} \setminus \{v_p(1,j,1), v_p(1,j,2)\}, C_{p,j} \setminus \{v_p(2,j,1), v_p(2,j,2)\}, \ldots,$ $C_{p,j} \setminus \{v_p(k,j,1), v_p(k,j,2)\}$.

Row selector gadget

In order to keep small balanced separators, our *k-row selector* gadget is quite different from the k-column selector. Its vertex set is a single row $R_{p,i}$ plus $\mathcal{O}(1)$ additional vertices $\mathcal{R}_{\text{sel}}(p,i)$. Furthermore *no* edge can be added within $R_{p,i}$. Again the open neighborhood of $\mathcal{R}_{\text{sel}}(p,i)$ has to be contained in $R_{p,i}$.

A problem Π *admits a row selector gadget* if, for every positive integer k, one can build in time $k^{\mathcal{O}(1)}$ a k-row selector such that, for every $j \neq j' \in [k]$, $\mathcal{R}_{\text{sel}}(p,i) \cup \{v_p(i,j,1), v_p(i,j,2),$ $v_p(i,j',1), v_p(i,j',2)\}$ is a Π-obstruction.

Edge gadget

The vertex set of an *edge gadget* is of the form $\{v_p(i,j,1), v_p(i,j,2), v_p(i',j',1), v_p(i',j',2)\} \cup \mathcal{E}_p(i,j,i',j')$ where $i \neq i' \in [k]$, $j \neq j' \in [k]$, and $\mathcal{E}_p(i,j,i',j')$ is a set of $\mathcal{O}(k)$ vertices[2]. There is no restriction on the edge set. As usual the open neighborhood of $\mathcal{E}_p(i,j,i',j')$ has to be contained in $\{v_p(i,j,1), v_p(i,j,2), v_p(i',j',1), v_p(i',j',2)\}$.

A problem Π *admits an edge gadget* if one can build in time $k^{\mathcal{O}(1)}$ an edge gadget such that $\mathcal{E}_p(i,j,i',j') \cup \{v_p(i,j,1), v_p(i,j,2), v_p(i',j',1), v_p(i',j',2)\}$ is a Π-obstruction.

Propagation gadget

The vertex set of a *propagation gadget* is of the form $H_p \cup H_{p+1} \cup \mathcal{P}_p$ where $\mathcal{P}_p$ is a set of $k^{\mathcal{O}(1)}$ vertices. There is a subset $\mathcal{P}'_p \subseteq \mathcal{P}_p$ of size $\mathcal{O}(k)$ such that each vertex of $\mathcal{P}_p \setminus \mathcal{P}'_p$ has at most one neighbor in $H_p \cup H_{p+1}$ and the rest of its neighborhood in $\mathcal{P}'_p$. This fairly technical condition aims to give some extra flexibility while keeping sufficiently small separators between H_p and H_{p+1}. In particular, if $\mathcal{P}_p$ is itself of size $\mathcal{O}(k)$, then the condition is trivially met with $\mathcal{P}'_p = \mathcal{P}_p$. The propagation gadget has no edge with both endpoints in $H_p \cup H_{p+1}$. Everything else is permitted, but the open neighborhood of $\mathcal{P}_p$ has to be contained in $H_p \cup H_{p+1}$.

A problem Π *admits a propagation gadget* if one can build in time $k^{\mathcal{O}(1)}$ a propagation gadget such that for every $i, j \neq j' \in [k]$, $\mathcal{P}_p \cup \{v_p(i,j,1), v_p(i,j,2), v_{p+1}(i,j',1), v_{p+1}(i,j',2)\}$ is a Π-obstruction.

Intended-solution property

A hereditary vertex-deletion problem Π and a description of the four above gadgets for Π have the *intended-solution property* if the following holds. On any graph G built by adding to the base $H_1 \cup \ldots \cup H_p \cup \ldots H_m$ at most one edge gadget in each H_p, one propagation gadget between *consecutive* pairs H_p and H_{p+1}, and some column and row selector gadgets, every deletion set $\bigcup_{p\in[m], i\in[k], j\in[k]\setminus\{j_i\}, z\in[2]} \{v_p(i,j,z)\}$ (with $\{j_1, j_2, \ldots, j_k\} = [k]$) intersecting every edge gadget is Π-legal.

We can now state the lower bound for the generic hereditary vertex-deletion problems.

[2] $\mathcal{O}(1)$ vertices will actually suffice for all the gadgets of Section 3.2.

▶ **Theorem 3.** *Unless the ETH fails, every vertex-deletion problem* Π *admitting a column selector, a row selector, an edge, and a propagation gadget, satisfying the intended-solution property, cannot be solved in time* $2^{o(\mathsf{pw}\log\mathsf{pw})}n^{\mathcal{O}(1)}$ *on* n*-vertex graphs with pathwidth* pw*.*

Proof. From any instance H of $k \times k$-Permutation Independent Set, we build an equivalent Π-instance $(G, k' = k^{\mathcal{O}(1)})$ of size $k^{\mathcal{O}(1)}$ with pathwidth in $\mathcal{O}(k)$. Since under the ETH there is no algorithm solving $k \times k$-Permutation Independent Set in time $2^{o(k \log k)}k^{\mathcal{O}(1)}$, we derive the claimed lower bound.

Construction. We number the edges in $E(H)$ as $e_1, \ldots, e_m$. We start with a base consisting of m copies of $H_\bullet$, labelled H_p for $p \in [m]$ (see description of the common base). The vertices $v_p(i,j,1)$ and $v_p(i,j,2)$ encode the vertex $(i,j) \in V(H)$; recall that we call such a pair *homologous*. We attach to each column $C_{p,j}$, for $p \in [m]$ and $j \in [k]$, a column selector gadget (for Π), with additional vertices $\mathcal{C}_{\text{sel}}(p,j)$. For each pair $p \in [m], i \in [k]$, we add a row selector gadget to $R_{p,i}$, with additional vertices $\mathcal{R}_{\text{sel}}(p,i)$.

For each edge $e_p = (i_p, j_p)(i'_p, j'_p) \in E(H)$ $(p \in [m])$, we attach an edge gadget, with additional vertices $\mathcal{E}_p(i_p, j_p, i'_p, j'_p)$, to $\{v_p(i_p, j_p, 1), v_p(i_p, j_p, 2), v_p(i'_p, j'_p, 1), v_p(i'_p, j'_p, 2)\}$. For each $p \in [m-1]$, we add a propagation gadget between H_p and H_{p+1}, with additional vertices $\mathcal{P}_p$. This finishes the construction of G. We set $k' := 2(k-1)km$.

Correctness. We first assume that there is a solution I to $k \times k$-Permutation Independent Set. That is, I is an independent set of H with exactly one vertex per column and per row. Say the vertices of I are $(1, j_1), (2, j_2), \ldots (k, j_k)$ with $\{j_1, j_2, \ldots, j_k\} = [k]$. Then

$$X := \bigcup_{p \in [m]} H_p \setminus \cup_{i \in [k]} \{v_p(i, j_i, 1), v_p(i, j_i, 2)\}$$

is a solution to Π. Indeed it is Π-legal since it intersects every edge gadget (if not, the edge gadget would be between two vertices of I, a contradiction) and Π satisfies the intended-solution property, by assumption. Furthermore $|X| = 2mk(k-1) = k'$.

We now assume that the Π-instance (G, k') admits a solution (of size k'), say X. The graph G has km disjoint Π-obstructions $C_{p,j} \cup \mathcal{C}_{\text{sel}}(p,j)$. For each of these sets, at least $s := 2(k-1)$ vertices must be deleted, by the specification of the column sector gadget. Since globally only $k' = kms$ vertices can be deleted, X intersects each $C_{p,j} \cup \mathcal{C}_{\text{sel}}(p,j)$ at a set $C_{p,j} \setminus \{v_p(i_{j,p}, j, 1), v_p(i_{j,p}, j, 2)\}$ for some $i_{j,p} \in [k]$. Moreover, the k row selector gadgets attached to each H_p enforce that $\{i_{1,p}, i_{2,p}, \ldots, i_{k,p}\} = [k]$, and the propagation gadget $\mathcal{P}_p$ enforces that $i_{j,p} = i_{j,p+1}$ for every $j \in [k]$. This implies that $i_{j,1} = i_{j,2} = \ldots = i_{j,m}$ for every $j \in [k]$, and we simply denote this common value by i_j. We claim that $\{(i_1, 1), (i_2, 2), \ldots, (i_k, k)\}$ is a solution to the $k \times k$-Permutation Independent Set instance. We have already argued that $\{i_1, i_2, \ldots, i_k\} = [k]$. Finally there cannot be an edge $e_p = (i_j, j)(i_{j'}, j') \in E(H)$ since then the Π-obstruction $\mathcal{E}_p(i_j, j, i_{j'}, j') \cup \{v_p(i_j, j, 1), v_p(i_j, j, 2), v_p(i_{j'}, j', 1), v_p(i_{j'}, j', 2)\}$ would be disjoint from X.

Pathwidth in $\mathcal{O}(k)$. Let $\mathcal{P}'_p$ be the $\mathcal{O}(k)$ vertices of $\mathcal{P}_p$ with strictly more than one neighbor in $H_p \cup H_{p+1}$. For every $p \in [m-1]$, we set $Y_p := \mathcal{P}'_p \cup \mathcal{E}_p(i_p, j_p, i'_p, j'_p) \cup C_{p,j_p} \cup \mathcal{C}_{\text{sel}}(p, j_p) \cup C_{p,j'_p} \cup \mathcal{C}_{\text{sel}}(p, j'_p) \cup \bigcup_{i \in [k]} \mathcal{R}_{\text{sel}}(p,i)$, and we observe that $|Y_p| = \mathcal{O}(k)$ (this is where it is important that each $\mathcal{R}_{\text{sel}}(p,i)$ has constant size). For each $p \in [m]$ and $j \in [k-2]$, let $Z_{p,j}$ be $C_{p,j^*} \cup C_{\text{sel}}(p, j^*)$ where j^* is the j-th index, by increasing value, in $[k] \setminus \{j_p, j'_p\}$. Again we notice that $|Z_{p,j}| = \mathcal{O}(k)$.

Here is a path-decomposition of G of width $\mathcal{O}(k)$ in case every $\mathcal{P}_p \setminus \mathcal{P}'_p$ is empty: $Y_1, Y_1 \cup Z_{1,1}, Y_1 \cup Z_{1,2}, \ldots, Y_1 \cup Z_{1,k-2}, Y_1 \cup Y_2, Y_1 \cup Y_2 \cup Z_{2,1}, Y_1 \cup Y_2 \cup Z_{2,2}, \ldots, Y_1 \cup Y_2 \cup Z_{2,k-2}, Y_2 \cup Y_3, \ldots, Y_{p-2} \cup Y_{p-1}, Y_{p-2} \cup Y_{p-1} \cup Z_{p-1,1}, Y_{p-2} \cup Y_{p-1} \cup Z_{p-1,2}, \ldots, Y_{p-2} \cup Y_{p-1} \cup$

$Z_{p-1,k-2}, Y_{p-1}, Y_{p-1} \cup Z_{p,1}, Y_{p-1} \cup Z_{p,2}, \ldots, Y_{p-1} \cup Z_{p,k-2}$. Indeed the maximum bag size is $\mathcal{O}(k)$ and each edge of G appears in at least one bag. Two crucial properties used in this path-decomposition are that (1) the removal of $\mathcal{P}'_p \cup \mathcal{P}'_{p+1}$, so in particular of $Y_p \cup Y_{p+1}$, disconnects H_{p+1} from the rest of G, and (2) there is no edge between $Z_{p,j}$ and $Z_{p,j'}$ for $j \neq j' \in [k-2]$ and $p \in [m]$.

In the general case, a path-decomposition of width $\mathcal{O}(k)$ for G is obtained from the previous decomposition by observing the following rule. Each time a vertex of H_p appears in a bag for the first time, we introduce and immediately remove each of its neighbors in $\mathcal{P}_p \setminus \mathcal{P}'_p$ one after the other. ◀

3.2 Designing ad hoc gadgets

We now build specific gadgets for Subset Feedback Vertex Set, Subset Odd Cycle Transversal, and Even Cycle Transversal. For these problems, we always use S to denote the prescribed subset of vertices through which no cycle, no odd cycle, or no even cycle should go, respectively.

3.2.1 Column selector gadgets

We begin with the column selector gadget $\mathcal{G}_1(\mathcal{C})$ used for Subset FVS and Subset OCT, followed by the gadget $\mathcal{G}_2(\mathcal{C})$ used for ECT. The column selector gadget $\mathcal{G}_1(\mathcal{C})$ attached to a column $C_{p,j}$ is defined as follows. It comprises $3k$ additional vertices. These $3k$ vertices are all added to S, and they form an independent set. Each of the first k vertices, $d_{p,j}(1,1), \ldots, d_{p,j}(k,1) \in S$, are adjacent to all vertices in $\bigcup_{i\in[k]}\{v_p(i,j,1)\}$, so these vertices induce a biclique. The next k vertices, $d_{p,j}(1,2), \ldots, d_{p,j}(k,2) \in S$, also twins, are adjacent to all vertices in $\bigcup_{i\in[k]}\{v_p(i,j,2)\}$. We add $d_{p,j}(1), \ldots, d_{p,j}(i), \ldots, d_{p,j}(k)$ and, for each $i \in [k]$, we link $d_{p,j}(i)$ to all the vertices in $\{v_p(i,j,1)\} \cup \bigcup_{i'\in[k]\setminus\{i\}}\{v_p(i',j,2)\}$. Finally we make every distinct pair $v_p(i,j,z), v_p(i',j,z')$ adjacent, except if $i = i'$.

We obtain the column selector gadget $\mathcal{G}_2(\mathcal{C})$ from $\mathcal{G}_1(\mathcal{C})$ by adding, for each $z \in [2]$, a vertex $d_{p,j}(k+1,z)$ adjacent to all vertices in $\bigcup_{i\in[k]}\{v_p(i,j,z)\}$, and by subdividing each edge $d_{p,j}(i)v_p(i,j,1)$ once.

▶ **Lemma 4.** *$\mathcal{G}_1(\mathcal{C})$ is a column selector gadget for Subset Feedback Vertex Set and Subset Odd Cycle Transversal, and $\mathcal{G}_2(\mathcal{C})$ is a column selector gadget for Even Cycle Transversal.*

Proof. The gadgets $\mathcal{G}_1(\mathcal{C})$ and $\mathcal{G}_2(\mathcal{C})$ add $3k$ and $4k+2$, respectively, new vertices, thus $\mathcal{O}(k)$. Their edge set respects the specification of the column selector.

We first show that the only Π-legal $(2k-2)$-deletions within $\mathcal{G}_1(\mathcal{C})$ are the sets $C_{p,j} \setminus \{v_p(i,j,1), v_p(i,j,2)\}$ (for $i \in [k]$), for $\Pi \in \{$Subset FVS, Subset OCT$\}$. For every $p \in [m]$, $j \in [k]$, and $z \in [2]$, the biclique $K_{k,k}$ between $\bigcup_{i\in[k]}\{v_p(i,j,z)\}$ and $\bigcup_{i\in[k]}\{d_{p,j}(i,z)\} \subseteq S$ forces the removal of all but at most one vertex of $\bigcup_{i\in[k]}\{v_p(i,j,z)\}$, or all the vertices in $\bigcup_{i\in[k]}\{d_{p,j}(i,z)\}$. Indeed, recall that the former set is a clique, while the latter set is an independent set and is contained in the prescribed set S. Hence keeping at least one vertex in $\bigcup_{i\in[k]}\{d_{p,j}(i,z)\}$ and at least two in $\bigcup_{i\in[k]}\{v_p(i,j,z)\}$ results in an odd cycle (a triangle) going through at least one vertex of S. Thus the only Π-legal $(2k-2)$-deletions within $\mathcal{G}_1(\mathcal{C})$ have to remove exactly $k-1$ vertices in $\bigcup_{i\in[k]}\{v_p(i,j,1)\}$ and exactly $k-1$ vertices in $\bigcup_{i\in[k]}\{v_p(i,j,2)\}$. Let Y denote such a deletion set, and observe that $Y \cap S = \emptyset$. We further claim that if $v_p(i,j,1)$ is not in Y, then $v_p(i,j,2)$ is also not in Y. Assume, for the

sake of contradiction, that $v_p(i,j,1)$ and $v_p(i',j,2)$ are two (adjacent) vertices, not in Y, with $i \neq i'$. Then $d_{p,j}(i) \in S$ forms a surviving triangle with $v_p(i,j,1)$ and $v_p(i',j,2)$. Thus $Y = C_{p,j} \setminus \{v_p(i,j,1), v_p(i,j,2)\}$ for some $i \in [k]$.

This finishes the proof that $\mathcal{G}_1(\mathcal{C})$ is a column selector gadget for Subset FVS and Subset OCT. We now adapt the arguments for $\mathcal{G}_2(\mathcal{C})$ and $\Pi =$ ECT. Now the biclique $K_{k,k+1}$ between $\bigcup_{i\in[k]}\{v_p(i,j,z)\}$ and $\bigcup_{i\in[k+1]}\{d_{p,j}(i,z)\} \subseteq S$ forces the removal of all but at most one vertex of $\bigcup_{i\in[k]}\{v_p(i,j,z)\}$, or all but at most one vertex of $\bigcup_{i\in[k+1]}\{d_{p,j}(i,z)\}$, otherwise there would be a surviving even cycle C_4. Since only $k-1$ vertices can be removed from each Π-obstruction $\bigcup_{i\in[k]}\{v_p(i,j,z)\} \cup \bigcup_{i\in[k+1]}\{d_{p,j}(i,z)\} \subseteq S$ (with $z \in [2]$), the only Π-legal $(2k-2)$-deletions within $\mathcal{G}_2(\mathcal{C})$ remove all but one vertex in $\bigcup_{i\in[k]}\{v_p(i,j,1)\}$ and in $\bigcup_{i\in[k]}\{v_p(i,j,2)\}$. The end of the proof is similar to the previous paragraph since the triangle $d_{p,j}(i)v_p(i,j,1)v_p(i',j,2)$ is now a C_4 (recall that we subdivided the edge $d_{p,j}(i)v_p(i,j,1)$ once). ◀

3.2.2 Row selector gadgets

The row selector $\mathcal{G}_1(\mathcal{R})$, attached to $R_{p,i}$, consists of two additional vertices $r_1(p,i), r'_1(p,i) \in S$ made adjacent to every vertex in $\bigcup_{j\in[k]}\{v_p(i,j,1)\}$. The row selector $\mathcal{G}_2(\mathcal{R})$ consists of three additional vertices $r_2(p,i), r'_2(p,i), r''_2(p,i)$, each adjacent to all vertices in $\bigcup_{j\in[k]}\{v_p(i,j,1)\}$. We put only $r'_2(p,i)$ in S, and we add an edge between $r_2(p,i)$ and $r''_2(p,i)$.

▶ **Lemma 5** (⋆). *$\mathcal{G}_1(\mathcal{R})$ is a row selector gadget for Subset Feedback Vertex Set and Even Cycle Transversal, and $\mathcal{G}_2(\mathcal{R})$ is a row selector gadget for Subset Odd Cycle Transversal.*

Crucially for the intended-solution property, the odd cycle $r_2(p,i)v_p(i,j,1)r''_2(p,i)$ does not contain any vertex of S.

3.2.3 Edge gadgets

Let $\mathcal{G}_1(\mathcal{E})$ be the following edge gadget, that we present for $e_p = (i,j)(i',j')$. We add an edge between $v_p(i,j,1)$ and $v_p(i',j',1)$. We add a vertex s_p adjacent to both $v_p(i,j,1)$ and $v_p(i',j',1)$. We add s_p to the set $S \subseteq V(G)$. The edge gadget $\mathcal{G}_2(\mathcal{E})$ is obtained from $\mathcal{G}_1(\mathcal{E})$ by subdividing the edge $s_pv_p(i',j',1)$ once.

▶ **Lemma 6** (⋆). *$\mathcal{G}_1(\mathcal{E})$ is an edge gadget for Subset Feedback Vertex Set and Subset Odd Cycle Transversal, and $\mathcal{G}_2(\mathcal{E})$ is an edge gadget for Even Cycle Transversal.*

3.2.4 Propagation gadgets

We present $\mathcal{G}_1(\mathcal{P})$, a propagation gadget inserted between H_p and H_{p+1}. We first add an independent set of $2k$ vertices. Among them, the k vertices $r_{p,1}, \ldots, r_{p,k}$ represent the row indices in H_p and H_{p+1}, while the k other vertices $c_{p,1}, \ldots, c_{p,k}$ represent the column indices. We link $r_{p,i}$ to all the vertices in $\bigcup_{j\in[k]}\{v_p(i,j,2)\} \cup \bigcup_{j\in[k]}\{v_{p+1}(i,j,1)\}$. Similarly, we link $c_{p,j}$ to all the vertices in $\bigcup_{i\in[k]}\{v_p(i,j,2)\} \cup \bigcup_{i\in[k]}\{v_{p+1}(i,j,1)\}$. Finally, we add a vertex $c_p \in S$ adjacent to all the vertices $c_{p,1}, \ldots, c_{p,k}$.

The gadget $\mathcal{G}_2(\mathcal{P})$ is defined similarly, except that we subdivide the edge $r_{p,i}v_p(i,j,2)$ once, for each $i,j \in [k]$. Finally the gadget $\mathcal{G}_3(\mathcal{P})$ adds to $\mathcal{G}_2(\mathcal{P})$, a vertex $c'_{p,j}$, for each $j \in [k]$. The vertex $c'_{p,j}$ is linked to $c_{p,j}$ and to c_p.

▶ **Lemma 7** (⋆). *$\mathcal{G}_1(\mathcal{P})$ is a column selector gadget for* Subset Feedback Vertex Set, *$\mathcal{G}_2(\mathcal{P})$ is a column selector gadget for* Subset Odd Cycle Transversal, *and $\mathcal{G}_3(\mathcal{P})$ is a column selector gadget for* Even Cycle Transversal.

3.2.5 Wrap-up

We can now use the above gadgets to establish the following.

▶ **Theorem 8.** *Unless the ETH fails, the following problems cannot be solved in time $2^{o(\mathsf{pw}\log\mathsf{pw})}n^{\mathcal{O}(1)}$ on n-vertex graphs with pathwidth* pw*:*

- *Subset Feedback Vertex Set,*
- *Subset Odd Cycle Transversal, and*
- *Even Cycle Transversal.*

Proof. We need to check that these problems satisfy the preconditions of Theorem 3. Sections 3.2.1 to 3.2.4 and Lemmas 4 to 7 show how to build the four types of gadgets. Which problem uses which version of the gadget is summarized in Table 1. See Figure 1 for a schematic representation of the construction for Subset FVS.

Table 1 The different gadgets used for the different problems.

	column selector	row selector	edge gadget	propagation gadget
Subset FVS	$\mathcal{G}_1(\mathcal{C})$	$\mathcal{G}_1(\mathcal{R})$	$\mathcal{G}_1(\mathcal{E})$	$\mathcal{G}_1(\mathcal{P})$
Subset OCT	$\mathcal{G}_1(\mathcal{C})$	$\mathcal{G}_2(\mathcal{R})$	$\mathcal{G}_1(\mathcal{E})$	$\mathcal{G}_2(\mathcal{P})$
ECT	$\mathcal{G}_2(\mathcal{C})$	$\mathcal{G}_1(\mathcal{R})$	$\mathcal{G}_2(\mathcal{E})$	$\mathcal{G}_3(\mathcal{P})$

Finally we have to check that the problems have the intended-solution property. We shall prove that every set $X := \bigcup_{p\in[m],i\in[k],z\in[2]}\{v_p(i,j_i,z)\}$, with $\{j_1,\ldots,j_k\} = [k]$ and intersecting all the edge gadgets is Π-legal in any graph G obtained by attaching to the base the four types of gadgets with respect to their specification of Section 3.1. The set X is a solution to $\Pi \in \{$Subset FVS, Subset OCT, ECT$\}$, if and only if no 2-connected component (i.e., a block of size at least 3) of $G - X$ is a Π-obstruction. Indeed no cycle can go through a cut-vertex.

We first note that there is no 2-connected component within $\mathcal{G}_1(\mathcal{C})$, $\mathcal{G}_2(\mathcal{C})$, $\mathcal{G}_1(\mathcal{R})$, $\mathcal{G}_1(\mathcal{E})$, $\mathcal{G}_2(\mathcal{E})$ restricted to $G - X$. For the latter two gadgets, this is because, by assumption, X intersects every edge gadget. In a gadget $\mathcal{G}_2(\mathcal{R})$ restricted to $G - X$, there is one 2-connected component, namely a triangle; but none of its vertices belongs to S.

We now observe that every vertex c_p is a cut-vertex in $\mathcal{G}_1(\mathcal{P})$, $\mathcal{G}_2(\mathcal{P})$, and $\mathcal{G}_3(\mathcal{P})$ restricted to $G - X$. So the remaining 2-connected components of $G - X$ are induced cycles C_4 of the form $r_{p,i}v_p(i,j,2)c_{p,j}v_{p+1}(i,j,1)$ when $\mathcal{G}_1(\mathcal{P})$ is used, or induced C_5 when $\mathcal{G}_2(\mathcal{P})$ is used, or triangle and induced cycle C_5 when $\mathcal{G}_3(\mathcal{P})$ is used. In the first two cases, none of the vertices of the cycles belongs to S. In the third case, no cycle is even. This establishes that Subset FVS, Subset OCT, and ECT with their respective combination of gadgets have the intended-solution property. ◀

3.3 Lower bound for Node Multiway cut

For Node Multiway Cut we will also start from the base $\bigcup_{p\in[m]} H_p$ but we will deviate from the gadget specification of Section 3.1. We will "communalize" the selector, edge, and propagation gadgets. That way, we are able to show the claimed lower bound even when the

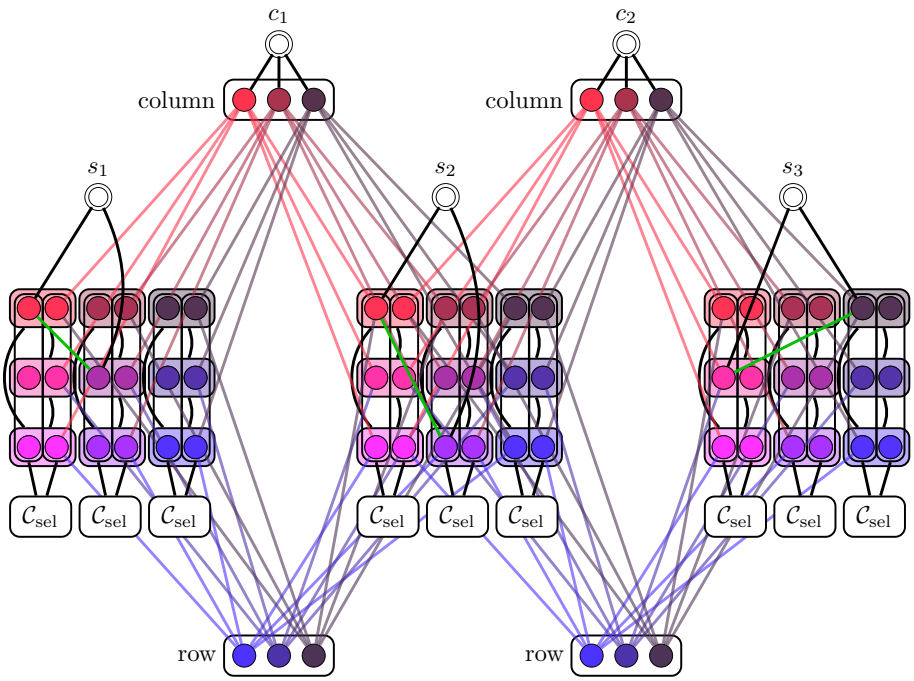


Figure 1 Example of the overall picture for Subset Feedback Vertex Set. The first three edges (in green) in the reduction from $k \times k$-Permutation Independent Set, with $k = 3$, to Subset FVS. The doubly-circled vertices are vertices in S. The column selector gadget $\mathcal{C}_{\text{sel}}$, of size $\mathcal{O}(k)$, forces that only one pair of homologous vertices is retained in each column. We did *not* represent the row selector gadget.

number of terminals is linearly tied to the pathwidth. This is unlike our constructions for Subset FVS and Subset OCT in Theorem 8 where the size of the prescribed subsets S is significantly larger than the pathwidth.

▶ **Theorem 9** (⋆). *Unless the ETH fails, Node Multiway Cut cannot be solved in time* $2^{o(p \log p)} n^{\mathcal{O}(1)}$ *on n-vertex graphs where $p = \mathsf{pw} + |T|$ is the sum of the pathwidth of the input graph and the number of terminals.*

3.4 Lower bound for Multiway Cut

To obtain the lower bound for Multiway Cut, we reduce from $k \times k$-Permutation Clique.

▶ **Theorem 10** (⋆). *Unless the ETH fails, Multiway Cut cannot be solved in time* $2^{o(p \log p)} n^{\mathcal{O}(1)}$ *on n-vertex graphs where $p = \mathsf{pw} + |T|$ is the sum of the pathwidth of the input graph and the number of terminals.*

By a simple reduction from Multiway Cut to Restricted Edge-Subset Feedback Edge Set, we obtain the following as a corollary.

▶ **Theorem 11.** *Unless the ETH fails, Restricted Edge-Subset Feedback Edge Set cannot be solved in time* $2^{o(p \log p)} n^{\mathcal{O}(1)}$ *on n-vertex graphs where $p = \mathsf{pw} + |S|$ is the sum of the pathwidth of the input graph and the number of undeletable (terminal) edges.*

It is not difficult to adapt the construction of Theorem 10 for the directed variant of Multiway Cut.

▶ **Theorem 12.** *Unless the ETH fails, Directed Multiway Cut cannot be solved in time* $2^{o(\mathsf{pw}\log\mathsf{pw})}n^{\mathcal{O}(1)}$ *on* n*-vertex directed graphs whose underlying undirected graph has pathwidth* pw*.*

4 Slightly superexponential algorithms

In this section, we present $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}n^3$-time algorithms for the weighted variants of the considered problems with the exception of ECT. We first present in Theorem 15 a $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}n^3$-time algorithm for Subset OCT. Then, we show that with simple modifications this algorithm can solve Subset FVS. We deduce the algorithms for the other problems by reducing these problems to the weighted variant of Subset FVS.

Let us focus on the Subset OCT problem. For a graph G and a vertex set S of G, we say that G is *S-bipartite* if it has no odd cycle containing a vertex of S. Solving Subset OCT is equivalent to find an S-bipartite induced subgraph of maximum size. The following characterization of S-bipartite graphs will be useful.

▶ **Lemma 13** ($\star$)**.** *A graph G is S-bipartite if and only if for every block B of G, either B has no vertex of S, or it is bipartite.*

One can easily modify the proof of one direction of Lemma 13 to prove the following fact.

▶ **Fact 14.** *If a graph G is 2-connected and not bipartite, then there exists an odd path and an even path between every pair of vertices.*

A tree decomposition $(T, \mathcal{B} = \{B_t\}_{t\in V(T)})$ is a *nice tree decomposition* with root node $r \in V(T)$ if T is a rooted tree with root node r, and every node t of T is one of the following:

1. a *leaf node*: t is a leaf of T and $B_t = \emptyset$;
2. an *introduce node*: t has exactly one child t' and $B_t = B_{t'} \cup \{v\}$ for some $v \in V(G) \setminus B_{t'}$;
3. a *forget node*: t has exactly one child t' and $B_t = B_{t'} \setminus \{v\}$ for some $v \in B_{t'}$; or
4. a *join node*: t has exactly two children t_1 and t_2, and $B_t = B_{t_1} = B_{t_2}$.

▶ **Theorem 15.** *(Weighted) Subset Odd Cycle Transversal can be solved in time* $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}n^3$ *on n-vertex graphs with treewidth* tw*.*

Proof. In the following, we fix a graph G, $S \subseteq V(G)$, and a weight function $w : V(G) \to \mathbb{R}$. Using Bodlaender et al.'s fpt approximation algorithm [4] and an algorithm of constructing a nice tree-decomposition (folklore; see Lemma 7.4 in [10]), we can obtain a nice tree decomposition of G of width at most $5\mathsf{tw}+4$ in time $\mathcal{O}(c^{\mathsf{tw}}\cdot n)$ for some constant c. Let $(T, \{B_t\}_{t\in V(T)})$ be the resulting nice tree decomposition. For each node t of T, let G_t be the subgraph of G induced by the union of all bags $B_{t'}$ where t' is a descendant of t.

Let t be a node of T. A *partial solution of G_t* is a subset $X \subseteq V(G_t)$ such that $G[X]$ is S-bipartite. We are going to introduce an equivalence relation $\equiv_t$ between partial solutions in order to obtain the property that if $X \equiv_t Y$, then for every $W \subseteq V(\overline{G_t})$, $G[X \cup W]$ is S-bipartite if and only if $G[Y \cup W]$ is S-bipartite.

Let $X \subseteq V(G)$ (not necessarily contained in G_t). We denote by $\mathrm{Inc}(X)$ the block-cut tree of $G[X]$, that is the bipartite graph whose vertices are the blocks and the cut vertices of $G[X]$ and where a block B is adjacent to a cut vertex v if $v \in V(B)$. Observe that $\mathrm{Inc}(X)$ is by definition a forest.

We say that a vertex v of $\mathrm{Inc}(X)$ is *active* (with respect to t) if:

- v is a cut vertex of $G[X]$ in B_t,
- v is a block of $G[X]$ that contains at least two vertices in B_t, or
- v is a block of $G[X]$ that contains exactly one vertex in B_t that is not a cut vertex.

Note that every vertex in B_t is an active cut vertex or it is in an active block of $G[X]$.

We construct the auxiliary graphs $\mathrm{Aux}_p(X,t)$ and $\mathrm{Aux}(X,t)$ from $\mathrm{Inc}(X)$ as follows:

1. We remove recursively the leaves and the isolated vertices that are inactive. Let $\mathrm{Aux}_p(X,t)$ be the resulting graph (p for "prototype").
2. For every maximal path P of $\mathrm{Aux}_p(X,t)$ between u and v and with inactive internal vertices of degree 2, we remove the internal vertices of P and we add an edge between u and v (shrinking degree 2 nodes that are inactive).

Figure 2 illustrates the constructions of $\mathrm{Aux}_p(X,t)$ and $\mathrm{Aux}(X,t)$. Observe that Operation 1 removes the inactive blocks of $G[X]$ that contain one vertex in B_t. Thus, every block in $\mathrm{Aux}_p(X,t)$ that contains vertices in B_t is active. By construction, $\mathrm{Aux}(X,t)$ is a forest whose vertices are the active vertices of $\mathrm{Inc}(X)$ and the inactive vertices that have degree at least 3 in $\mathrm{Aux}_p(X,t)$. Importantly, the algorithm uses the graphs $\mathrm{Aux}(X,t)$ for $X \subseteq V(G_t)$ and in the proof we will use $\mathrm{Aux}(X,t)$ and $\mathrm{Aux}_p(X,t)$ for $X \subseteq V(G_t)$ or $X \subseteq B_t \cup V(\overline{G_t})$.

By Step 2, any edge uv of $\mathrm{Aux}(X,t)$ corresponds to an alternating sequence P of cut vertices and blocks $A_1, A_2, \ldots, A_x$ that forms a path from $u = A_1$ to $v = A_x$ in $\mathrm{Inc}(X)$. We define the graph M_{uv} as the union of the blocks in P. Note that one of A_1 and A_2 is a cut vertex and one of A_{x-1} and A_x is a cut vertex. We say that these cut vertices are the endpoints of M_{uv}.

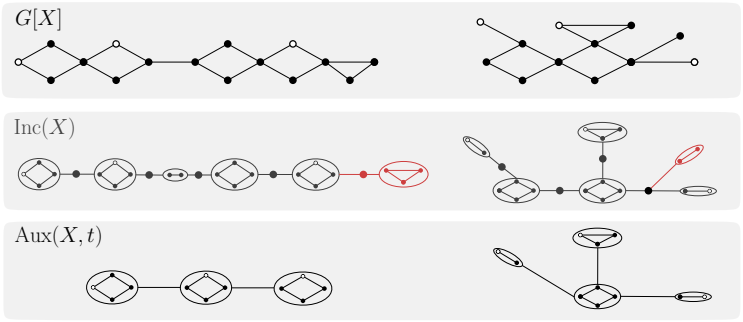


Figure 2 Example of graphs $\mathrm{Inc}(X)$ and $\mathrm{Aux}(X,t)$ constructed from a graph $G[X]$. The vertices in B_t are white filled. The red vertices and edges in $\mathrm{Inc}(X)$ are those we remove to obtain $\mathrm{Aux}_p(X,t)$.

Let X and Y be two partial solutions of G_t. We say that $X \equiv_t Y$ if $X \cap B_t = Y \cap B_t$, and there is an isomorphism φ from $\mathrm{Aux}(X,t)$ to $\mathrm{Aux}(Y,t)$ such that:

1. For every vertex v in $\mathrm{Aux}(X,t)$, v is active if and only if $\varphi(v)$ is active.
2. For every vertex v in $\mathrm{Aux}(X,t)$, v is a block if and only if $\varphi(v)$ is a block.
3. For every active cut vertex v in $\mathrm{Aux}(X,t)$, we have $\varphi(v) = v$.
4. For every active block B in $\mathrm{Aux}(X,t)$:
   a. $V(B) \cap B_t = V(\varphi(B)) \cap B_t$,
   b. $V(B) \cap S \neq \emptyset$ if and only if $V(\varphi(B)) \cap S \neq \emptyset$, and
   c. B is bipartite if and only if $\varphi(B)$ is bipartite.
5. For every edge uv in $\mathrm{Aux}(X,t)$:
   a. M_{uv} is bipartite if and only if $M_{\varphi(u)\varphi(v)}$ is bipartite, and
   b. $V(M_{uv}) \cap S \neq \emptyset$ if and only if $V(M_{\varphi(u)\varphi(v)}) \cap S \neq \emptyset$.
6. For every pair (u,v) of vertices in $B_t \cap X$ and every path P_X between u and v in $G[X]$, there exists a path P_Y in $G[Y]$ between u and v with the same parity as P_X.

▷ Claim 16 (⋆). For every node t of T, $\equiv_t$ has $2^{\mathcal{O}(\mathsf{tw}\log\mathsf{tw})}$ equivalence classes.

▷ Claim 17 (⋆). Let t be a node of T and X, Y be two partial solutions associated with t. If $X \equiv_t Y$, then, for every $Z \subseteq V(\overline{G_t})$, the graph $G[X \cup Z]$ is S-bipartite if and only if $G[Y \cup Z]$ is S-bipartite.

We are now ready to describe our algorithm. For each node t of T and $I \subseteq B_t$, let $\mathcal{P}[t, I]$ be the set of all partial solutions X of G_t where $X \cap B_t = I$. A reduced set $\mathcal{R}[t, I]$ is a subset of $\mathcal{P}[t, I]$ satisfying that

- for every partial solution $X \in \mathcal{P}[t, I]$, there exists $X' \in \mathcal{R}[t, I]$ where $X \equiv_t X'$ and $w(X') \geq w(X)$, and
- no two partial solutions in $\mathcal{R}[t, I]$ are equivalent.

We will recursively compute a reduced set $\mathcal{R}[t, I]$ for every node t of T and $I \subseteq B_t$. Claim 16 guarantees that $|\bigcup_{I \subseteq B_t} \mathcal{R}[t, I]| = 2^{\mathcal{O}(\mathsf{tw} \log \mathsf{tw})}$.

We describe how to compute a reduced set $\mathcal{R}[t, I]$ depending on the type of the node t. We fix a node t and $I \subseteq B_t$. For each leaf node t and $I = \emptyset$, we assign $\mathcal{R}[t, I] := \emptyset$. For $\mathcal{A} \subseteq 2^{V(G_t)}$, we define $\mathsf{reduce}_t(\mathcal{A})$ as the operation which removes the elements of $\mathcal{A}$ that does not induce S-bipartite graph and then returns a set that contains, for each equivalence class $\mathcal{C}$ of $\equiv_t$ over $\mathcal{A}$, a partial solution of $\mathcal{C}$ of maximum weight.

1) t is an introduce node with child t' and $B_t \setminus B_{t'} = \{v\}$:

If $v \notin I$, then it is easy to see that $\mathcal{R}[t', I]$ is a reduced set of $\mathcal{P}[t, I] = \mathcal{P}[t', I]$. In this case, we take $\mathcal{R}[t, I] = \mathcal{R}[t', I]$. Assume now that $v \in I$. We set $\mathcal{R}[t, I] = \mathsf{reduce}_t(\mathcal{A})$ with $\mathcal{A}$ the set that contains $X \cup \{v\}$ for every $X \in \mathcal{R}[t', I \setminus \{v\}]$.

2) t is a forget node with child t' and $B_{t'} \setminus B_t = \{v\}$:

We simply set $\mathcal{R}[t, I] = \mathsf{reduce}_t(\mathcal{R}[t', I] \cup \mathcal{R}[t', I \cup \{v\}])$.

3) t is a join node with two children t_1 and t_2:

We set $\mathcal{R}[t, I] = \mathsf{reduce}_t(\mathcal{A})$ where $\mathcal{A}$ is the set that contains $X_1 \cup X_2$ for every $X_1 \in \mathcal{R}[t_1, I]$ and $X_2 \in \mathcal{R}[t_2, I]$.

We defer the proof of the correctness and the runtime to the long version. ◀

▶ **Theorem 18** (⋆). *Subset Feedback Vertex Set, Restricted Edge-Subset Feedback Edge Set, and Node Multiway Cut, and their weighted variants can be solved in time* $2^{\mathcal{O}(\mathsf{tw} \log \mathsf{tw})} n^3$ *on n-vertex graph with treewidth* tw.

References

1 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting minors on bounded treewidth graphs. *CoRR*, abs/1704.07284, 2017. arXiv:1704.07284.

2 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. A complexity dichotomy for hitting connected minors on bounded treewidth graphs: the chair and the banner draw the boundary. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 951–970. SIAM, 2020. doi:10.1137/1.9781611975994.57.

3 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

4 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi:10.1137/130947374.

5 Marthe Bonamy, Lukasz Kowalik, Jesper Nederlof, Michal Pilipczuk, Arkadiusz Socala, and Marcin Wrochna. On directed feedback vertex set parameterized by treewidth. In *Graph-Theoretic Concepts in Computer Science - 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, pages 65–78, 2018. doi:10.1007/978-3-030-00256-5_6.

6 Édouard Bonnet, Nick Brettell, O-joung Kwon, and Dániel Marx. Generalized feedback vertex set problems on bounded-treewidth graphs: chordality is the key to single-exponential parameterized algorithms. *Algorithmica*, 81(10):3890–3935, 2019. doi:10.1007/s00453-019-00579-4.

7 Hajo Broersma, Petr A. Golovach, and Viresh Patel. Tight complexity bounds for FPT subgraph problems parameterized by the clique-width. *Theor. Comput. Sci.*, 485:69–84, 2013. doi:10.1016/j.tcs.2013.03.008.

8 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

9 Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socala. Tight lower bounds on graph embedding problems. *J. ACM*, 64(3):18:1–18:22, 2017. doi:10.1145/3051094.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159, 2011. doi:10.1109/FOCS.2011.23.

12 Xiaojie Deng, Bingkai Lin, and Chihao Zhang. Multi-multiway cut problem on graphs of bounded branch width. In *Frontiers in Algorithmics* and *Algorithmic Aspects in Information and Management, Third Joint International Conference, FAW-AAIM 2013, Dalian, China, June 26-28, 2013. Proceedings*, pages 315–324, 2013. doi:10.1007/978-3-642-38756-2_32.

13 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/s00453-016-0127-x.

14 Samuel Fiorini, Nadia Hardy, Bruce A. Reed, and Adrian Vetta. Planar graph bipartization in linear time. *Discret. Appl. Math.*, 156(7):1175–1180, 2008. doi:10.1016/j.dam.2007.08.013.

15 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997. doi:10.1007/BF02523685.

16 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

17 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 760–776, 2011. doi:10.1137/1.9781611973082.60.

18 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. *SIAM J. Comput.*, 47(3):675–702, 2018. doi:10.1137/16M1104834.

19 Michal Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, pages 520–531, 2011. doi:10.1007/978-3-642-22993-0_47.

20 Ignasi Sau and Uéverton S. Souza. Hitting forbidden induced subgraphs on bounded treewidth graphs. *CoRR*, abs/2004.08324, 2020. arXiv:2004.08324.

21 Mingyu Xiao and Hiroshi Nagamochi. An FPT algorithm for edge subset feedback edge set. *Inf. Process. Lett.*, 112(1-2):5–9, 2012. doi:10.1016/j.ipl.2011.10.007.

Parameterized Complexity of Scheduling Chains of Jobs with Delays

Hans L. Bodlaender
Department of Information and Computing Sciences, Utrecht University, The Netherlands
H.L.Bodlaender@uu.nl

Marieke van der Wegen
Department of Information and Computing Sciences, Utrecht University, The Netherlands
Mathematical Institute, Utrecht University, The Netherlands
M.vanderWegen@uu.nl

Abstract

In this paper, we consider the parameterized complexity of the following scheduling problem. We must schedule a number of jobs on m machines, where each job has unit length, and the graph of precedence constraints consists of a set of chains. Each precedence constraint is labelled with an integer that denotes the exact (or minimum) delay between the jobs. We study different cases; delays can be given in unary and in binary, and the case that we have a single machine is discussed separately. We consider the complexity of this problem parameterized by the number of chains, and by the thickness of the instance, which is the maximum number of chains whose intervals between release date and deadline overlap.

We show that this scheduling problem with exact delays in unary is $W[t]$-hard for all t, when parameterized by the thickness, even when we have a single machine ($m = 1$). When parameterized by the number of chains, this problem is $W[1]$-complete when we have a single or a constant number of machines, and $W[2]$-complete when the number of machines is a variable. The problem with minimum delays, given in unary, parameterized by the number of chains (and as a simple corollary, also when parameterized by the thickness) is $W[1]$-hard for a single or a constant number of machines, and $W[2]$-hard when the number of machines is variable.

With a dynamic programming algorithm, one can show membership in XP for exact and minimum delays in unary, for any number of machines, when parameterized by thickness or number of chains. For a single machine, with exact delays in binary, parameterized by the number of chains, membership in XP can be shown with branching and solving a system of difference constraints. For all other cases for delays in binary, membership in XP is open.

2012 ACM Subject Classification Computing methodologies → Planning and scheduling; Theory of computation → Fixed parameter tractability; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Scheduling, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.4

Related Version Missing proofs can be found at the full version of this paper, which can be found at arXiv [4] (`https://arxiv.org/abs/2007.09023`).

Acknowledgements We would like to thank Sukanya Pandey for helpful discussions.

1 Introduction

In this paper, we study a problem in the field of parameterized complexity of scheduling problems. Here, we look at scheduling jobs with precedence constraints with exact or minimum delays, and assume that all jobs have unit length. We study one of the simplest types of precedence constraint graphs: we assume that the precedences form a collection of disjoint chains. Chains have a release date and deadline. It is not hard to see (by a simple

reduction from 3-Partition) that this problem is NP-hard, even when all delays are 0. In this paper, we study the parameterized complexity of the problem, and look at two different parameters: the number of chains, and the thickness of the instance – that is, the maximum number of chains that have overlapping intervals from release time to deadline. We look at different variants: a constraint gives an exact bound or a lower bound on the delay between successive jobs; we can have one, a constant, or a variable number of machines, and the delays can be encoded in unary or binary. If delays are given in unary, then each of the studied variants belongs to XP, and is hard for $W[1]$ (or classes higher in the W-hierarchy). For one variation (see below), we also show membership in XP when delays are given in binary. We call the studied problems Chain Scheduling with Exact Delays and Chain Scheduling with Minimum Delays, for details see Section 2.

1.1 Related literature

Looking at variants of scheduling problems with special attention to parameters (like the number of available machines) is a common approach in the rich field of study of scheduling problems. Studying such parameterizations using techniques and terminology from the field of parameterized algorithms and complexity was pioneered in 1995 [3], but recently receives growing attention, e.g. [2, 11, 14].

The scheduling of chains of jobs (without delays) was studied by Woeginger [17] and Bruckner [5], who gave respectively a 2-approximation algorithm, and a linear time algorithm for two machines. General precedence graphs with delays between jobs were studied already in 1992 by Wikum et al. [16]; this was followed by a large body of literature, studying different variations and approaches, including theoretical and experimental studies.

Cieliebak et al. [6] considered scheduling jobs with release dates and deadlines, under several parameterizations, one being the height, which is similar to the parameter called thickness in this paper – the height of an instance is the maximum number of jobs that have mutually overlapping intervals from release date to deadline. Several additional, and stronger results on this problem were obtained by van Bevern et al. [15], including a proof that for each fixed looseness $\lambda > 0$, the problem to schedule jobs with release dates and deadlines on a given number of machines is $W[1]$-hard, where λ is the maximum ratio between the difference of release date and deadline and the duration of jobs. This latter result is related to ours: where we have $W[t]$-hardness for all t or $W[2]$-hardness for chains of jobs with exact or minimum delays, respectively, parameterized by thickness, the result by van Bevern et al. [15] gives $W[1]$-hardness for single jobs, i.e., chains of size one.

A special case of chains of jobs is the case where we have coupled jobs, i.e., each chain consists of two jobs. Bessy and Giroudeau [2] consider the parameterized complexity of scheduled jobs with compatibility constraints, where jobs can be executed in the idle time of another pair of coupled jobs when they are compatible.

A survey of parameterized algorithms for scheduling with a number of interesting open problems was given by Mnich and van Bevern [13].

1.2 Our results

In this paper, we give a number of hardness results, which are summarized in Table 1. All variants are already hard when the (exact or minimum) delays are given in unary. We also give the following algorithmic results:

- With a dynamic programming algorithm, one can show that the Chain Scheduling with Exact Delays and Chain Scheduling with Minimum Delays belong to XP, when delays are given in unary, and parameterized by either thickness or number of

Table 1 Hardness results for different variants of the problem. Exact and minimum delays are given in unary. (*) = $W[1]$-hardness follows from [15].

	parameter	exact delays	minimum delays
Single machine	thickness	$W[t]$-hard for all t	$W[1]$-hard
	chains	$W[1]$-complete	$W[1]$-hard
Constant number of machines	thickness	$W[t]$-hard for all t	$W[1]$-hard
	chains	$W[1]$-complete	$W[1]$-hard
Variable number of machines	thickness	$W[t]$-hard for all t (*)	$W[2]$-hard (*)
	chains	$W[2]$-complete	$W[2]$-hard

chains, for any number of machines. Our algorithm is similar to an algorithm by Cieliebak et al. [6], for a related problem – they consider the problem where each job has a release date and deadline, and show that this problem belongs to XP when we use the maximum number of jobs whose intervals between release date and deadline mutually overlap.

- Combining branching with solving a set of difference constraints shows XP-membership of Chain Scheduling with Exact Delays when parameterized by the number of chains, for the case of one machine, when delays are given in binary.

For all other cases, the membership in XP when delays are given in binary is open.

1.3 Organization of this paper

In Section 2, we give a number of preliminary definitions. Section 3 gives hardness proofs for Chain Scheduling with Exact Delays when parameterized by the thickness. The complexity of Chain Scheduling with Exact Delays parameterized by the number of chains is established in Section 4; a relatively simple modification then gives hardness for the corresponding problems with minimum delays. Section 5 gives our algorithmic results (membership in XP). Some conclusions are given in Section 6.

2 Preliminaries

We first describe the problems we study in more details. We have a number of identical machines m. In the paper, we study separately the cases that we have a single machine ($m = 1$), the number of machines is some fixed constant, or the number of machines is variable.

On these machines, we must schedule n jobs. Each job has unit length. On the set of jobs, we have a collection of precedence constraints. Each precedence constraint is an ordered pair of jobs (j, j'): it tells that job j' cannot be started before job j is completed. We say that j is a *direct predecessor* of j', and j is a *predecessor* of j' if there is a directed path from j to j' in the graph formed by the precedence constraints; j' then is a *successor* of j.

The precedence constraints have associated with them a *delay*, denoted $l_{j,j'}$: each delay is a non-negative integer. We study two variations of the problem: exact delays and minimum delays. If we consider exact (resp. minimum) delays, then if a constraint (j, j') has delay $l_{j,j'}$, then job j' must be started exactly (resp. at least) $l_{j,j'}$ time steps after job j was finished. That is: when job j starts at time t, then job j' starts at time exactly (resp. at least) $t + l_{j,j'} + 1$. (Note that jobs run directly after each other, only if the delay is 0.) It is allowed to schedule a job on a different machine than its predecessor – thus, we do not need to specify on which machine a job is running, but only ensure that at each time step, the number of scheduled jobs is at most the number of available machines.

In this paper, we consider the case that the graph of the precedence constraints consists of a set of chains. I.e., each job has at most one direct predecessor and at most one direct successor. Chains are the maximal sets of jobs that are predecessors or successors of each other.

Each chain C has a *release date* r_C and a deadline d_C. We have that the first job in the chain cannot start before time r_C and the last job in the chain should be completed at or before time d_C.

In this paper, we consider the following two parameterizations of the problem. The first is the number of chains, denoted by c. The second is the *thickness*, denoted by τ, defined as follows. We say that two chains *overlap*, when their intervals $[r_C, d_C)$ have a non-empty intersection. We define the thickness τ to be the maximum size of a collection of chains that mutually overlap. That is, for any time t, there are at most τ chains C for which we have that $r_C \leq t$ and $d_C > t$.

Chain Scheduling with Exact Delays is the problem where we are given as input the set of jobs with chains of precedence constraints, delays for each precedence constraint, release dates and deadlines of chains, and number of machines, and ask whether there exists a schedule that fulfills all the demands: at each time step, the number of jobs scheduled is at most the number of machines; jobs in a chain are not scheduled before the release date or after the deadline, and for each precedence constraint (j, j') the delay between j and j' is exactly $l_{j,j'}$.

As said, we study several variants of this problem: delays can be given in unary or binary, the number of machines can be 1, fixed or variable, and we can parameterize by the number of chains or by thickness. If we require that the stated delays are lower bounds, we obtain the Chain Scheduling with Minimum Delays problem: here, when we have a precedence constraint (j, j') with delay $l_{j,j'}$, we must have that job j' starts at least $l_{j,j'}$ time steps after job j is finished.

For the $W[t]$-hardness proofs, we use fpt-reductions from the following version of the Satisfiability problem. A Boolean formula is said to be *t-normalized*, if it is the conjunction of the disjunction of the conjunction of ... of literals, with t alternations of AND's and OR's.

The following parameterized problem was considered by Downey and Fellows [8].

Weighted t-Normalized Satisfiability
Given: A t-normalized Boolean formula F and a positive integer $k \in \mathbb{N}$.
Parameter: k
Question: Can F be satisfied by setting exactly k variables to true?

▶ **Theorem 2.1** (Downey and Fellows [8, 9]). *For every $t \geq 2$,* Weighted t-Normalized Satisfiability *is $W[t]$-complete.*

For the $W[1]$- and $W[2]$-completeness results, we use reductions from Independent Set and Dominating Set. It is know that Independent Set is $W[1]$-complete [9] and Dominating Set is $W[2]$-complete [8].

3 Parameterization by thickness

In this section, we look at the Chain Scheduling with Exact Delays problem, when parameterized by the thickness τ. We will show, for several variations, that the problem is hard for the class $W[t]$, for all integers t.

3.1 Parallel machines

We consider the version with m parallel machines, where m is part of the input. The delays are assumed to be exact.

We will give a reduction from Weighted t-Normalized Satisfiablity. Assume we have a t-normalized Boolean formula F and an integer k. Let t' be the number of "levels" of disjunction. Notice that $t' = \lfloor t/2 \rfloor$. We assume the variables of F to be $x_0, \ldots, x_{n-1}$.

We make an instance of the Chain Scheduling with Exact Delays problem, with $m = k + t'$ machines and thickness $\tau = 2k + t'$.

An *element* of the formula is either a literal, or a disjunction or conjunction of smaller elements. We will first assign to each element F' an *interval size* $s(F')$, and then we assign to each element F' an integer interval with length $s(F')$.

The *interval size* of a literal (i.e., a formula of the form x_i or $\neg x_i$) is $2n$. The interval size of a conjunction is the sum of the size of the terms, i.e., $s(F_1 \wedge F_2 \wedge \cdots \wedge F_q) = \sum_{i=1}^{q} s(F_i)$. For each disjunction F' of q terms, its interval size is $2q + 1$ times the maximum size of its terms: define $s_{\max}(F') = \max_{1 \leq i \leq q} s(F_i)$, and then $s(F_1 \vee F_2 \vee \cdots \vee F_q) = (2q+1) \cdot s_{\max}(F')$.

To each element F' of F we assign an integer interval $[\ell(F'), r(F')]$ with $s(F') = r(F') - \ell(F')$. We will do this top-down: first we assign an interval to F, then we define a subinterval for every term of F, etc.

To F, we assign the interval $[n, n + s(F)]$. To elements of a conjunction and disjunction, we assign subintervals of the intervals assigned to the conjunction of disjunction, in such a way that these intervals have the same nesting as the elements in the formula.

Consider an element F' that is the conjunction $F_1 \wedge F_2 \wedge \cdots \wedge F_q$. Then assign F_1 the interval $[\ell(F'), \ell(F') + s(F_1)]$; F_2 the interval $[\ell(F') + s(F_1), \ell(F') + s(F_1) + s(F_2)]$, etc. I.e., F_i is assigned the interval $[\ell(F') + \sum_{j=1}^{i-1} s(F_j), \ell(F') + \sum_{j=1}^{i} s(F_j)]$.

Suppose an element F' is the disjunction $F_1 \vee F_2 \vee \cdots \vee F_q$. The construction is similar to that of conjunctions, but now we assign each term the same length interval and keep unused intervals between the terms. Recall that $s_{\max}(F') = \max_{1 \leq i \leq q} s(F_i)$. Assign to F_i the interval $[\ell(F') + (2i - 1) \cdot s_{\max}(F'), \ell(F') + 2i \cdot s_{max}(F')]$.

Note the nesting of intervals, and that we assigned to each element an interval equal to its size. Also note that we can compute all intervals and sizes in polynomial time in the size of the input instance.

We now can describe the jobs, precedence constraints, and release dates and deadlines.

For each i, $1 \leq i \leq k$, we start a chain c_i. Each of those chains starts with a job and then a delay of $n - 1$. The first job of the chain is released at time 0. We will add jobs and specify delays between jobs in the chain such that the total processing time including the delay times is $n + s(F) + 1$. Set the deadline of those chains to $2n + s(F)$, so that the first job can start at times $0, 1, \ldots, n - 1$.

These chains reflect the variables that are set to true; more precisely, when the first job of one of the chains starts at a time i, then this corresponds to setting x_i to true. We call these the *true variable chains*.

To prevent two chains selecting the same variable, we add $m - 1$ chains, each with n jobs with delay 0, release date 0 and deadline n. We call those chains *fill chains*. Those chains have to be scheduled from time 0 until time n. This implies that at each time $0, 1, \ldots, n - 1$ at most one other job can be scheduled, thus at most one true variable chain starts. Hence, the true variable chains select exactly k variable to be set to true.

We will now extend the true variable chains. Consider the timesteps in the interval $[n, n + s(F)]$ from left to right.

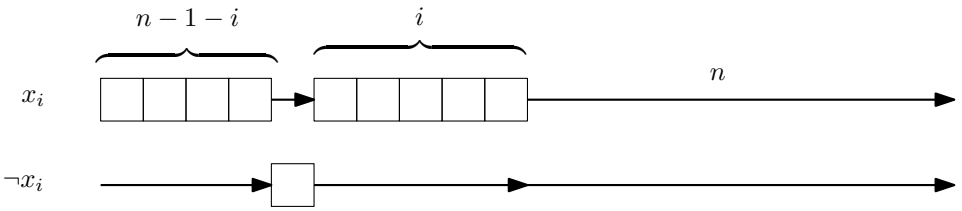


Figure 1 The variable gadgets.

- For each timestep that we encounter that is not part of an interval that corresponds to a literal, we add a delay of 1 at the end of the chain.
- For each interval $[\ell(F'), r(F')]$ that corresponds to a positive literal x_i, we add the following gadget to the chain: $n-1-i$ jobs with delay 0, then a delay of 1, then i jobs with delay 0 and then a delay of n. (See Figure 1.) Notice that no job is scheduled from $\ell(F')+n-1$ until $\ell(F')+n$ if the chain starts at time i, and there is a job scheduled at this time otherwise.
- For each element F' of F that is a negative literal $\neg x_i$, we make the following gadget: a delay of $n-1-i$, a job, then a delay of i, and then a delay of n. (See Figure 1.) Notice that a job is scheduled from $\ell(F')+n-1$ until $\ell(F')+n$ if the chain starts at time i, and there is no job scheduled at this time otherwise.

Add one job at the end of the chain. Notice that the total processing time of those chains is indeed $n+s(F)+1$.

To check whether variables are true, we add some chains that consist of a single job. We call those chains *variable check chains*.

- For each element F' of F that consists of a single positive literal (i.e., is of the form x_i), we make a chain with one job, that is released at time $\ell(F')+n-1$ and has deadline $\ell(F')+n$.
- For each element F' of F that consists of a single negative literal (i.e., is of the form $\neg x_i$), we make k chains with one job, release date $\ell(F')+n-1$ and deadline $\ell(F')+n$.

The intuition behind this construction is as follows: suppose that we have k machines. For each element F' of F that is of the form x_i, there is one job scheduled from $\ell(F')+n-1$ until $\ell(F')+n$. So for at least one of the true variable chains, we need that no job of this chain to be scheduled from $\ell(F')+n-1$ until $\ell(F')+n$. This means that one of the true variable chains starts at time i. For each element F' of F that is of the form $\neg x_i$, there are k jobs scheduled from $\ell(F')+n-1$ until $\ell(F')+n$. So for none of the true variable chains we can schedule a job of this chain from $\ell(F')+n-1$ until $\ell(F')+n$. This means that none of the true variable chains starts at time i. The other t' machines take care of the disjunctions.

For each element $F'=F_1\vee F_2\vee\cdots\vee F_q$ of F that is a disjunction, we make one chain. This chain has $3\cdot s_{\max}(F')$ jobs with delay 0. The chain will be released at time $\ell(F')$ and has deadline $r(F')$. We call those chains *disjunction chains*. Notice that for every element F' of F that is a literal, there are exactly t' disjunction chains that overlap the interval $[l(F'), r(F')]$, that is, there are exactly t' disjunction chains C with release time at most $l(F')$ and deadline at least $r(F')$.

We now have specified all jobs and the machines they run on. Note that the thickness of this construction is at most $2k+t'$.

▶ **Example 3.1.** Figure 2 shows an example of this construction for the formula $(x_0\vee\neg x_1\vee x_2)\wedge(\neg x_0\vee x_1)$ and $k=1$. The intervals that correspond to the literals are indicated. The interval that corresponds to the first disjunction $x_0\vee\neg x_1\vee x_2$ is $[3,45]$, and the interval

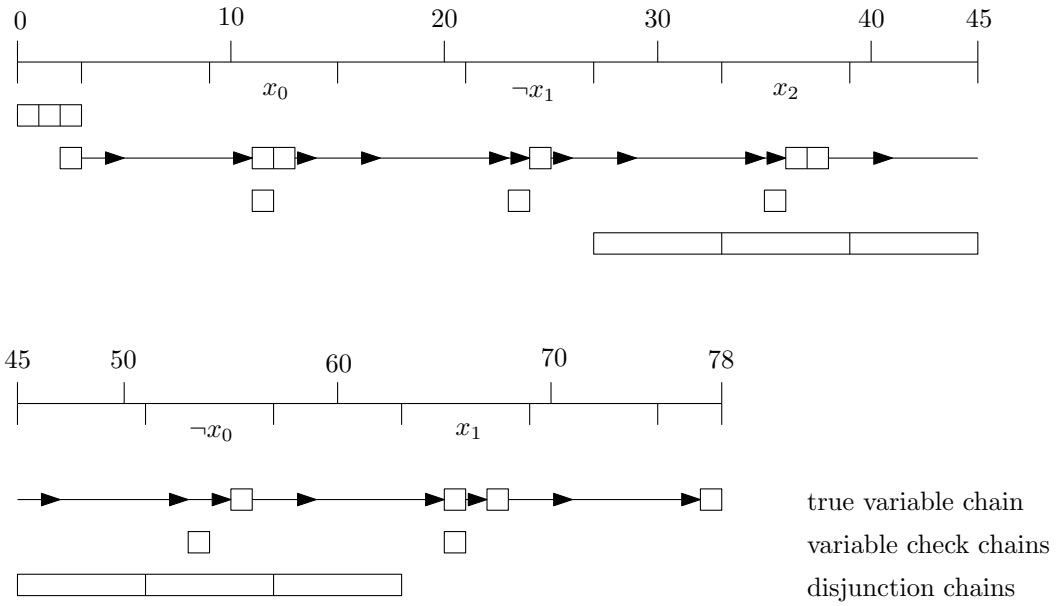


Figure 2 An example of the construction of Section 3 for the formula $(x_0 \vee \neg x_1 \vee x_2) \wedge (\neg x_0 \vee x_1)$ and $k = 1$. A feasible solution is shown which corresponds to setting x_2 to true.

$[45, 75]$ is assigned to the second disjunction $\neg x_0 \vee x_1$. The figure shows a feasible solution of the scheduling problem, that corresponds to setting x_2 to true, and x_0 and x_1 to false. The term x_2 satisfies the first disjunction, and the term $\neg x_0$ satisfies the second disjunction.

▷ Claim 3.2. If F is satisfiable by setting exactly k variables to true, then the constructed instance of the Chain Scheduling with Exact Delays problem has a solution.

Proof. Suppose F is satisfiable by making variables $x_{i_1}, \ldots, x_{i_k}$ true. For each j, with $1 \leq j \leq k$, we let one true variable chain start at time i_j.

First we introduce a notion *satisfying*, intuitively, this will be the elements that make F true. We define this top-down. First we call F satisfying. For each element F' of F: for a satisfying conjunction, all its terms are satisfying, for a satisfying disjunction, at least one of its terms is satisfied, we fix one of them and call it satisfying.

For each disjunction $F' = F_1 \vee \cdots \vee F_q$, consider the disjunction chain C associated with this element. If F' is not satisfying, then start this chain C arbitrarily, say at its release time. If F' is satisfying, let F_j be its term that is satisfying. Now, start this chain C at time $\ell(F') + (2j - 2)s_{\max}(F')$, where $s_{\max}(F')$ is again the maximum over the terms F_i of their interval size $s(F_i)$.

To check that this is a feasible schedule, we have to check that we never use more than m machines. For most timesteps this is straightforward, see [4] for the details. The interesting timesteps are the timesteps from $l(F') + n - 1$ to $l(F') + n$ for some satisfying literal, since at those times a variable check chain and t' disjunction chains are scheduled. Since F is true and F' is satisfying, we know that the literal F' is set to true (resp. false). It follows that the corresponding true variable chain has no job starting at time $l(F') + n - 1$. Feasiblity follows, for details see the full version [4]. ◁

▷ Claim 3.3 (See [4]). Suppose the Chain Scheduling with Exact Delays problem has a solution, then F can be satisfied by setting exactly k variables to true.

We now have shown:

▶ **Theorem 3.4.** *The* Chain Scheduling with Exact Delays *problem, parameterized by the thickness τ, is $W[t]$-hard for all $t \in \mathbb{N}$.*

3.2 Single machine

Again, assume the delays are exact. We now show that the problem stays hard when there is only one machine.

▶ **Theorem 3.5.** *The* Chain Scheduling with Exact Delays *problem, parameterized by the thickness τ, is $W[t]$-hard for all $t \in \mathbb{N}$, when only 1 machine is available.*

Let τ be the thickness of the original instance. Notice that we may assume that $\tau \geq m$ without loss of generality. The main idea of the reduction is to replace each time step on m machines by τ time steps on a single machine. Every chain will be assigned a number $i \in \{0, 1, \ldots, \tau - 1\}$ and its jobs will be scheduled at times $i \pmod{\tau}$, this will make sure that at every timestep only one job is scheduled. For every interval $[i\tau, (i+1)\tau]$, we will have $\tau - m$ chains that have one job; this ensures that in that interval at most m jobs of an original chain are scheduled. We now proceed with the formal description.

We reduce from the case with m machines (Theorem 3.4). Suppose we have an instance with m machines. For every interval $[i\tau, (i+1)\tau]$, we add $\tau - m$ additional chains, with a single job, release date $i\tau$ and deadline $(i+1)\tau$. We call those chains *extra*.

We copy the chains from the given instance, except that:

- If a chain has release date α, then it now has release date $\alpha \cdot \tau$.
- If a chain has deadline β, then it now has deadline $\beta \cdot \tau$.
- Every delay d is replaced by a delay $\tau d + \tau - 1$.

We call these chains *regular*.

▷ Claim 3.6. Suppose we have a solution for the constructed instance with one machine. Then we have a solution for the original instance with m machines.

Proof. For each regular chain, let it start at time $\lfloor t/\tau \rfloor$, when its copy in the constructed instance starts at time t. This implies that for every job in the chain it will start at time $\lfloor t/\tau \rfloor$, when its copy in the constructed instance starts at time t. We know that for each time interval $[t\tau, (t+1)\tau]$, $\tau - m$ steps are used for extra jobs, so m time steps are available for jobs of regular chains. Thus, at every time step t in the original instance, at most m jobs are scheduled. ◁

▷ Claim 3.7. Suppose we have a solution for the original instance with m machines. Then we have a solution for the constructed instance with 1 machine.

Proof. We start with assigning a number $0, 1, \ldots, \tau - 1$ to every chain such that for every time step all the chains that overlap this time step have different number. We denote this assignment by c. We can do this as follows: go through time $0, 1, 2, \ldots$, and every time a chain is released, assign a number that is currently unused, if a deadline passes, the number of the corresponding chain becomes available again. We can do this with τ numbers, since by definition for every timestep there are at most τ chains that overlap this timestep.

For every regular chain C, let C start at time $t\tau + c(C)$, where t is the starting time of the corresponding original chain. Then chains of thickness number $c(C)$ only have jobs starting at times t with $t \equiv c(C) \pmod{\tau}$. For every time t, all jobs that are scheduled to start

at time t in the original instance, will now be scheduled in the interval $[t\tau, (t+1)\tau]$ in the constructed instance. Those jobs are in different chains and those chains are assigned different numbers. Thus those jobs are scheduled at different times in the constructed instance.

In the original instance, at each time t at most m chains have a job scheduled to start at t, so, in the constructed instance, for each interval $[t\tau, (t+1)\tau]$, at most m regular chains have a job scheduled in this interval. Thus, we can schedule the $\tau - m$ extra chains in this time interval at the time steps where no job of a regular chain is scheduled. ◁

As the reduction can be carried out in polynomial time, Theorem 3.5 follows from the reduction and Theorem 3.4.

3.3 Constant number of parallel machines

We can easily reduce the single machine instance to an instance with a constant number m of parallel machines. Let T be the maximum deadline of all chains. Introduce $m-1$ new chains with T jobs each and 0 delays. The $m-1$ new machines will be processing those $m-1$ new chains, while the original machine processes the original chains. We conclude the following result.

▶ **Theorem 3.8.** *The* Chain Scheduling with Exact Delays *problem with a fixed number of machines, parameterized by the thickness τ, is $W[t]$-hard for all $t \in \mathbb{N}$.*

3.4 Minimum delays

The proof above seems not to be modifiable to the Chain Scheduling with Minimum Delays problem. In Section 4.4, we show that Chain Scheduling with Minimum Delays, parameterized by the number of chains is $W[1]$-hard when $m = 1$, and $W[2]$-hard when the number of machines m is variable. As the thickness is at most the number of chains, it follows that Chain Scheduling with Minimum Delays, parameterized by the thickness is $W[1]$-hard for one machine, and $W[2]$-hard for a variable number of machines. Membership in $W[1]$ or $W[2]$ is open, however.

4 Parameterization by the number of chains

We now give the complexity results when we use the number of chains as the parameter. In Sections 4.1, 4.2, and 4.3, we consider Chain Scheduling with Exact Delays, with the number of machines respectively 1, a constant, or variable. In Section 4.4, we consider Chain Scheduling with Exact Delays.

4.1 Single machine

In this section, we consider the variant where the number of chains c is a parameter, and there is one machine available. We assume that the delays are exact.

▶ **Theorem 4.1.** *The* Chain Scheduling with Exact Delays *problem, parameterized by the number of chains c is $W[1]$-complete, when there is one machine.*

Theorem 4.1 is proven by two reductions: from and to Independent Set with standard parameterization.

▶ **Lemma 4.2.** *The* Chain Scheduling with Exact Delays *problem with one machine, parameterized by the number of chains c, is $W[1]$-hard.*

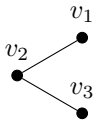


Figure 3 An example graph G. If we pick $p = 3$, the corresponding Golomb ruler is $\{0, 7, 13\}$.

Proof. A set of integers S is said to be a *Golomb ruler* if all differences $a - b$ of two elements $a, b \in S$ are unique, that is, $s_1 - s_2 \neq s_3 - s_4$ for $s_1, s_2, s_3, s_4 \in S$ with $s_1 \neq s_2$ and $s_3 \neq s_4$.

Erdös and Turán [10] gave the following explicit construction of a Golumb ruler. Let $p > 2$ be a prime number. Then the set $\{2pk + (k^2 \bmod p) \mid k \in \{0, 1, \ldots, p-1\}\}$ is a Golomb ruler with p elements. We can build a Golumb ruler of size n in $O(n\sqrt{n})$ time: with help of the Sieve of Eratosthenes, we find a prime number p between n and $2n$ (such a number always exist, by the classic postulate of Bertrand (see [1])), and then follow the Erdös-Turán-construction with this value of p, and take the first n elements of this set. Notice that the elements in this set are smaller than $4n^2$.

Suppose we have an input of INDEPENDENT SET $G = (V, E)$ and k. Assume $V = \{v_1, \ldots, v_n\}$ and let m be the number of edges. First, build a Golumb ruler S^n of size n. Denote the elements by $s_1 \leq s_2 \leq \cdots \leq s_n$. Notice that $s_1 = 0$. Write $c_0 = s_n + 1$.

We will construct an instance of CHAIN SCHEDULING WITH EXACT DELAYS.

We will make $k + 1$ chains. We call one chain the *start time forcing chain*, the other k chains are the *vertex selection chains*.

The start time forcing chain has release date 1, deadline c_0 and total execution time (including delays) $c_0 - 1$. I.e., it must start at time 1. The chain will have a job starting at every time in $[1, c_0 - 1]$ except the times $s_2, s_3, \ldots, s_n$, there, it has a delay of 1.

The vertex selection chains have release date 0. They have deadline $c_0 + T - 1$ and total execution time T, where $T = (m \cdot k(k-1))(2c_0 + 1) + 1$. So, they can start at times $0, 1, \ldots, c_0 - 1$. The vertex selection chains start with a job and then a delay of $c_0 - 1$. Note that as a result of this, in order not to conflict with the start time forcing chain, they have to start at an element of S^n.

Now, for each edge $v_i, v_j \in E$, and each ordered pair of vertex selection chains C_a, C_b with $a, b \in \{1, 2, \ldots, k\}$ we dedicate an interval I_{v_i, v_j, C_a, C_b} of $2c_0 + 1$ time steps. More precisely, we have $m \cdot k(k-1)$ intervals $[c_0 + \alpha(2c_0 + 1), c_0 + (\alpha + 1)(2c_0 + 1)]$ for $\alpha = 0, 1, \ldots, m \cdot k(k-1) - 1$. And to each interval we assign a unique label I_{v_i, v_j, C_a, C_b} where $v_i v_j \in E$ and $a, b \in \{1, 2, \ldots, k\}$. In the interval I_{v_i, v_j, C_a, C_b} we will check whether the chains C_a and C_b did not select the edge $v_i v_j$, that is, whether C_a does not start at s_i or C_b does not start at s_j.

We will now extend the vertex selection chains. Consider the interval $[c_0, c_0 + (m \cdot k(k-1))(2c_0 + 1)]$ from left to right. For each interval I_{v_i, v_j, C_a, C_b} that we encounter, we extend the vertex selection chains as follows.

- For each chain C, with $C \neq C_a$, $C \neq C_b$, add a delay of $2c_0 + 1$.
- Add the following gadget to the chain C_a: a delay of $c_0 - s_i$, a job, and then a delay of $c_0 + s_i$.
- Add the following gadget to the chain C_b: a delay of $c_0 - s_j$, a job, and then a delay of $c_0 + s_j$.

Add one job at the end of all chains. See Figures 3, 4, and 5 for an example of the construction and a feasible schedule.

We claim that there is a feasible schedule, if and only if G has an independent set of size at least k.

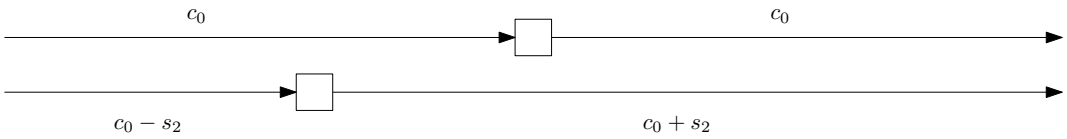


Figure 4 The part of the chains C_a and C_b for the interval I_{v_1,v_2,C_a,C_b} for the graph in Figure 3.

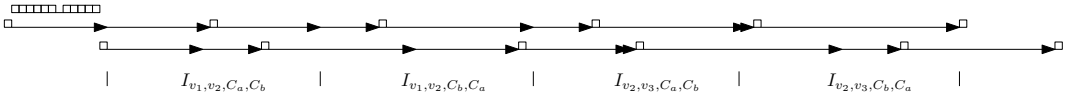


Figure 5 The instance of the scheduling problem constructed from the graph in Figure 3 and $k = 2$, and a feasible schedule for this instance.

▷ Claim 4.3. If G has an independent set of size at least k, then there is a feasible schedule.

Proof. Suppose $v_{i_1}, \ldots, v_{i_k}$ form an independent set in G. Let the vertex selection chain C_a start at times s_{i_a} for $a = 1, 2, \ldots, k$.

Notice that for the first c_0 time steps, at most one machine is used. The same holds for the last c_0 time steps. We show that this is a feasible scheduling by contradiction. Suppose that there are two machines needed at some time step β, and that β is in the interval $I = I_{v_i,v_j,C_a,C_b}$ for checking whether the chains C_a and C_b do not select the edge v_iv_j. Write $I = [\ell(I), r(I)]$. Notice that the job of C_a in the interval I starts at time $\ell(I) + c_0 - s_i + s_{i_a}$. Furthermore, the job of C_b in the interval I starts at time $\ell(I) + c_0 - s_j + s_{i_b}$. Thus, $\ell(I) + c_0 - s_i + s_{i_a} = \beta = \ell(I) + c_0 - s_j + s_{i_b}$. Equivalently, $s_{i_a} - s_i = s_{i_b} - s_j$. Since S^n is a Golomb ruler, it follows that either $s_{i_a} = s_i$ and $s_{i_b} = s_j$ or $s_{i_a} = s_{i_b}$ and $s_i = s_j$.

In the first case, $s_{i_a} = s_i$ and $s_{i_b} = s_j$, we see that $v_i = v_{i_a}$ and $v_j = v_{i_b}$. But there is no edge $v_{i_a}v_{i_b}$, since v_{i_a} and v_{i_b} are in an independent set. This yields a contradiction.

In the second case, $s_{i_a} = s_{i_b}$ and $s_i = s_j$, we see that $v_{i_a} = v_{i_b}$, but this yields a contradiction with the fact that the vertices of the independent set are distinct.

We conclude that this schedule always uses at most one machine. ◁

▷ Claim 4.4. If there is a feasible schedule, then G has an independent set of size at least k.

Proof. Suppose that there is a feasible schedule. Notice that the time of the first step of a vertex selection chains must be an element of S^n, otherwise the job conflicts with the start time forcing chain. Now, set $W = \{v_i \mid$ there is a vertex selection chain that starts at time $s_i\}$. Notice that $|W| = k$, as otherwise two vertex selection chains start at the same time, and conflict with each other for their first job.

We prove that W is an independent set by contradiction. Suppose that there exists an edge v_iv_j, with $v_i, v_j \in W$. Let C_a be the chain that starts at s_i and C_b the chain that starts at s_j. Now consider the interval $I = I_{v_i,v_j,C_a,C_b}$, and write $I = [\ell(I), r(I)]$. By the construction of the chains, it follows that C_a starts its job of the interval I at time $\ell(I) + c_0 - s_i + s_i = \ell(I) + c_0$. The job of C_b in the interval starts at time $\ell(I) + c_0$ as well. This yields a contradiction. We conclude that W is an independent set. ◁

This shows that the Chain Scheduling with Exact Delays problem with a single machine, parametrized by the number of chains, is $W[1]$-hard. ◀

▶ **Lemma 4.5** (See [4, Lemma 4.5]). *The* Chain Scheduling with Exact Delays *problem with one machine, parameterized by the number of chains c is in $W[1]$.*

4.2 Constant number of parallel machines

If we assume that there are m machines, but m is considered to be a constant (i.e., not a fixed parameter; m is part of the problem description), then the problem is $W[1]$-complete.

▶ **Theorem 4.6.** *The* Chain Scheduling with Exact Delays *problem with m machines, parameterized by the number of chains, is $W[1]$-complete.*

Hardness follows easily from the case that $m = 1$: add $m - 1$ chains with maximum length that consist of jobs with 0 delay. Membership follows by formulating the problem as a Weighted CNF-SAT problem, where we have a variable for each chain C and each time t that it can start and clauses that check that we never use more than m machines. For details see the full version [4].

4.3 Variable number of parallel machines

The main result of this section is the following theorem.

▶ **Theorem 4.7.** *The* Chain Scheduling with Exact Delays *problems with a variable number of machines, parameterized by the number of chains c, is $W[2]$-complete.*

The proof can be found in the full version [4]. Hardness is shown by a reduction from Dominating Set; membership by a reduction to Threshold Dominating Set.

4.4 Minimum delays

With a simple modification, the hardness proofs for exact delays in unary can be modified to hardness proofs for minimum delays (still in unary). The modification consists of taking a number of copies of the instance, as described below.

Suppose we have an instance for Chain Scheduling with Exact Delays with c chains and m machines. We build an instance for Chain Scheduling with Minimum Delays. The intuition behind the construction is the following: we build $cT + 1$ identical copies of the original instance after each other. Here, T is the maximum deadline of all chains in the Chain Scheduling with Exact Delays instance. The delay between two copies of a chain C is $T - \ell_C$, where ℓ_C is the minimum duration of C. So there will be at least T time units between the execution of a job in copy i and the same job in copy $i + 1$. Each copy is executed in its own slot of T consecutive time steps, in the same way as a solution of the original instance. The release date of the new chain will stay r_C, while we set the deadline as $cT^2 + d_C$.

Suppose we have a solution of the new instance. Every new chain has a minimum duration of $cT^2 + \ell_C$. Thus, the total slack in a chain, *i.e.*, the sum of the differences between the scheduled delay time and the stated minimum delay, cannot be larger than $cT^2 + d_C - r_C - cT^2 - l_C = d_C - r_C - l_C$. Notice that this is at most T. Thus, for one chain, we have at most T copies where there is a pair of successive jobs with delays not equal to the minimum delay. As we have c chains, and $cT + 1$ copies, there must be at least one copy where each pair of successive jobs of all chains has delay equal to the minimum delay. This copy gives a solution of the original instance.

See Figure 6 for an illustration, and [4, Section 4.4] for more details.

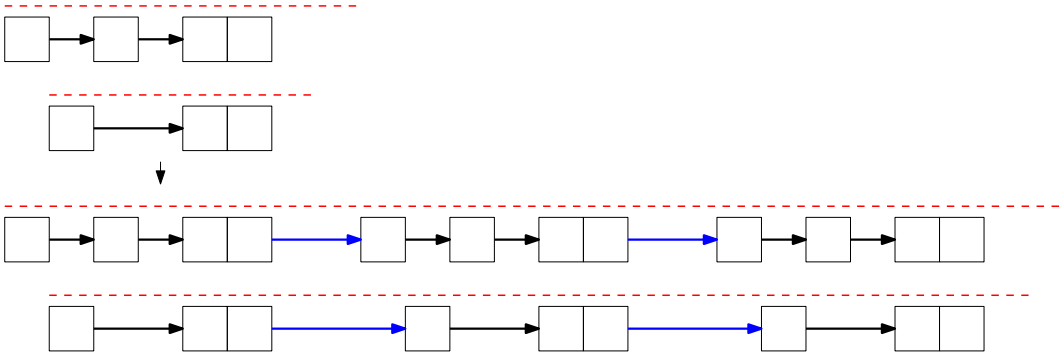


Figure 6 Construction in Section 4.4: copy every instance, the blue arrows are the delays between two copies of a chain, the red dashed line represent the interval from release date until deadline. The example shows the construction with three copies.

5 XP algorithms

In this section, we give two positive results. First, we show membership in XP for all studied variants, when delays are given in unary, with a relatively straightforward dynamic programming algorithm (Theorem 5.2; the proof can be found in the full version [4]). Then, in the case of one machine, we show that Scheduling with Exact Delays is in XP, when parameterized by the number of chains, even when delays are given in binary. For a related, earlier result, see [6].

▶ **Lemma 5.1.** *Given an instance of* Chain Scheduling with Exact Delays *or* Chain Scheduling with Minimum Delays*, one can build in polynomial time an equivalent instance, where all release dates and deadlines of chains are nonnegative integers, bounded by* $cn(D+1)$*, where* c *is the number of chains,* n *the number of jobs, and* D *the maximum delay between two successive jobs in a chain. In addition, for each chain* C*, we have* $d_C - r_C \leq n(D+1)$*.*

▶ **Theorem 5.2.** Chain Scheduling with Exact Delays *and* Chain Scheduling with Minimum Delays *belong to XP, when delays are given in unary, and parameterized by the number of chains or thickness, for any number of machines.*

The algorithm uses dynamic programming: we compute for each time step a table of states of partial solutions; the state tells for each chain what is the last job of the chain that is executed and at what time step this job was executed. For details, see the full version [4].

▶ **Theorem 5.3.** Chain Scheduling with Exact Delays *with* $m = 1$*, parameterized by the number of chains, with delays in binary belongs to XP.*

Proof. Suppose we have chains $C_1, \ldots, C_c$. Suppose chain C_i has jobs $j_{i,1}$, $j_{i,2}$, …, j_{i,ℓ_i}, with $j_{i,a}$ directly preceding $j_{i,a+1}$; we write the exact delay of this constraint as $l_{i,a}$. Write $s(i,a) = \sum_{b=1}^{a-1}(l_{i,b}+1)$. Note that $j_{i,a}$ has to be scheduled exactly $s(i,a)$ time steps after $j_{i,1}$ starts.

For each chain C_i, we take a variable x_i that denotes the time that the first job of C_i is scheduled.

▷ Claim 5.4. Variables $x_1, x_2, \ldots, x_c$ give a valid schedule, if and only if the following constraints are fulfilled.

1. For each i, $x_i \geq r_{C_i}$.
2. For each i, $x_i + s(i, \ell_i) < d_{C_i}$.
3. For each i and i' with $i \neq i'$, and each j, j' with $1 \leq j \leq \ell_i$, $1 \leq j' \leq \ell_{i'}$, we have $x_i + s(i,j) \neq x_{i'} + s(i',j')$.

The conditions state that chains do not start before the release date (1), do not finish after the deadline (2), and that no pair of jobs are scheduled at the same time (3). More details in [4].

The first step of the algorithm is to compute for each pair i, i' with $i \neq i'$ a set $U(i,i') = \{s(i',j') - s(i,j) \mid 1 \leq j \leq \ell_i,\ 1 \leq j' \leq \ell_{i'}\}$. Note that each of these sets has size $O(n^2)$, or more precisely, is at most the product of the sizes of the two chains. Now, sort each set $U(i,i')$.

Suppose $U(i,i') = \{a_1, a_2, \ldots, a_r\}$ with $a_1 < a_2 < \cdots < a_r$. Condition 3 of Claim 5.4 for the pair i, i' can be expressed as

$$(x_i - x_{i'} < a_1) \vee (a_1 < x_i - x_{i'} < a_2) \vee (a_2 < x_i - x_{i'} < a_3) \vee \cdots \\ \cdots \vee (a_{r-1} < x_i - x_{i'} < a_r) \vee (a_r < x_i - x_{i'})$$

Our algorithm now branches on these $O(n^2)$ possibilities. For each of the $O(c^2)$ pairs of chains, we have $O(n^2)$ branches, which gives a total of $O(n^{O(c^2)})$ subproblems.

Each of these subproblems asks to solve a set of inequalities. These inequalities are of the form $x_i - x_{i'} < a$ or $x_i \geq a$ (Condition 1 of Claim 5.4) or $x_i \leq a$ (Condition 2 of Claim 5.4), for some integers a. As we work with integers and look for integer solutions, we reformulate constraints of the form $x_i - x_{i'} < a$ as $x_i - x_{i'} \leq a - 1$. We now have a system of linear inequalities which can be solved in polynomial time with text book (shortest paths) methods, see e.g., [7, Section 24.4]. If at least one of the subproblems has a solution, then this solution gives starting times for the chains that gives a valid schedule; otherwise, there is no valid schedule.

We have $O(n^{O(c^2)})$ branches, each taking polynomial time, and this gives a running time of $O(n^{O(c^2)})$. ◀

6 Conclusions

In this paper, we have shown a number of results on the parameterized complexity of Chain Scheduling with Exact Delays and Chain Scheduling with Minimum Delays. In a few cases, we obtained $W[1]$-completeness or $W[2]$-completeness; in the other cases, we only showed hardness results, often together with XP-membership. We expect that the problems, parameterized by the thickness do not belong to $W[P]$ – for the same "compositionality" reason as why one can believe that Graph Bandwidth does not belong to $W[P]$: see the discussion in [12, Section 4]. The machinery to prove such results currently is not available, but we conjecture that also the variants with minimum delays inhibit some form of compositionality and do not belong to $W[P]$.

We end this paper with mentioning some open problems. In this paper, we proved for the case that delays are given in binary, for only one of the cases membership in XP. What is the complexity of the other cases when delays are given in binary? Also, an interesting question is to study the variant where we have maximum delays, with all of its subcases.

Another open problem is whether the results where we prove $W[t]$-hardness for all t can be improved to $W[SAT]$-hardness. Our current constructions give instances that grow exponentially with the nesting depth of conjunctions and disjunctions. We currently do not

know how to avoid this exponential blowup, so it appears that possible $W[SAT]$-hardness proofs for Chain Scheduling with Exact Delays parameterized by thickness need different techniques.

References

1. Martin Aigner and Günter M. Ziegler. Bertrand's postulate. In *Proofs from THE BOOK*, pages 7–12. Springer, 2001. doi:10.1007/978-3-662-04315-8_2.
2. S. Bessy and R. Giroudeau. Parameterized complexity of a coupled-task scheduling problem. *Journal of Scheduling*, 22(3):305–313, 2019. doi:10.1007/s10951-018-0581-1.
3. Hans L. Bodlaender and Michael R. Fellows. $W[2]$-hardness of precedence constrained K-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995. doi:10.1016/0167-6377(95)00031-9.
4. Hans L. Bodlaender and Marieke van der Wegen. Parameterized complexity of scheduling chains of jobs with delays, 2020. arXiv preprint. arXiv:2007.09023.
5. Peter Brucker, Johann Hurink, and Wieslaw Kubiak. Scheduling identical jobs with chain precedence constraints on two uniform machines. *Mathematical Methods of Operations Research*, 49:211—219, 1999. doi:10.1007/PL00020913.
6. Mark Cieliebak, Thomas Erlebach, Fabian Hennecke, Birgitta Weber, and Peter Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In *Exploring New Frontiers of Theoretical Informatics. IFIP International Federation for Information Processing*, volume 155, pages 209–222, 2004. doi:10.1007/1-4020-8141-3_18.
7. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms-third-edition.
8. Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24:873–921, 1995. doi:10.1137/S0097539792228228.
9. Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for $W[1]$. *Theoretical Computer Science*, 141(1-2):109–131, 1995. doi:10.1016/0304-3975(94)00097-3.
10. P. Erdös and P. Turán. On a problem of Sidon in additive number theory, and on some related problems. *Journal of the London Mathematical Society*, s1-16(4):212–215, 1941. doi:10.1112/jlms/s1-16.4.212.
11. Michael R. Fellows and Catherine McCartin. On the parametric complexity of schedules to minimize tardy tasks. *Theoretical Computer Science*, 298(2):317–324, 2003. doi:10.1016/S0304-3975(02)00811-3.
12. Michael R. Fellows and Frances A. Rosamond. Collaborating with Hans: Some remaining wonderments. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms — Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 7–17. Springer, 2020. doi:10.1007/978-3-030-42071-0_2.
13. Matthias Mnich and René van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Comput. Oper. Res.*, 100:254–261, 2018. doi:10.1016/j.cor.2018.07.020.
14. Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1):533–562, 2015. doi:10.1007/s10107-014-0830-9.
15. René van Bevern, Christian Komusiewicz, and Manuel Sorge. A parameterized approximation algorithm for the mixed and windy capacitated arc routing problem: Theory and experiments. *Networks*, 70(3):262–278, 2017. doi:10.1002/net.21742.
16. Erick D. Wikum, Donna C. Llewellyn, and George L. Nemhauser. One-machine generalized precedence constrained scheduling problems. *Operations Research Letters*, 16(2):87–99, 1994. doi:10.1016/0167-6377(94)90064-7.
17. Gerhard J. Woeginger. A comment on scheduling on uniform machines under chain-type precedence constraints. *Operations Research Letters*, 26(3):107–109, 2000. doi:10.1016/S0167-6377(99)00076-0.

Vertex Deletion into Bipartite Permutation Graphs

Łukasz Bożyk
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland
l.bozyk@uw.edu.pl

Jan Derbisz
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland
jan.derbisz@doctoral.uj.edu.pl

Tomasz Krawczyk
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland
krawczyk@tcs.uj.edu.pl

Jana Novotná
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland
Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic
janca@kam.mff.cuni.cz

Karolina Okrasa
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland
k.okrasa@mini.pw.edu.pl

Abstract

A permutation graph can be defined as an intersection graph of segments whose endpoints lie on two parallel lines ℓ_1 and ℓ_2, one on each. A bipartite permutation graph is a permutation graph which is bipartite. In this paper we study the parameterized complexity of the bipartite permutation vertex deletion problem, which asks, for a given n-vertex graph, whether we can remove at most k vertices to obtain a bipartite permutation graph. This problem is NP-complete by the classical result of Lewis and Yannakakis [17].

We analyze the structure of the so-called almost bipartite permutation graphs which may contain holes (large induced cycles) in contrast to bipartite permutation graphs. We exploit the structural properties of the shortest hole in a such graph. We use it to obtain an algorithm for the bipartite permutation vertex deletion problem with running time $f(k)n^{O(1)}$, and also give a polynomial-time 9-approximation algorithm.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases permutation graphs, comparability graphs, partially ordered set, graph modification problems

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.5

Related Version A full version of this paper is available at `https://arxiv.org/abs/2010.11440`.

Funding *Łukasz Bożyk, Jana Novotná, Karolina Okrasa:* This research is supported by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement 714704. It was partially carried out during the Parameterized Algorithms Retreat of the University of Warsaw, PARUW 2020, held in Krynica-Zdrój in February 2020.
Tomasz Krawczyk: This research is supported by National Science Center of Poland, grant UMO-2015/17/B/ST6/01873.
Jana Novotná: Supported by the student grant SVV–2020–260578.

Acknowledgements The authors would like to thank Bartosz Walczak for valuable comments and help with merging two groups of researchers working on similar projects into one. They also thank to anonymous reviewers for helpful comments.

1 Introduction

Many standard computational problems, including maximum clique, maximum independent set, or minimum coloring, which are NP-hard in general, have polynomial-time exact or approximation algorithms in restricted classes of graphs. Due to the practical and theoretical applications, some of such graph classes are particularly intensively studied. Among them are:

- *interval graphs*: intersection graphs of intervals on a real line,
- *unit interval graphs*: intersection graphs of intervals none of which is contained in another,
- *chordal graphs*: intersection graphs of subtrees of a tree,
- *function and permutation graphs*: intersection graphs of continuous and linear functions, respectively, defined on the interval $[0, 1]$,
- *comparability graphs*: graphs whose edges correspond to the pairs of vertices comparable in some fixed partial order $<$ on the vertex set (such an order is called a *transitive orientation* of the graph),
- *co-comparability graphs*: the complements of comparability graphs.

It is well known that the class of function graphs corresponds to the class of co-comparability graphs [11], and the class of permutation graphs corresponds to the intersection of comparability and co-comparability graphs [21] (see Figure 1 for the hierarchy of inclusions). All these classes of graphs are *hereditary*, which means that they are closed under vertex deletion.

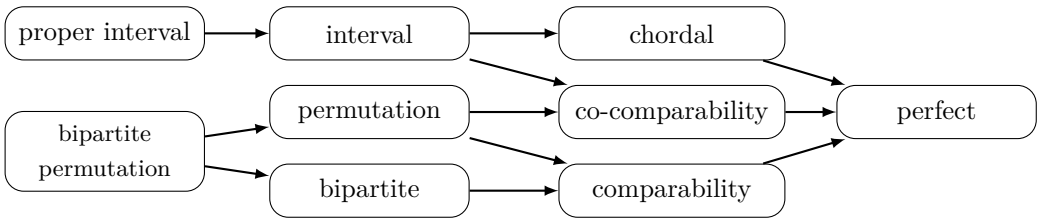


Figure 1 An hierarchy of inclusion of the hereditary graph classes considered in the introduction. An arrow from graph class A to graph class B indicates that $A \subset B$.

Being hereditary is a very useful property in algorithmic design as every hereditary class of graphs can also be uniquely characterized in terms of minimal forbidden induced subgraphs: a graph belongs to a class $\mathcal{G}$ if and only if it does not contain any graph from some family $\mathcal{F}$ as an induced subgraph. For every graph class introduced above, a characterization by forbidden subgraphs is known, see [7] for perfect graphs, [16] for interval graphs, [10] for comparability and permutation graphs. However, for all of them, the family of forbidden subgraphs is infinite and it may also be quite complex. Moreover, every graph G from any class introduced above is *perfect*. Grötschel, Lovász, and Schrijver [12] showed that in the class of perfect graphs the maximum clique, the maximum independent set, and the minimum coloring problems can be solved in polynomial time.

Polynomial-time algorithms devised for the above-mentioned graph classes can sometimes be adjusted to also work on graphs that are "close" to graphs from these classes. Usually, the "closeness" of a graph G to a graph class $\mathcal{G}$ is measured by the number of operations required to transform G into a graph from the class $\mathcal{G}$, where a single operation consists either on removing a vertex from G or on adding or removing an edge from G. Such an approach leads us to the following generic problem.

Problem: Graph modification problem into a class of graphs $\mathcal{G}$
Input: A graph G (typically not from $\mathcal{G}$) and a number k
Question: Can the graph G be transformed into a graph of the class $\mathcal{G}$ by performing at most k modifications of an appropriate kind?

Depending on the kind of modifications allowed, we obtain four variants of this problem: vertex deletion problem, edge deletion problem, edge completion problem, and edge edition problem (the latter allowing both deletions and additions of edges). For the class of graphs defined above, all four variants of the modification problem are NP-hard – see [19] for references to NP-hardness proofs. In particular, Lewis and Yannakakis [17] showed that the vertex deletion problem into any non-trivial hereditary class of graphs is NP-hard. This is not surprising, as many classical hard problems can be formulated as vertex deletion problems into particular classes of graphs, for example, Vertex Cover as vertex deletion to edgeless graphs, Feedback Vertex Set as vertex deletion to forests, and Odd Cycle Transversal as vertex deletion to bipartite graphs.

Graph modification problems are a popular research direction in the study of the *parameterized complexity* of NP-complete problems. In general, for a problem Π, an input of a parameterized problem consists of an instance I of Π and a *parameter* $k \in \mathbb{N}$. Then we say that Π is *fixed parameter tractable* (FPT) if there exists an algorithm deciding whether I is a yes-instance of Π in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where f is some computable function. For a graph modification problem, we often choose the parameter k as a number of allowed modifications, so the instance of such a problem is still a pair (G, k).

It turns out that characterizations by forbidden structures are sometimes useful to design FPT algorithms for graph modification problems. For example, Cai [3] proposed an FPT algorithm for modification problems into classes of graphs characterized by a finite family of forbidden induced subgraphs $\mathcal{F}$. His algorithm identifies a forbidden structure in the input graph (which can be done in polynomial time when $\mathcal{F}$ is finite) and branches over all possible ways of modifying that structure. Since the families of forbidden structures are infinite for graph classes introduced above, modification algorithms for these classes have to be much more sophisticated. For several of them modification problems have satisfactory solutions:

- chordal graphs: all four versions of the modification problem are FPT [6, 20];
- interval graphs: edge completion and edge deletion are FPT [25, 4], vertex deletion is FPT [6], edge edition remains open;
- proper interval graphs: all four versions of the modification problem are FPT [5].

On the other hand, it is known that the vertex deletion to perfect graphs is W[2]-hard [13]. It is worth mentioning that for a long time, it was unknown whether there are classes of graphs recognizable in polynomial time for which modification problems are hard. The first such example was given by Lokshtanov [18], who proved that the vertex deletion is W[2]-hard for graphs avoiding all *wheels* (i.e., cycles with an additional vertex adjacent to all other vertices). It is unknown whether comparability graphs, co-comparability graphs, and permutation graphs have FPT modification algorithms. The class of co-comparability graphs, which constitutes the superclass of interval graphs and an important subclass of perfect graphs, seems to be particularly interesting from the parameterized point of view.

Our focus. Like the class of interval graphs, the class of permutation graphs admits polynomial-time algorithms for rich family problems which are NP-complete in general. Apart from the already mentioned classical hard problems which are polynomial-time solvable for perfect graphs, there also exist polynomial algorithms solving e.g., Hamiltonian Cycle, Feedback Vertex Set or Dominating Set in the class of permutation graphs [2, 8].

In light of the above considerations, since all the modification problems into the class of permutation graphs – and the related classes of comparability and co-comparability graphs – remain open, restricting our attention to the class of *bipartite permutation graphs* appears to be a natural research direction. Bipartite permutation graphs form an interesting graph class themselves, first investigated by Spinrad, Brandstädt, and Stewart [22], who characterized them by means of appropriately chosen linear orderings of its bipartition classes.

One of the most interesting results concerning the bipartite permutation graphs is by Heggernes et al. [14], who showed that the NP-complete problem of computing the *cutwidth* of a graph (i.e., finding a linear order of the vertices of a graph that minimizes the maximum number of edges intersected by any line inserted between two consecutive vertices) is polynomial for bipartite permutation graphs.

Our algorithm exploits the absence of some forbidden structures in bipartite permutation graphs. Since these structures cannot, in particular, occur in permutation graphs, we believe that besides being a complete result itself, our research is a step towards understanding the parameterized complexity of modification problems into permutation graphs.

Our results. We focus mainly on the modification by vertex deletion.

▶ **Theorem 1.** *There is an $\mathcal{O}(9^k \cdot |V(G)|^9)$-time algorithm for instances (G, k) of the vertex deletion into bipartite permutation graphs problem.*

We prove Theorem 1 in Section 4. Our algorithm is based on the characterization of bipartite permutation graphs by forbidden subgraphs. Using the characterization, at first, we get rid of constant-size forbidden subgraphs by branching, which is a standard technique in modification problems on hereditary graph classes [24, 25]. We call graphs without these forbidden subgraphs *almost bipartite permutation graphs*.

Our main contribution is in the structural analysis of almost bipartite permutation graphs which may contain holes (on more than ten vertices) in contrast to bipartite permutation graphs. This approach is partially inspired by the ideas of van 't Hof and Villanger [24] who used similar tools in their work on proper interval vertex deletion problem. We use the result of Spinrad, Brandstädt, and Stewart [22], who showed that the vertices of every connected bipartite permutation graph $G = (U, W, E)$ can be embedded into a strip in such a way that the vertices from U are on the bottom edge of the strip, the vertices from W are on the top edge of the strip, the neighbors $N(u)$ of u occur consecutively on the top edge of the strip for every $u \in U$ (adjacency property), the vertices from $N(u) - N(u')$ occur consecutively on the top edge of the strip for every $u, u' \in U$ (enclosure property), and the analogous properties are satisfied by the vertices in W (see Figure 2).

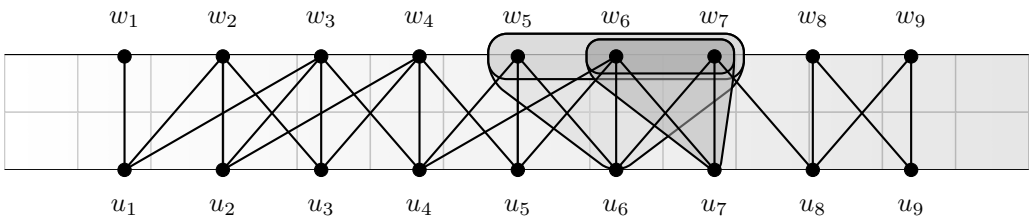


Figure 2 Embedding of a bipartite permutation graph (U, W, E) into a strip satysfying the adjacency and the enclosure properties.

Our structural result asserts that, depending on the parity of the length of the shortest hole, a connected almost bipartite permutation graph may be naturally embedded in either a cylinder, or a Möbius strip, locally satisfying adjacency and enclosure properties (see Fig. 3).

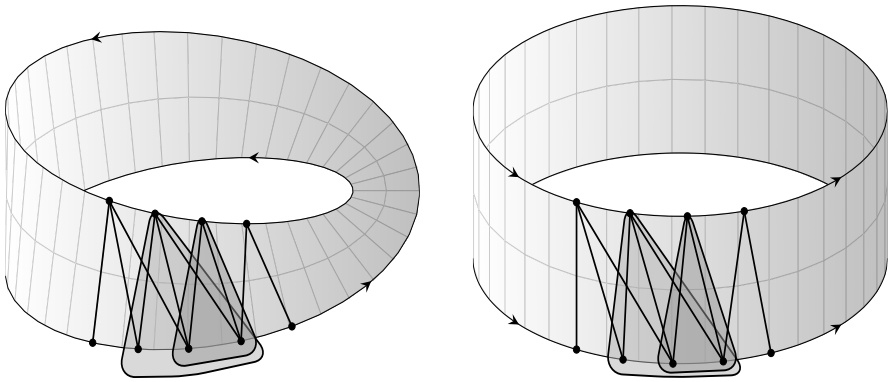


Figure 3 Embedding of an almost bipartite permutation graph in a cylinder or Möbius strip.

Once we obtain such structure, we show that every minimal vertex cut that destroys all holes lies nearby a few consecutive vertices from the shortest hole. This allows us to check all the possibilities where we can find a minimum cut. Finally, we use a polynomial algorithm for finding maximum flow (and thus a minimum cut).

The approach used to prove Theorem 1 can be slightly modified to obtain a 9-approximation algorithm for the bipartite permutation vertex deletion problem. We show the following.

▶ **Theorem 2.** *There exists a polynomial-time* 9*-approximation algorithm for vertex deletion into bipartite permutation graphs problem.*

2 Preliminaries

Unless stated otherwise, all graphs considered in this work are simple, i.e., undirected, with no loops and parallel edges. Let $G = (V, E)$ be a graph. For a subset $S \subseteq V$, the subgraph of G induced by S is the graph $G[S] = (S, \{uv \mid uv \in E, u, v \in S\})$. The *neighborhood* of a vertex $u \in V$ is the set $N(u) = \{v \in V \mid uv \in E\}$. Similarly, we write $N(U) = \bigcup_{u \in U} N(u) \setminus U$ for a set $U \subseteq V$. Let $u, v \in V$. We say that u and v *are at distance* k *(in* G*)* if k is the length of a shortest path between u and v in G. We denote a complete graph and a cycle on n vertices by K_n and C_n, respectively. By *hole* we mean an induced cycle on at least five vertices. We say that a hole is *even* (or *odd*) if it contains even (odd) number of vertices, respectively.

For a graph $G = (V, E)$, a pair $(V, <)$ is a *transitive orientation* of G if $<$ is a transitive and irreflexive relation on V that satisfies either $u < v$ or $v < u$ iff $uv \in E$ for every $u, v \in V$.

A *partially ordered set* (shortly *partial order* or *poset*) is a pair $P = (X, \leqslant_P)$ that consists of a set X and a reflexive, transitive, and antisymmetric relation $\leqslant_P$ on X. For a poset $(X, \leqslant_P)$, let the *strict partial order* $<_P$ be a binary relation defined on X such that $x <_P y$ if and only if $x \leqslant_P y$ and $x \neq y$. Equivalently, $(X, <_P)$ is a strict partial order if $<_P$ is irreflexive and transitive. Two elements $x, y \in X$ are *comparable* in P if $x \leqslant_P y$ or $y \leqslant_P x$; otherwise, x, y are *incomparable* in P. A *linear order* $L = (X, \leqslant_L)$ is a partial order in which for every $x, y \in X$ we have $x \leqslant_L y$ or $y \leqslant_L x$. A *strict linear order* $(X, <_L)$ is a binary relation defined in a way that $x <_L y$ if and only if $x \leqslant_L y$ and $x \neq y$.

Let $P = (X, \leqslant_P)$ be a poset. A linear order $L = (X, \leqslant_L)$ is called a *linear extension* of P if $\leqslant_P \subseteq \leqslant_L$. Given a family of posets $\mathcal{P} = \{P_i = (X, \leqslant_{P_i}) : i \in I\}$, we say that P is the *intersection of* $\mathcal{P}$ if for every $x, y \in X$ we have $x \leqslant_P y$ if and only if $x \leqslant_{P_i} y$ for every $i \in I$. The *dimension* of a poset P is the minimal number of linear extensions of P that intersect to P. We say that P is *two-dimensional* if it is is the intersection of two linear extensions of P.

A *comparability graph* (*incomparability graph*) of a poset $P = (X, \leqslant_P)$ has X as the set of its vertices and the set including every two vertices comparable (incomparable, respectively) in P as the set of its edges. Note the following: if $(X, \leqslant_P)$ is a poset, then $(X, <_P)$ is a transitive orientation of the comparability graph of P. A graph $G = (V, E)$ is a *comparability graph* (*co-comparability graph*) if G is a comparability (incomparability, respectively) graph of some poset defined on V. So, G is a comparability graph if and only if G admits a transitive orientation. A graph G is a *permutation graph* if and only if G and the complement of G are comparability graphs [21] (or equivalently, G and the complement of G admit transitive orientations). Baker, Fishburn, and Roberts [1] proved that G is a permutation graph if and only if G is the incomparability graph of a two-dimensional poset.

We say that two sets X and Y are *comparable* if X and Y are comparable with respect to $\subseteq$-relation (that is, $X \subseteq Y$ or $Y \subseteq X$ holds). We use the convenient notation $[m] := \{0, 1, \ldots, m\}$, for every $m \in \mathbb{N}$. For every $i, j \in \mathbb{Z}$ such that $i \leqslant j$ by $[i, j]$ we mean the set $\{i, i+1, \ldots, j\}$.

3 The structure of (almost) bipartite permutation graphs

The characterization of bipartite permutation graphs presented below was proposed by Spinrad, Brandstädt, and Stewart [22].

Suppose $G = (U, W, E)$ is a connected bipartite graph. A linear order $(W, <_W)$ satisfies *adjacency property* if for each vertex $u \in U$ the set $N(u)$ consists of vertices that are consecutive in $(W, <_W)$. A linear order $(W, <_W)$ satisfies *enclosure property* if for every pair of vertices $u, u' \in U$ such that $N(u)$ is a subset of $N(u')$, vertices in $N(u') - N(u)$ occur consecutively in $(W, <_W)$. A *strong ordering* of the vertices of $U \cup W$ consists of linear orders $(U, <_U)$ and $(W, <_W)$ such that for every $(u, w'), (u', w)$ in E, where u, u' are in U and w, w' are in W, $u <_U u'$ and $w <_W w'$ imply $(u, w) \in E$ and $(u', w') \in E$. Note that, whenever $(U, <_U)$ and $(W, <_W)$ form a strong ordering of $U \cup W$, then $(U, <_U)$ and $(W, <_W)$ satisfy the adjacency and enclosure properties.

▶ **Theorem 3** (Spinrad, Brandstädt, Stewart [22]). *The following three statements are equivalent for a connected bipartite graph $G = (U, W, E)$:*

1. *(U, W, E) is a bipartite permutation graph.*
2. *There exists a strong ordering of $U \cup W$.*
3. *There exists a linear order $(W, <_W)$ of W satisfying adjacency and enclosure properties.*

An example of a bipartite permutation graph $G = (U, W, E)$ with linear order $w_1 <_W w_2 <_W \ldots <_W w_8 <_W w_9$ of the vertices of W which satisfies the adjacency and the enclosure properties is shown in Figure 2.

Another characterization of bipartite permutation graphs can be obtained by listing all minimal forbidden induced subgraphs for this class of graphs. Such a list can be compiled by taking all odd cycles of length $\geqslant 3$ (forbidden structures for bipartite graphs) and all bipartite graphs from the list of forbidden structures for permutation graphs obtained by Gallai [10]. The whole list is shown in Figure 4.

3.1 Almost bipartite permutation graphs

The goal of this section is to characterize graphs which do not contain small forbidden subgraphs for the class of bipartite permutation graphs. Following terminology of van 't Hof and Villanger [24] we call such graphs almost bipartite permutation graphs.

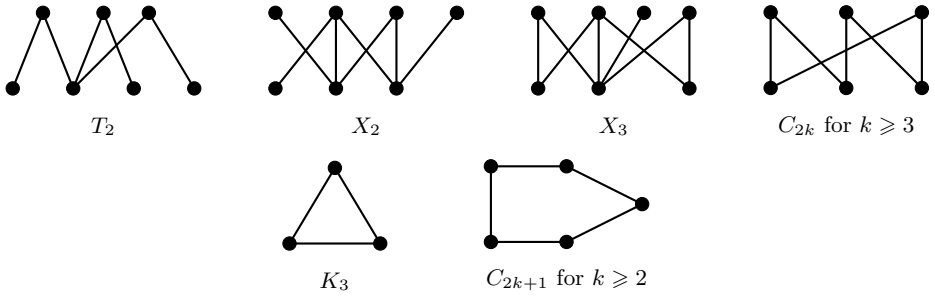


Figure 4 Forbidden structures for bipartite permutation graphs.

▶ **Definition 4.** *A graph $G = (V, E)$ is* almost bipartite permutation graph *if G does not contain T_2, X_2, X_3, K_3, C_k for $k \in [5, 9]$ as induced subgraphs.*

Suppose $G = (V, E)$ is a connected almost bipartite permutation graph.

▶ **Proposition 5.** *Every hole in G is a dominating set.*

Proof. Let $C = \{c_0, c_1, \ldots, c_\ell\}$ be a hole in G. Hence, $\ell \geqslant 10$. Suppose, for contradiction, that there exists a vertex in the set $V \setminus (C \cup N(C))$. As G is connected, there must exist $v \in V$ in distance two from C. Let $w \in N(v) \cap N(C)$ and let c_j be a neighbor of w in C. We now look at the neighborhood of w. As G contains no triangle, wc_{j-1} and wc_{j+1} are non-edges. Moreover, as G contains no copy of T_2, vertex w is adjacent to at least one of c_{j-2} and c_{j+2}, say c_{j-2}. Thus, w is nonadjacent to c_{j-3}. Therefore, the set $\{c_{j-3}, c_{j-2}, c_{j-1}, c_j, c_{j+1}, w, v\}$ induces a copy of X_2, which leads to a contradiction. ◀

Let C be a shortest hole in G, m be the size of C, and $c_0, c_1, \ldots, c_{m-1}$ be the consecutive vertices of C, $m \geqslant 10$. In the remaining part of the paper we use the following notation with respect to C. For any integral number i by c_i we denote the unique vertex $c_{i \bmod m}$ from the cycle C. For any two different vertices c_i, c_j in C, by *the set of all vertices between c_i and c_j from C* we mean the set $\{c_i, c_{i+1}, \ldots, c_{i+k}\}$, where k is the smallest natural number such that $c_{i+k} = c_j$. Note that this notion is not symmetric, i.e., the set of all vertices between c_j and c_i from C contains c_i, c_j and all the vertices from C that are not between c_i and c_j.

▶ **Proposition 6.** *For every vertex $v \in V$ either:*
(1) *$N(v) \cap C = \{c_i\}$ for some $i \in [m-1]$, or*
(2) *$N(v) \cap C = \{c_i, c_{i+2}\}$ for some $i \in [m-1]$.*

Proof. Since C is a cycle, (2) clearly holds for the vertices from C, so let v be a vertex in $V \setminus C$. As C is a dominating set, by Proposition 5, vertex v has at least one neighbor in C. If v has exactly one neighbor in C, then (1) holds and we are done. So assume that it has more than one neighbor. We now distinguish two cases. First, suppose that there exist two vertices $c_j, c_\ell \in N(v) \cap C$ at distance at least three in C such that v has no neighbor in the set of vertices between c_j and c_l, except c_j and c_l. Then, $\{c_j, c_{j+1}, \ldots, c_\ell, v\}$ induces a cycle C' on at least five vertices in G. As c_j and c_ℓ are at distance at least three in C, C' is shorter than C. In particular, C' contradicts either G containing no copy of C_ℓ, for $\ell \in \{5, \ldots, 9\}$, or C being a shortest hole in G. Therefore, this case never occurs.

Hence, v has either (i) exactly two neighbors in C and those are at distance two as there is no triangle in G, so (2) holds, or (ii) C has an even number of vertices and v is adjacent to every second vertex of C. It remains to show that the latter never occurs. Indeed, if it does, then without loss of generality $c_0 \in N(v)$. But observe that since C has at least ten vertices, the set $\{c_0, c_1, c_2, c_3, c_4, c_6, v\}$ induces a copy of X_3. This concludes the proof. ◀

Given Prop. 6, for every $i \in [m-1]$ we can set $A_i = \{v \in V : N(v) \cap C = \{c_{i-1}, c_{i+1}\}\}$ and $B_i = \{v \in V : N(v) \cap C = \{c_i\}\}$. Note that $A_0, B_0, \ldots, A_{m-1}, B_{m-1}$ is a partition of V and $c_i \in A_i$. Following our notation, for any integer i by A_i and B_i we denote the sets $A_{i \bmod m}$ and $B_{i \bmod m}$, respectively. Furthermore, for every $i \leqslant j$ we set:

$$A_G[i,j] = \begin{cases} A_i \cup B_{i+1} \cup A_{i+2} \cup B_{i+3} \cup \ldots \cup A_{j-1} \cup B_j & \text{if } j-i \text{ is odd,} \\ A_i \cup B_{i+1} \cup A_{i+2} \cup B_{i+3} \cup \ldots \cup B_{j-1} \cup A_j & \text{if } j-i \text{ is even,} \end{cases}$$

$$B_G[i,j] = \begin{cases} B_i \cup A_{i+1} \cup B_{i+2} \cup A_{i+3} \cup \ldots \cup B_{j-1} \cup A_j & \text{if } j-i \text{ is odd,} \\ B_i \cup A_{i+1} \cup B_{i+2} \cup A_{i+3} \cup \ldots \cup A_{j-1} \cup B_j & \text{if } j-i \text{ is even,} \end{cases}$$

and $V_G[i,j] = A_G[i,j] \cup B_G[i,j]$.

We write just $A[i,j]$, $B[i,j]$, and $V[i,j]$, respectively, instead of $A_G[i,j]$, $B_G[i,j]$, and $V_G[i,j]$, when there is no confusion.

We now characterize the neighborhoods of the vertices in sets A_i and B_i, see also Figure 5.

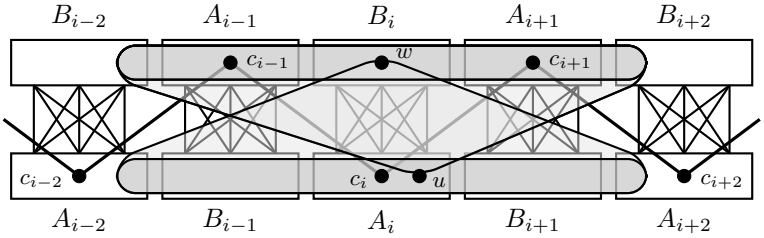


Figure 5 A possible neighborhood of u in A_i and w in B_i.

▶ **Proposition 7.** *Let $i \in [m-1]$. Then:*

(1) *A_i and B_i are independent sets.*

(2) *For every $u \in A_i$ and every $w \in B_i$ we have $uw \in E$.*

(3) *For every $u \in A_i$ we have $B_i \subseteq N(u) \subseteq B[i-2, i+2]$.*

(4) *For every $w \in B_i$ we have $A_i \subseteq N(w) \subseteq A[i-2, i+2]$.*

The proof of Proposition 7 can be found in the full version or it can also be easily seen by examining the forbidden subgraphs and the fact that C is a shortest hole containing at least 10 vertices. Proposition 7 asserts that all the neighbors of the vertices from A_i and from B_i are contained in the set $B[i-2, i+2]$ and $A[i-2, i+2]$, respectively. The next proposition describes the relations that hold between the neighborhoods of the vertices from $B[i-2, i+2]$ restricted to the set A_i and between the neighborhoods of the vertices from $A[i-2, i+2]$ restricted to the set B_i.

▶ **Proposition 8.** *Let $i \in [m-1]$. For $(i \pm 2, i \pm 1) \in \{(i-2, i-1), (i+2, i+1)\}$, the following hold:*

(1) *For every $w, w' \in B_{i\pm2} \cup A_{i\pm1}$ the sets $N(w) \cap A_i$ and $N(w') \cap A_i$ are comparable. Moreover, if $w \in B_{i\pm2}$ and $w' \in A_{i\pm1}$, then $N(w) \cap A_i \subseteq N(w') \cap A_i$.*

(2) *For every $u, u' \in A_{i\pm2} \cup B_{i\pm1}$ the sets $N(u) \cap B_i$ and $N(u') \cap B_i$ are comparable. Moreover, if $u \in A_{i\pm2}$ and $u' \in B_{i\pm1}$, then $N(u) \cap B_i \subseteq N(u') \cap B_i$.*

Proof. To prove (1), we consider the case $(i \pm 2, i \pm 1) = (i-2, i-1)$, as the other one follows by symmetry. Suppose that $w, w' \in B_{i-2} \cup A_{i-1}$ are such that neither $N(w) \cap A_i \subseteq N(w') \cap A_i$ nor $N(w') \cap A_i \subseteq N(w) \cap A_i$ holds. It means that there are $u, u' \in A_i$ such that $wu \in E$, $w'u' \in E$, $wu' \notin E$, and $w'u \notin E$. Since $w, w' \in B_{i-2} \cup A_{i-1}$, we

have $c_{i-2}w, c_{i-2}w' \in E$ and $c_{i-4}w, c_{i-3}w, c_{i-4}w', c_{i-3}w' \notin E$. Furthermore, $ww' \notin E$ and $uu' \notin E$ as G contains no triangle. Consequently, the set $\{c_{i-3}, w, w', c_{i-4}, c_{i-2}, u, u'\}$ induces a copy of T_2 in G, which cannot be the case. Moreover, if $w \in B_{i-2}$, $w' \in A_{i-1}$, then since $c_i \in (N(w') \cap A_i) \setminus (N(w) \cap A_i)$, the latter statement holds.

To show (2), we again only consider the case $(i \pm 2, i \pm 1) = (i-2, i-1)$. Suppose that $u, u' \in A_{i-2} \cup B_{i-1}$ are such that neither $N(u') \cap B_i \subseteq N(u) \cap B_i$ nor $N(u) \cap B_i \subseteq N(u') \cap B_i$ holds. It means that there are $w, w' \in B_i$ such that $uw, u'w' \in E$ and $u'w, uw' \notin E$. Since $u, u' \in A_{i-2} \cup B_{i-1}$, we have $uc_{i-1}, u'c_{i-1} \in E$ and $uc_{i+1}, u'c_{i+1} \notin E$. Furthermore, $uu' \notin E$ and $ww' \notin E$ as G contains no triangle. Hence, the set $\{c_{i-1}, w, w', c_{i+1}, u, c_i, u'\}$ induces a copy of X_3 in G, which cannot be the case.

To see the second part of the statement, assume that $N(u) \cap B_i \not\subseteq N(u') \cap B_i$ for some $u \in A_{i-2}$, $u' \in B_{i-1}$. That is, there is $w \in B_i$ such that $uw \in E$ and $u'w \notin E$. In particular, it means that $u \neq c_{i-2}$. Note that $uc_{i-1}, u'c_{i-1} \in E$. Consequently, the set $\{c_{i-3}, c_{i-2}, c_{i-1}, c_i, u, u', w\}$ induces a copy of X_3 in G, which is a contradiction. ◀

Proposition 8 allows us to order vertices of A_i based on two properties. We now define relation $<_{A_i}$ which combines them and we show that $<_{A_i}$ is a partial order (see Figure 6 for an illustration). We define for every $u, u' \in A_i$:

$$u <_{A_i} u' \text{ iff } \begin{array}{l} \text{there is } w \in B_{i-2} \cup A_{i-1} \text{ such that } u \in N(w) \text{ and } u' \notin N(w), \text{ or} \\ \text{there is } w \in A_{i+1} \cup B_{i+2} \text{ such that } u' \in N(w) \text{ and } u \notin N(w), \end{array}$$

Similarly, we define a relation $<_{B_i}$ for every $w, w' \in B_i$:

$$w <_{B_i} w' \text{ iff } \begin{array}{l} \text{there is } u \in A_{i-2} \cup B_{i-1} \text{ such that } w \in N(u) \text{ and } w' \notin N(u), \text{ or} \\ \text{there is } u \in B_{i+1} \cup A_{i+2} \text{ such that } w' \in N(u) \text{ and } w \notin N(u). \end{array}$$

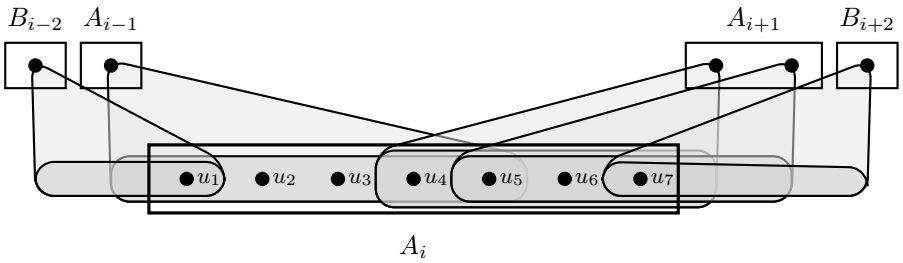


Figure 6 The neighborhoods of the vertices from $B_{i-2} \cup A_{i-1} \cup A_{i+1} \cup B_{i+2}$ restricted to A_i. We have $u_1 <_{A_i} \{u_2, u_3\} <_{A_i} u_4 <_{A_i} u_5 <_{A_i} u_6 <_{A_i} u_7$.

▶ **Proposition 9.** *The following statements hold for every $i \in [m-1]$:*

1. *$(A_i, <_{A_i})$ is a strict partial order. Moreover, $u, u' \in A_i$ are incomparable in $(A_i, <_{A_i})$ if and only if $N(u) = N(u')$.*
2. *$(B_i, <_{B_i})$ is a strict partial order. Moreover, $w, w' \in B_i$ are incomparable in $(B_i, <_{B_i})$ if and only if $N(w) = N(w')$.*

A formal proof of Proposition 9 can be found in the full version. Here, we only observe that we obtain two orders of vertices of A_i (resp. B_i) from Proposition 8 and they are reflected in the definition of $<_{A_i}$ (resp. $<_{B_i}$). However, the fact that our graph does not contain small forbidden induced subgraphs, combined with the previous proposition, implies that we cannot have $u <_{A_i} u'$ and $u' <_{A_i} u$ simultaneously, for any $u, u' \in A_i$ (an analogous

situation holds for $<_{B_i}$). For instance, if it did not hold for some $u, u' \in A_i$ because of the existence of some $w \in B_{i-2} \cup A_{i-1}$ such that $u \in N(w)$ and $u' \notin N(w)$, and some $w' \in A_{i+1} \cup B_{i+2}$ such that $u \in N(w')$ and $u' \notin N(w')$, an induced copy of X_2 would be found on vertices $\{w, w', c_{i+1}, c_{i+2}, c_{i+3}, u, u'\}$. These observations are a key ingredient of proving the transitivity of our relation, the proof itself requires more technicalities.

Finally, for every $i \in [m-1]$ we order arbitrarily the elements inside every antichain of $(A_i, <_{A_i})$ and of $(B_i, <_{B_i})$, obtaining strict linear orders $(A_i, <_{A_i})$ and $(B_i, <_{B_i})$. We introduce a binary relation $\prec$ defined on the set V, such that $v \prec v'$ for $v, v' \in V$ if one of the following conditions holds for some $i \in [m-1]$:

- $v, v' \in A_i$, $v <_{A_i} v'$, and v, v' are consecutive in $(A_i, <_{A_i})$,
- $v, v' \in B_i$, $v <_{B_i} v'$, and v, v' are consecutive in $(B_i, <_{B_i})$,
- v is the maximum of $(A_i, <_{A_i})$ and v' is the minimum of $(B_{i+1}, <_{B_{i+1}})$,
- v is the maximum of $(B_i, <_{B_i})$ and v' is the minimum of $(A_{i+1}, <_{A_{i+1}})$.

Informally, to get an embedding of G into a cylinder (the shortest hole is even) or into a Möbius strip (the shortest hole is odd) which locally satisfies the adjacency and the enclosure properties, we place the vertices v, v' satisfying $v \prec v'$ next to each other, v before v' assuming that the border of the cylinder or the Möbius strip are oriented as shown in Figure 3. In what follows we extend $\prec$ relation as follows:

- For every $V' \subsetneq V$ by $<_{V'}$ we denote the transitive closure of $\prec$ restricted to V',
- For $v, v' \in V$ we set $v <_{cl} v'$ if $v, v' \in A[i-2, i+2]$ and $v <_{A[i-2,i+2]} v'$ for some $i \in [m-1]$ or $v, v' \in B[i-2, i+2]$ and $v <_{B[i-2,i+2]} v'$ for some $i \in [m-1]$.

Finally, the following lemma characterizes the global structure of an almost bipartite permutation graph. The proof is omitted and can be found in the full version.

▶ **Lemma 10.** *Let i, j be such that $i \leqslant j$, $|j - i| = m - 3$. Let $U = A[i, j]$ and $W = B[i, j]$. Then $G[U \cup W]$ is a bipartite permutation graph with bipartition classes U and W.*

Moreover, $(U, <_U)$ and $(W, <_W)$ are strict linear orders that satisfy the adjacency and enclosure properties in $G[U \cup W]$.

Lemma 10 provides an interesting view on classification of almost bipartite permutation graphs. Specifically, if m is even, then the graph may be drawn on a cylinder, whose boundary consists of two closed curves, one of which traverses vertices of $A[0, m-1]$, and the second one – the vertices of $B[0, m-1]$. If in turn m is odd, then the graph can be represented on a Möbius strip, whose boundary traverses consecutive vertices of $A[0, m-1]$ and then $B[0, m-1]$ (recall Figure 3).

The following definitions are taken from [24]. A *hole cut* of G is a vertex set $X \subseteq V$ such that $G - X$ is a bipartite permutation graph. Lemma 10 asserts that for every $i \in [m-1]$ the set $V[i, i+1]$ is a hole cut in G. A hole cut X of G is *minimum* if G does not have a hole cut whose size is strictly smaller than the size of X. A hole cut X of G is *minimal* if any proper subset of X is not a hole cut in G. Note, in particular, that for every $i \in [m-1]$ the set $V[i, i+1]$ is a hole cut.

The next proposition describes the structure of every hole in G.

▶ **Proposition 11.** *Suppose C' is a hole of size k in G for some $k \geqslant m$. Then, the consecutive vertices of C' can be labeled by $c'_0, c'_1, \ldots, c'_{k-1}$ so as the following conditions hold (the indices are taken modulo k):*

- *$c'_i c'_{i+1} \in E$ for every $i \in [k-1]$,*
- *$c'_i <_{cl} c'_{i+2}$ for every $i \in [k-1]$,*
- *$C' \cap \{c'' : c'_i <_{cl} c'' <_{cl} c'_{i+2}\} = \emptyset$ for every $i \in [k-1]$.*

Proof. Let $c'_0 = v_i, c'_1, c'_2, \ldots, c'_{n-1}$ be consecutive vertices of C' denoted in such a way that $c'_0 <_{cl} c'_2$. We note that we can suppose it as $c'_0, c'_2 \in N(c'_1)$, thus, by Proposition 7(3) and (4), both c'_0, c'_2 belong to $A[\ell-2, \ell+2]$ or both belong to $B[\ell-2, \ell+2]$ for some $\ell \in [m-1]$.

Now, we show that if there exists $j \in [m-1]$ such that $c'_j <_{cl} c'_{j+2}$, then $c'_{j+1} <_{cl} c'_{j+3}$. Suppose, for contradiction that $c'_j <_{cl} c'_{j+2}$ and $c'_{j+1} \not<_{cl} c'_{j+3}$. Let $i \in [m-1]$ be such that $c_{j+2} \in (A_i \cup B_i)$. Similarly, as $c'_{j+1}, c'_{j+3} \in N(c'_{j+2})$, either $c'_{j+1}, c'_{j+3} \in B[i-2, i+2]$ if $c'_{j+2} \in A_i$ or $c'_{j+1}, c'_{j+3} \in A[i-2, i+2]$ if $c'_{j+2} \in B_i$. In both cases Lemma 10 implies that $<_{cl}$ restricted to $V[i-4, i+2]$ is a strong ordering of $G[V[i-4, i+2]]$. Moreover, c'_{j+1}, c'_{j+3} are comparable in $<_{cl}$, by Proposition 7(3) and (4), thus, $c'_{j+3} <_{cl} c'_{j+1}$. Therefore Theorem 3(2) yields $c'_j c'_{j+3} \in E$, so a chord in C' – contradiction. Therefore $c'_j <_{cl} c'_{j+2}$ implies $c'_{j+1} <_{cl} c'_{j+3}$ for every integer j. Applying above observation repeatedly for $j = 0, 1, 2, \ldots$, we get that $c'_0 <_{cl} c'_2 <_{cl} c'_4 <_{cl} \ldots$ and $c'_1 <_{cl} c'_3 <_{cl} c'_5 <_{cl} \ldots$

Suppose for the sake of contradiction that there exists $j \notin \{i, i+2\}$ such that $c'_i <_{cl} c'_j <_{cl} c'_{i+2}$. Then by Lemma 10, $c'_j c'_{i+1} \in E$ due to the adjacency property. But the edge $c'_j c'_{i+1}$ is a chord in C'. This completes the proof. ◀

The structure of holes described above asserts that for every $i \in [m-1]$ the sets $A[i-2, i+2]$ and $B[i-2, i+2]$ are hole cuts. We use this observation to prove the following statement about minimal hole cuts in G.

▶ **Proposition 12.** *Every minimal hole cut X in G is fully contained in the set $V[i-2, i+2]$ for some $i \in [m-1]$.*

Proof. First, note that we can choose elements z_1, x_1, x_2, z_2 in V and an index $i \in [m-1]$ such that the following conditions hold:

- we have $z_1 \prec x_1 \leqslant_{cl} x_2 \prec z_2$, the set $X' = \{x : x_1 \leqslant_{cl} x \leqslant_{cl} x_2\}$ is non-empty and is contained in X, and the elements z_1, z_2 are not in X,
- the set $X' \cup \{z_1, z_2\}$ is contained in either $B[i-2, i+2]$ or in $A[i-2, i+2]$ and we have $z_1 \in B[i-2, i]$ and $z_2 \in B[i, i+2]$ if $X' \cup \{z_1, z_2\} \in B[i-2, i+2]$, and similarly for the other case.

Note that such a choice of z_1, x_1, x_2, z_2 and i is possible as the sets $A[j, j+3]$ and $B[j, j+3]$ are hole cuts for every $j \in [m-1]$, by combining Proposition 11 and Proposition 7(3),(4). For the rest of the proof we assume $X' \subset B[i-2, i+2]$, $z_1 \in B[i-2, i]$ and $z_2 \in B[i, i+2]$ (see Figure 7 for an illustration).

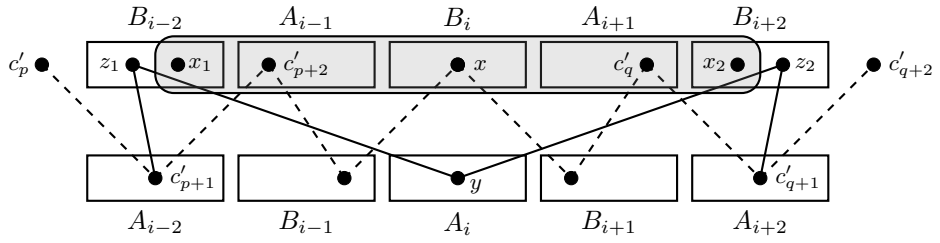


Figure 7 Illustration of the proof: the cycle C' is marked with a dashed line. The set X' is shaded.

Suppose Y' is the set consisting of all the neighbors of z_1 and z_2; that is, $Y' = N(z_1) \cap N(z_2)$. Clearly, we have $Y' \subset A[i-2, i+2]$. To complete the proof of the proposition we show that:

- every element of Y' is a member of X,
- $X' \cup Y'$ is a hole cut in G.

Then we have $X' \cup Y' = X$ by minimality of X and consequently $X \subset V[i-2, i+2]$. So, it remains to prove the claims about the set Y'.

Suppose we have $y \in Y'$ such that $y \notin X$. Since X is a minimal hole cut, $X \setminus \{x\}$ is not a hole cut, where x is some fixed element from X'. That is, there is a hole C' in $G - (X \setminus \{x\})$. Note that C' must contain x. Suppose $c'_0, \ldots, c'_{\ell-1}$ for some $\ell \geqslant 9$ are consecutive vertices in C' chosen such that $c'_j <_{cl} c_{j+2}$ for every $j \in [\ell - 1]$ (indices are taken modulo ℓ). Now we pick $p, q \in [\ell - 1]$ such that $c'_p <_{cl} z_1 \leqslant_{cl} c'_{p+2}$ and $c'_q \leqslant_{cl} z_2 <_{cl} c'_{q+2}$. Since $x \in C'$, we have $c'_{p+2} \leqslant_{cl} x$ and $c'_q \leqslant_{cl} x$. Note that c'_{p+1} is adjacent to z_1 and c'_{q+1} is adjacent to z_2. Next we replace in C' all the vertices between c'_{p+2} and c'_q (this set includes x) with the vertices z_1, y, z_2 and we obtain a cycle C'' containing no elements from X. Clearly, we can easily find a hole among the elements from C'' that avoids all the elements from X. This yields a contradiction as X is a hole cut.

To prove the second claim, suppose there is a hole C' in $G - (X' \cup Y')$. By Proposition 11 there are $c_1, c_2, c_3 \in C'$ such that $c_1 <_{cl} X' <_{cl} c_3$ and $c_1, c_3 \in N(c_2)$. However, this yields $c_2 \in Y'$, which is a contradiction. ◀

4 Proof of Theorem 1

The aim of this section is to provide a complete proof of Theorem 1 using structural results from the previous section. Let us start by showing that the Bipartite Permutation Vertex Deletion problem can be decided in polynomial time on almost bipartite permutation graphs.

▶ **Lemma 13.** *Let (G, k) be an instance of* Bipartite Permutation Vertex Deletion *where G is an n-vertex almost bipartite permutation graph. Then* Bipartite Permutation Vertex Deletion *can be decided in time $\mathcal{O}(n^6)$.*

Proof. If G is a bipartite permutation graph, (G, k) is a **yes**-instance, thus, we are done in this case. If G is not connected, we can consider each connected component independently and, at the end, we compare k with the total number of deleted vertices over all components. Let G' be a connected r-vertex component of G such that G' is not a bipartite permutation graph (otherwise, clearly, no vertex needs to be deleted). Let $C = \{c_0, \ldots, c_{m-1}\}$ be a shortest hole in G' (it exists as G' is not a bipartite permutation graph). It can be found in time $\mathcal{O}(r^6)$ as follows. We iterate over all possible four-element subsets $S = \{v_1, v_2, v_3, v_4\}$ of $V(G')$. For these S for which $G'[S]$ is an induced P_4, with consecutive vertices v_1, v_2, v_3, v_4, we construct a graph $\widetilde{G'}$ by removing the vertices from $(N(v_2) \cup N(v_3)) \setminus \{v_1, v_4\}$ (note that v_2 and v_3 also get removed). Then we find a shortest v_1-v_4-path in $\widetilde{G'}$ in time $\mathcal{O}(r^2)$.

By Proposition 12, every minimal hole cut X in G' is contained in the set $V' = V_{G'}[i-2, i+2]$ for some $i \in [m-1]$. Therefore, we may check all the possibilities where a minimal cut is contained. For every i, we run an algorithm for finding a maximum flow in the following digraph H_i.

Digraph H_i has the vertex set $V' \times \{\mathsf{in}, \mathsf{out}\} \cup \{s, t\}$ and arc set consisting of:

- all arcs of the form $(u, \mathsf{out})(v, \mathsf{in})$, where uv is an edge of $G'[V']$,
- $s(v, \mathsf{in})$ if there exists $u \in V_{G'}[i-4, i-3]$ such that uv is an edge of G',
- $(u, \mathsf{out})t$ if there exists $v \in V_{G'}[i+3, i+4]$ such that uv is an edge of G',
- $(u, \mathsf{in})(u, \mathsf{out})$ for all $u \in V'$.

Set capacities of arcs of the form $(u, \mathsf{in})(u, \mathsf{out})$ to 1 and capacities of all the remaining arcs to ∞ (practically $|V_{G'}|$). It is readily seen that minimum (s,t)-cut in the defined network H_i corresponds to minimum hole cut in $G'[V']$ (arc of unit capacity $(u, \mathsf{in})(u, \mathsf{out})$ naturally corresponds to the vertex u of G').

Therefore it remains to apply classical max-flow algorithm to each H_i for $i \in [m-1]$ and remember the smallest size $k_{G'}$ of minimal (s,t)-cuts. This can be performed in time $\mathcal{O}(m \cdot (|V'|+2) \cdot (|E_{G'[V']}| + 2|V'|)^2) = \mathcal{O}(r^6)$ [9]. Finally, (G,k) is a **yes**-instance if and only if the sum of remembered sizes $k_{G'}$ over the all considered connected components G' is at most k. Clearly, the total running time is $\mathcal{O}(n^6)$. ◀

We now propose the algorithm. Given an n-vertex graph $G = (V,E)$ and number k, we want to answer the Bipartite Permutation Vertex Deletion problem. We say that (G,k) is the *initial* instance. We split our algorithm into two parts. The first part consists of a branching algorithm for deletion to almost bipartite permutation graphs. The output of the first part is a set of instances (G',k') where G' is an almost bipartite permutation graph and $0 \leq k' \leq k$ (or **no**-answer is no such instance exists) such that the initial instance (G,k) is a **yes**-instance if and only if at least one of these instances is a **yes**-instance. We show that the overall time of the first phase is $\mathcal{O}(n^9 \cdot 9^k)$. In the second part, the algorithm runs an $\mathcal{O}(s^6)$-time algorithm for Bipartite Permutation Vertex Deletion where the initial graph is already an almost bipartite permutation graph.

Let us start with the first part. We say that $X \subseteq V$ is a *forbidden set* if $G[X]$ is isomorphic to one of the graphs: $K_3, T_2, X_2, X_3, C_5, C_6, C_7, C_8, C_9$. We define the following rule.

Rule: Given an instance (G,k), $k \geq 1$, and a minimal forbidden set X, branch into $|X|$ instances, $(G-v, k-1)$ for each $v \in X$.

Starting with the initial instance, the algorithm applies the rule exhaustively. In other words, the algorithm is a branching tree with leaves corresponding to instances (G',k') where $k'=0$ or G' is an almost bipartite permutation graph. Clearly, as at least one vertex from each forbidden set must be removed from G, the initial instance is a **yes**-instance if and only if at least one of the leaves is a **yes**-instance.

The algorithm continues to the second part only with such leaves (G',k') that G' is an almost bipartite permutation graph (as otherwise, the leaf is **no**-instance). It runs the algorithm described in Lemma 13 to find if G' can be transformed into a bipartite permutation graph by using at most k' vertex deletions. It either finds a **yes**-instance or concludes after checking all the instances that there is no solution; that is, the initial instance is a **no**-instance.

We note that such a branching into a bounded number of smaller instances is a standard technique, see e.g., [24] for more details.

We now analyze the running time of the whole algorithm. In the first part, observe that the branching tree has depth at most k and has at most 9^k leaves, as k decreases by one whenever the algorithm branches and each of the listed forbidden subgraphs has at most nine vertices. Therefore the total number of nodes in the branching tree is $\mathcal{O}(9^k)$. Moreover, in each node (G'',k''), the algorithm works in time $\mathcal{O}(n^9)$ as it checks if G'' contains a forbidden set. In the second part, the algorithm does a work $\mathcal{O}(n^6)$ in each leaf, by Lemma 13. We conclude that the total running time of our algorithm for Bipartite Permutation Vertex Deletion is $\mathcal{O}(9^k \cdot n^9)$.

5 Proof of Theorem 2

In this section, we provide a proof of Theorem 2. The idea of the algorithm is very similar to the FPT algorithm described in Section 4.

Let $G = (V, E)$ be a graph and let $Y \subseteq V$ be a subset of vertices of G such that $G - Y$ is a bipartite permutation graph. We want to construct a set $Z \subseteq V$ in polynomial time such that $G - Z$ is a bipartite permutation graph and $|Z| \leqslant 9|Y|$. We construct Z as follows. We start with $Z = \emptyset$. Then, as long as $G - Z$ contains a set X isomorphic to one of $K_3, T_2, X_2, X_3, C_5, C_6, C_7, C_8, C_9$ we add all vertices of X to Z. Observe that $Y \cap X \neq \emptyset$ and $|X| \leqslant 9$.

After this step $G - Z$ is an almost bipartite permutation graph. Note that $|Z| \leqslant 9|Z \cap Y|$. We find a shortest hole $C = \{c_0, \ldots, c_{m-1}\}$ in $G - Z$ and find a minimum hole cut X as described in Section 4. Since $(Y - Z)$ is a hole cut in $G - Z$ we have $|X| \leqslant |Y - Z|$. We add X to Z. Observe that $G - Z$ is a bipartite permutation graph.

Since $K_3, T_2, X_2, X_3, C_5, C_6, C_7, C_8, C_9$ have at most 9 vertices, we have that $|Z| \leqslant 9|Y|$. This implies that the above algorithm is a 9-approximation algorithm. It runs in polynomial time because finding small forbidden subgraphs can be done in polynomial time and finding minimum hole cut in an almost bipartite permutation graph can be done in polynomial time.

6 Conclusion

In this paper we investigate for the first time the modification problems in graph classes related to partial orders. Our main result says that the bipartite permutation vertex deletion problem is fixed parameter tractable. We leave open the following two questions that inspired our research.

▶ **Problem 1.** *What is the parameterized status of the vertex deletion problems to the class of permutation graphs and to the class of co-comparability graphs?*

We recall that, due to the result of Lewis and Yannakakis [17], both of these problems are NP-complete. One of the most important result of our work is the description of the structure of almost bipartite permutation graphs, which are defined as graphs which do not induce small graphs from the list of forbidden structures for bipartite permutation graphs. In a similar fashion we can define the class of *almost permutation* and *almost co-comparability graphs*. The next two questions seem very natural in order to solve Problem 1.

▶ **Problem 2.** *What is the structure of almost permutation and almost co-comparability graphs?*

We are aware that the two problems mentioned above can be quite difficult. Therefore, it is worth considering intermediate problems that may be easier to attack. One of the proposed simplifications relies on the transition from the world of graphs to the world of posets. The following *vertex deletion into two-dimensional posets* problem seems very natural in the context of our research: we are given in the input a poset P and a number k and we ask whether we can delete at most k points from P so that the remaining points induce a two-dimensional poset in P.

▶ **Problem 3.** *What is the parameterized status of the vertex deletion into two-dimensional poset problem?*

Since permutation graphs are co-comparability graphs of two-dimensional posets and since permutation graphs are both comparability and co-comparability graphs, the vertex deletion

into two-dimensional poset problem is equivalent to the vertex deletion into co-comparability graph (or into permutation graph) problem if we assume that only comparability graphs can be given in the input. The class of two-dimensional posets is very well understood; in particular, the list of minimal forbidden structures for this class of posets, which is still infinite, is known (obtained independently by Trotter and Moore [23] and by Kelly [15]). Of course, it is natural to ask the following question:

▶ **Problem 4.** *What is the structure of almost two-dimensional posets?*

Since the comparability graphs of posets do not contain odd holes of size $\geqslant 5$, we know the structure of almost two-dimensional posets that are bipartite. Indeed, these are the posets whose comparability graphs are almost bipartite permutation graphs embeddable into cylinder stripes. The last problem we want to ask is as follows:

▶ **Problem 5.** *Is there a polynomial kernel for the bipartite permutation vertex deletion problem?*

A positive answer to this question obtained by indicating so-called *irrelevant vertices* may give some hope to solve Problem 1 with the use of irrelevant vertex technique.

References

1 Kirby A Baker, Peter C Fishburn, and Fred S Roberts. Partial orders of dimension 2. *Networks*, 2(1):11–28, 1972.

2 Andreas Brandstädt and Dieter Kratsch. On the restriction of some NP-complete graph problems to permutation graphs. In *Fundamentals of Computation Theory, FCT '85, Cottbus, GDR, September 9-13, 1985*, volume 199 of *Lecture Notes in Computer Science*, pages 53–62. Springer, 1985. doi:10.1007/BFb0028791.

3 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

4 Yixin Cao. Linear recognition of almost interval graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1096–1115. SIAM, 2016. doi:10.1137/1.9781611974331.ch77.

5 Yixin Cao. Unit interval editing is fixed-parameter tractable. *Information and Computation*, 253:109–126, 2017.

6 Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016.

7 Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of Mathematics*, pages 51–229, 2006.

8 Jitender S. Deogun and George Steiner. Polynomial algorithms for hamiltonian cycle in cocomparability graphs. *SIAM J. Comput.*, 23(3):520–552, 1994. doi:10.1137/S0097539791200375.

9 Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

10 Tibor Gallai. Transitiv orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(1-2):25–66, 1967.

11 Martin Charles Golumbic, Doron Rotem, and Jorge Urrutia. Comparability graphs and intersection graphs. *Discret. Math.*, 43(1):37–46, 1983. doi:10.1016/0012-365X(83)90019-5.

12 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988. doi:10.1007/978-3-642-97881-4.

13 Pinar Heggernes, Pim van 't Hof, Bart M. P. Jansen, Stefan Kratsch, and Yngve Villanger. Parameterized complexity of vertex deletion into perfect graph classes. *Theor. Comput. Sci.*, 511:172–180, 2013. doi:10.1016/j.tcs.2012.03.013.

14 Pinar Heggernes, Pim van 't Hof, Daniel Lokshtanov, and Jesper Nederlof. Computing the cutwidth of bipartite permutation graphs in linear time. *SIAM J. Discret. Math.*, 26(3):1008–1021, 2012. doi:10.1137/110830514.

15 David Kelly. The 3-irreducible partially ordered sets. *Can. J. of Math.*, 29:367–383, 1977.

16 C Lekkeikerker and J Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962.

17 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

18 Daniel Lokshtanov. Wheel-free deletion is $W[2]$-hard. In Martin Grohe and Rolf Niedermeier, editors, *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, volume 5018 of *Lecture Notes in Computer Science*, pages 141–147. Springer, 2008. doi:10.1007/978-3-540-79723-4_14.

19 Federico Mancini. Graph modification problems related to graph classes. *PhD degree dissertation, University of Bergen Norway*, 2, 2008.

20 Dániel Marx. Chordal deletion is fixed-parameter tractable. *Algorithmica*, 57(4):747–768, 2010. doi:10.1007/s00453-008-9233-8.

21 Amir Pnueli, Abraham Lempel, and Shimon Even. Transitive orientation of graphs and identification of permutation graphs. *Canadian Journal of Mathematics*, 23(1):160–175, 1971.

22 Jeremy P. Spinrad, Andreas Brandstädt, and Lorna Stewart. Bipartite permutation graphs. *Discret. Appl. Math.*, 18(3):279–292, 1987. doi:10.1016/S0166-218X(87)80003-3.

23 William T. Trotter and John I. Moore Jr. Characterization problems for graphs, partially ordered sets, lattices, and families of sets. *Discret. Math.*, 16(4):361–381, 1976. doi:10.1016/S0012-365X(76)80011-8.

24 Pim van 't Hof and Yngve Villanger. Proper interval vertex deletion. *Algorithmica*, 65(4):845–867, 2013. doi:10.1007/s00453-012-9661-3.

25 Yngve Villanger, Pinar Heggernes, Christophe Paul, and Jan Arne Telle. Interval completion is fixed parameter tractable. *SIAM J. Comput.*, 38(5):2007–2020, 2009. doi:10.1137/070710913.

Bounding the Mim-Width of Hereditary Graph Classes

Nick Brettell
School of Mathematics and Statistics, Victoria University of Wellington, New Zealand
nick.brettell@vuw.ac.nz

Jake Horsfield
School of Computing, University of Leeds, UK
sc15jh@leeds.ac.uk

Andrea Munaro
School of Mathematics and Physics, Queen's University Belfast, UK
a.munaro@qub.ac.uk

Giacomo Paesani
Department of Computer Science, Durham University, UK
giacomo.paesani@durham.ac.uk

Daniël Paulusma
Department of Computer Science, Durham University, UK
daniel.paulusma@durham.ac.uk

Abstract

A large number of NP-hard graph problems are solvable in XP time when parameterized by some width parameter. Hence, when solving problems on special graph classes, it is helpful to know if the graph class under consideration has bounded width. In this paper we consider mim-width, a particularly general width parameter that has a number of algorithmic applications whenever a decomposition is "quickly computable" for the graph class under consideration.

We start by extending the toolkit for proving (un)boundedness of mim-width of graph classes. By combining our new techniques with known ones we then initiate a systematic study into bounding mim-width from the perspective of hereditary graph classes, and make a comparison with clique-width, a more restrictive width parameter that has been well studied.

We prove that for a given graph H, the class of H-free graphs has bounded mim-width if and only if it has bounded clique-width. We show that the same is not true for (H_1, H_2)-free graphs. We identify several general classes of (H_1, H_2)-free graphs having unbounded clique-width, but bounded mim-width, illustrating the power of mim-width. Moreover, we show that a branch decomposition of constant mim-width can be found in polynomial time, for these classes. Hence, as mentioned, these results have algorithmic implications: when the input is restricted to such a class of (H_1, H_2)-free graphs, many problems become polynomial-time solvable, including classical problems such as k-Colouring and Independent Set, domination-type problems known as LC-VSVP problems, and distance versions of LC-VSVP problems, to name just a few. We also prove a number of new results showing that, for certain H_1 and H_2, the class of (H_1, H_2)-free graphs has unbounded mim-width.

Boundedness of clique-width implies boundedness of mim-width. By combining our results, which give both new bounded and unbounded cases for mim-width, with the known bounded cases for clique-width, we present summary theorems of the current state of the art for the boundedness of mim-width for (H_1, H_2)-free graphs. In particular, we classify the mim-width of (H_1, H_2)-free graphs for all pairs (H_1, H_2) with $|V(H_1)| + |V(H_2)| \leq 8$. When H_1 and H_2 are connected graphs, we classify all pairs (H_1, H_2) except for one remaining infinite family and a few isolated cases.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Width parameter, mim-width, clique-width, hereditary graph class

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.6

Funding The research in this paper received support from the Leverhulme Trust (RPG-2016-258).

1 Introduction

Many computationally hard graph problems can be solved efficiently after placing appropriate restrictions on the input graph. Instead of trying to solve individual problems in an ad hoc way, one may aim to find the underlying reasons why some sets of problems behave better on certain graph classes than other sets of problems. The ultimate goal in this type of research is to obtain complexity dichotomies for large families of graph problems. Such dichotomies tell us for which graph classes a certain problem or set of problems can or cannot be solved efficiently (under standard complexity assumptions).

One reason that might explain the jump from computational hardness to tractability after restricting the input to some graph class $\mathcal{G}$ is that $\mathcal{G}$ has bounded "width", that is, every graph in $\mathcal{G}$ has width at most c for some constant c. One can define the notion of "width" in many different ways (see the surveys [30, 31, 37, 47]). As such, the various width parameters differ in strength. To explain this, we say that a width parameter p *dominates* a width parameter q if there is a function f such that $p(G) \leq f(q(G))$ for all graphs G. If p dominates q but q does not dominate p, then p is said to be *more powerful* than q. If both p and q dominate each other, then p and q are *equivalent*. For instance, the width parameters boolean-width, clique-width, module-width, NLC-width and rank-width are all equivalent [15, 36, 42, 44], but more powerful than the equivalent parameters branch-width and treewidth [19, 45, 47].

In this paper we focus on an even more powerful width parameter called *mim-width* (maximum induced matching width). Vatshelle [47] introduced mim-width, which we define in Section 3, and proved that mim-width is more powerful than boolean-width, and consequently, clique-width, module-width, NLC-width and rank-width.

1.1 Algorithmic Implications

One trade-off of a more powerful width parameter is the difficulty in obtaining a branch decomposition of bounded width. In general, computing mim-width is NP-hard; deciding if the mim-width is at most k is W[1]-hard when parameterized by k; and there is no polynomial-time algorithm for approximating the mim-width of a graph to within a constant factor of the optimal, unless NP = ZPP [46]. Hence, in contrast to algorithms for graphs of bounded treewidth or clique-width, algorithms for graphs of bounded mim-width require a branch decomposition of constant mim-width as part of the input. On the other hand, there are many interesting graph classes for which mim-width is bounded and *quickly computable*, that is, the class admits a polynomial-time algorithm for obtaining a branch decomposition of constant mim-width. We give examples of such graph classes known in the literature in Section 1.2 before discussing the new graph classes we found in Section 1.4. Below we briefly discuss known algorithms for problems on graphs of bounded mim-width.

Belmonte and Vatshelle [1] and Bui-Xuan, Telle and Vatshelle [16] proved that a large set of problems, known as Locally Checkable Vertex Subset and Vertex Partitioning (LC-VSVP) problems [43], can be solved in polynomial time for graph classes where mim-width is bounded and quickly computable. Well-known examples of such problems include (Total) Dominating Set, Independent Set and k-Colouring for every fixed positive integer k. Later, Fomin, Golovach and Raymond [27] proved that the XP algorithms for Independent Set and Dominating Set are in a sense best possible, showing that these two problems are W[1]-hard when parameterized by mim-width.

On the positive side, XP algorithms parameterized by mim-width are now also known for problems outside the LC-VSVP framework. In particular, Jaffke, Kwon, Strømme and Telle [33] proved that the distance versions of LC-VSVP problems can be solved in polynomial

time for graph classes where mim-width is bounded and quickly computable. Jaffke, Kwon and Telle [34, 35] proved similar results for Longest Induced Path, Induced Disjoint Paths, H-Induced Topological Minor and Feedback Vertex Set. The latter result has recently been generalized to Subset Feedback Vertex Set and Node Multiway Cut, by Bergougnoux, Papadopoulos and Telle [3].

Bergougnoux and Kanté [2] gave a meta-algorithm for problems with a global constraint, providing unifying XP algorithms in mim-width for several of the aforementioned problems, as well as Connected Dominating Set, Node Weighted Steiner Tree, and Maximum Induced Tree. Galby, Munaro and Ries [29] proved that Semitotal Dominating Set is polynomial-time solvable for graph classes where mim-width is bounded and quickly computable.

1.2 Mim-width of Special Graph Classes

Belmonte and Vatshelle [1] proved that the mim-width of the following graph classes is bounded and quickly computable: permutation graphs, convex graphs and their complements, interval graphs and their complements, circular k-trapezoid graphs, circular permutation graphs, Dilworth-k graphs, k-polygon graphs, circular-arc graphs and complements of d-degenerate graphs.

Some of the results of [1] have been extended. Let $K_r \boxminus K_r$ be the graph obtained from $2K_r$ by adding a perfect matching, and let $K_r \boxminus rP_1$ be the graph obtained from $K_r \boxminus K_r$ by removing all the edges in one of the complete graphs (see Section 2 for undefined notation). Kang et al. [38] showed that for any integer $r \geq 2$, there is a polynomial-time algorithm for computing a branch decomposition of mim-width at most $r-1$ when the input is restricted to $(K_r \boxminus rP_1)$-free chordal graphs, which generalize interval graphs, or $(K_r \boxminus K_r)$-free co-comparability graphs, which generalize permutation graphs. Hence, in particular, all these classes have bounded mim-width.

In addition to the above results, Kang et al. [38] proved that chordal graphs, circle graphs and co-comparability graphs have unbounded mim-width; Mengel [41] also proved, independently, that the latter two classes have unbounded mim-width. Vatshelle [47] and Brault-Baron et al. [11] showed the same for grids and chordal bipartite graphs, respectively, whereas Mengel [41] proved that strongly chordal split graphs have unbounded mim-width.

Brettell et al. [12] showed that the mim-width of $(K_r, sP_1 + P_5)$-free graphs is bounded and quickly computable for every $r \geq 1$ and $s \geq 0$. In particular, this yielded an alternative proof for showing that List k-Colouring is polynomially solvable for $(sP_1 + P_5)$-free graphs for all $k \geq 1$ and $s \geq 0$ [20]. Let $K^1_{1,s}$ be the graph obtained from the $(s+1)$-vertex star $K_{1,s}$ after subdividing each edge once; note that $sP_1 + P_5$ is an induced subgraph of $K^1_{1,s+2}$. In [13], the result of [12] on the mim-width of $(K_r, sP_1 + P_5)$-free graphs was generalized to $(K_r, K^1_{1,s}, P_t)$-free graphs. As a consequence, for all $k \geq 3$, $s \geq 1$ and $t \geq 1$, List k-Colouring is polynomial-time solvable even for $(K^1_{1,s}, P_t)$-free graphs; previously this was shown for $k = 3$ by Chudnovsky et al. [18].

Brettell et al. [14] considered the following generalisation of convex graphs. A bipartite graph $G = (A, B, E)$ is $\mathcal{H}$-convex, for some family of graphs $\mathcal{H}$, if there exists a graph $H \in \mathcal{H}$ with $V(H) = A$ such that the set of neighbours in A of each $b \in B$ induces a connected subgraph of H (when $\mathcal{H}$ is the set of paths, we obtain exactly convex graphs). They showed that the class of $\mathcal{H}$-convex graphs has bounded and quickly computable mim-width if $\mathcal{H}$ is the set of cycles, or $\mathcal{H}$ is the set of trees with bounded maximum degree and bounded number of vertices of degree at least 3.

1.3 Our Focus

We continue the study on boundedness of mim-width and aim to identify more graph classes of bounded or unbounded mim-width. Our motivation is both algorithmic and structural. As discussed above, there are clear algorithmic benefits if a graph class has bounded mim-width. From a structural point of view, we aim to initiate a *systematic* study of the boundedness of mim-width, comparable to a similar, long-standing study of the boundedness of clique-width (see [23] for a survey).

The framework of hereditary graph classes is highly suitable for such a study. A graph class $\mathcal{G}$ is *hereditary* if it is closed under vertex deletion. A class $\mathcal{G}$ is hereditary if and only if there exists a (unique) set of graphs $\mathcal{F}$ of (minimal) forbidden induced subgraphs for $\mathcal{G}$. That is, a graph G belongs to $\mathcal{G}$ if and only if G does not contain any graph from $\mathcal{F}$ as an induced subgraph. We also say that G is *$\mathcal{F}$-free*. Note that $\mathcal{F}$ may have infinite size. For example, if $\mathcal{G}$ is the class of bipartite graphs, then $\mathcal{F}$ is the set of all odd cycles.

As a natural starting point we consider the case where $|\mathcal{F}| = 1$, say $\mathcal{F} = \{H\}$. It is not difficult to verify that a class of H-free graphs has bounded mim-width if and only if it has bounded clique-width if and only if H is an induced subgraph of the 4-vertex path P_4; see Section 3 for details. On the other hand, there exist hereditary graph classes, such as interval graphs and permutation graphs, that have bounded mim-width, even mim-width 1 [47], but unbounded clique-width [32]. However, these graph classes have an *infinite* set of forbidden induced subgraphs. Hence, questions we aim to address in this paper are: Does there exist a hereditary graph class characterized by a finite set $\mathcal{F}$ that has bounded mim-width but unbounded clique-width? Can we use the same techniques as when dealing with clique-width? In particular we will focus on the case where $|\mathcal{F}| = 2$, say $\mathcal{F} = \{H_1, H_2\}$. Such classes are called *bigenic*.

1.4 Our Results and Methodology

In order to work with width parameters it is useful to have a set of graph operations that *preserve* boundedness or unboundedness of the width parameter. That is, if we apply such a width-preserving operation, or only apply it a constant number of times, the width of the graph does not change by too much. In this way one might be able to modify an arbitrary graph from a given "unknown" class $\mathcal{G}_1$ into a graph from a class $\mathcal{G}_2$ known to have bounded or unbounded width. This would then imply that $\mathcal{G}_1$ also has bounded or unbounded width, respectively. Two useful operations preserving clique-width are vertex deletion [40] and subgraph complementation [37]. The latter operation replaces every edge in some subgraph of the graph by a non-edge, and vice versa. As we will see in Section 6, subgraph complementation does not preserve boundedness or unboundedness of mim-width[1].

To work around this limitation, we collect and generalize known mim-width preserving graph operations from the literature in Section 3 (some of these operations only show that the mim-width cannot decrease after applying them). In the same section we also state some known useful results on mim-width and prove that elementary graph classes, such as walls and net-walls, have unbounded mim-width. In Sections 4 and 5 we use the results from Section 3. In Section 4 we present new bigenic classes of bounded mim-width. These graph classes are all known to have unbounded clique-width. Hence, our results show that the dichotomy for boundedness of mim-width no longer coincides with the one for clique-width when $|\mathcal{F}| = 2$ instead of $|\mathcal{F}| = 1$. Moreover, for each of these classes, a branch decomposition

[1] The situation is different for mim-width 1; Vatshelle [47] showed that if $\mathrm{mimw}(G) = 1$ then $\mathrm{mimw}(\overline{G}) = 1$.

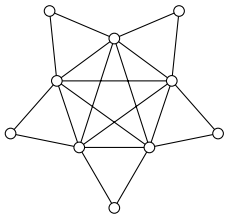


Figure 1 The graph sun_5.

of constant mim-width is easily computable for any graph in the class. This immediately implies that there are polynomial-time algorithms for many problems when restricted to these classes, as described in Section 1.1. In Section 5 we present new bigenic classes of unbounded mim-width; these graph classes are known to have unbounded clique-width. In Section 6 we give a state-of-the-art summary of our new results combined with known results. The known results include the bigenic graph classes of bounded clique-width (as bounded clique-width implies bounded mim-width). In the same section we compare our results for the mim-width of bigenic graph classes with the ones for clique-width. We also state a number of open problems.

2 Preliminaries

We consider only finite graphs $G = (V, E)$ with no loops and no multiple edges. For a vertex $v \in V$, the *neighbourhood* $N(v)$ is the set of vertices adjacent to v in G. The *degree* $d(v)$ of a vertex $v \in V$ is the size $|N(v)|$ of its neighbourhood. A graph is *subcubic* if every vertex has degree at most 3. For disjoint $S, T \subseteq V$, we say that S is *complete to* T if every vertex of S is adjacent to every vertex of T, and S is *anticomplete to* T if there are no edges between S and T. The *distance* from a vertex u to a vertex v in G is the length of a shortest path between u and v. A set $S \subseteq V$ *induces* the subgraph $G[S] = (S, \{uv \ : \ u, v \in S, uv \in E\})$. If G' is an induced subgraph of G we write $G' \subseteq_i G$. The *complement* of G is the graph $\overline{G}$ with vertex set $V(G)$, such that $uv \in E(\overline{G})$ if and only if $uv \notin E(G)$.

Given a graph G and a degree-k vertex v of G with $N(v) = \{u_1, \ldots, u_k\}$, the *clique implant on* v is the operation of deleting v, adding k new vertices $v_1, \ldots, v_k$ forming a clique, and adding edges $v_i u_i$ for each $i \in \{1, \ldots, k\}$. The *k-subdivision* of an edge uv in a graph replaces uv by k new vertices $w_1, \ldots, w_k$ with edges $uw_1, w_k v$ and $w_i w_{i+1}$ for each $i \in \{1, \ldots, k-1\}$, i.e. the edge is replaced by a path of length $k+1$. The *disjoint union* $G + H$ of graphs G and H has vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$. We denote the disjoint union of k copies of G by kG. For a graph H, a graph G is *H-free* if G has no induced subgraph isomorphic to H. For a set of graphs $\{H_1, \ldots, H_k\}$, a graph G is *$(H_1, \ldots, H_k)$-free* if G is H_i-free for every $i \in \{1, \ldots, k\}$.

An *independent set* of a graph is a set of pairwise non-adjacent vertices. A *clique* of a graph is a set of pairwise adjacent vertices. A *matching* of a graph is a set of pairwise non-adjacent edges. A matching M of a graph G is *induced* if there are no edges of G between vertices incident to distinct edges of M.

The path, cycle and complete graph on n vertices are denoted by P_n, C_n and K_n, respectively. The graph K_3 is also called the *triangle*. A graph is *r-partite*, for $r \geq 2$, if its vertex set admits a partition into r classes such that every edge has its endpoints in different classes. An r-partite graph in which every two vertices from different partition classes are adjacent is a *complete r-partite graph* and a 2-partite graph is also called *bipartite*.

A graph is *co-bipartite* if it is the complement of a bipartite graph. A *split graph* is a graph G that admits a *split partition* (C, I), that is, $V(G)$ can be partitioned into a clique C and an independent set I. Equivalently, a graph is split if and only if it is $(2P_2, C_4, C_5)$-free. The *subdivided claw* $S_{h,i,j}$, for $1 \leq h \leq i \leq j$ is the tree with one vertex x of degree 3 and exactly three leaves, which are of distance h, i and j from x, respectively. Note that $S_{1,1,1} = K_{1,3}$. For $t \geq 3$, sun_t denotes the graph on $2t$ vertices obtained from a complete graph on t vertices $u_1, \ldots, u_t$ by adding t vertices $v_1, \ldots, v_t$ such that v_i is adjacent to u_i and u_{i+1} for each $i \in \{1, \ldots, t-1\}$ and v_t is adjacent to u_1 and u_t. See Figure 1 for a picture of sun_5.

3 Mim-Width: Definition and Basic Results

A *branch decomposition* for a graph G is a pair (T, δ), where T is a subcubic tree and δ is a bijection from $V(G)$ to the leaves of T. Each edge $e \in E(T)$ naturally partitions the leaves of T into two classes, depending on which component they belong to when e is removed. In this way, each edge $e \in E(T)$ corresponds to a partition L_e and $\overline{L_e}$ of the set of leaves of T, depending on which component of $T - e$ the leaves of T belong to. Consequently, each edge e induces a partition $(A_e, \overline{A_e})$ of $V(G)$, where $\delta(A_e) = L_e$ and $\delta(\overline{A_e}) = \overline{L_e}$. For two disjoint sets X and Y, let $G[X, Y]$ denote the bipartite subgraph of G induced by the edges with one endpoint in X and the other in Y. For each edge $e \in E(T)$ and corresponding partition $(A_e, \overline{A_e})$ of $V(G)$, we denote by $\mathrm{cutmim}_G(A_e, \overline{A_e})$ the size of a maximum induced matching in $G[A_e, \overline{A_e}]$. The *mim-width* of the branch decomposition (T, δ) is the quantity $\mathrm{mimw}_G(T, \delta) = \max_{e \in E(T)} \mathrm{cutmim}_G(A_e, \overline{A_e})$. The *mim-width* of the graph G, denoted $\mathrm{mimw}(G)$, is the minimum value of $\mathrm{mimw}_G(T, \delta)$ over all possible branch decompositions (T, δ) for G.

Mim-Width Preserving Operations. The following three lemmas, the first of which is due to Vatshelle, show that vertex deletion, edge subdivision and clique implantation do not change the mim-width of a graph by too much. We omit the proofs of the second and third lemma.

▶ **Lemma 1** ([47]). *Let G be a graph and $v \in V(G)$. Then* $\mathrm{mimw}(G) - 1 \leq \mathrm{mimw}(G - v) \leq \mathrm{mimw}(G)$.

▶ **Lemma 2.** *Let G be a graph and let G' be the graph obtained by 1-subdividing an edge of G. Then* $\mathrm{mimw}(G) \leq \mathrm{mimw}(G') \leq \mathrm{mimw}(G) + 1$.

▶ **Lemma 3.** *Let G be a graph and let G' be the graph obtained from G by a clique implant on $v \in V(G)$. Then* $\mathrm{mimw}(G) \leq \mathrm{mimw}(G') \leq \mathrm{mimw}(G) + d(v)$.

Mengel [41] showed that adding edges inside the partition classes of a bipartite graph does not decrease mim-width by much. This result can be generalized to k-partite graphs in the following way.

▶ **Lemma 4.** *Let G be a k-partite graph with partition classes $V_1, \ldots, V_k$, and let G' be a graph obtained from G by adding edges where for each added edge, there exists some i such that both endpoints are in V_i. Then* $\mathrm{mimw}(G') \geq \frac{1}{k} \cdot \mathrm{mimw}(G)$.

Proof. Let (T, δ) be a branch decomposition for G'. Since G and G' have the same vertex set, (T, δ) is a branch decomposition for G as well. It is enough to show that $\mathrm{mimw}_G(T, \delta) \leq k \cdot \mathrm{mimw}_{G'}(T, \delta)$. Therefore, let $e \in E(T)$ be such that $\mathrm{mimw}_G(T, \delta) = \mathrm{cutmim}_G(A_e, \overline{A_e})$, and let M be a maximum induced matching in $G[A_e, \overline{A_e}]$. For each i, consider the set

$M_i = \{uv \in M : u \in A_e \cap V_i\}$. These k sets partition M. Let M' be a partition class of size at least $|M|/k$. Clearly, M' is an induced matching in $G'[A_e, \overline{A_e}]$ and so $k \cdot \mathrm{mimw}_{G'}(T,\delta) \geq k \cdot |M'| \geq |M| = \mathrm{mimw}_G(T,\delta)$. ◀

The next lemma shows that to bound the mim-width of a class of graphs, we may restrict our attention to 2-connected graphs in the class. We omit the proof and note that this property is not specific to mim-width: it has also been observed, in [30], for rank-width, and this argument also applies for any appropriate width parameter defined using branch decompositions. A *block* is a maximal connected subgraph with no cut-vertex.

▶ **Lemma 5.** *Let G be a graph. Then* $\mathrm{mimw}(G) = \max\{\mathrm{mimw}(H) : H \text{ is a block of } G\}$. *Moreover, given branch decompositions (T_H,δ_H) of each block H of G, with* $\mathrm{mimw}_H(T_H,\delta_H) \leq k$, *we can compute a branch decomposition of G with mim-width at most k in polynomial time.*

The following lemma is due to Galby and Munaro, who used it to prove that Dominating Set admits a PTAS for a subclass of VPG graphs when the representation is given.

▶ **Lemma 6** ([28]). *Let G be a graph and let $S \subseteq V$. Let $G' = (V', E')$ denote the graph with $V' = V$ and $E' = E \cup \{uv : u, v \in S\}$. Then* $\mathrm{mimw}(G') \leq \mathrm{mimw}(G) + 1$.

The final structural lemma is used to prove that $(sP_1 + P_5, K_t)$-free graphs have bounded mim-width for every $s \geq 0$ and $t \geq 1$. It shows how we can bound the mim-width of a graph in terms of the mim-width of the graphs induced by blocks of a partition of the vertex set and the mim-width between any two of the parts. We include it here as it might be useful for bounding the mim-width of other graph classes.

▶ **Lemma 7** ([12]). *Let G be a graph and $(X_1, \ldots, X_p)$ be a partition of $V(G)$ such that* $\mathrm{cutmim}_G(X_i, X_j) \leq c$ *for all distinct $i, j \in \{1, \ldots, p\}$, and $p \geq 2$. Then*

$$\mathrm{mimw}(G) \leq \max\left\{ c \left\lfloor \left(\frac{p}{2}\right)^2 \right\rfloor, \max_{i \in \{1,\ldots,p\}} \{\mathrm{mimw}(G[X_i])\} + c(p-1) \right\}.$$

Moreover, if (T_i, δ_i) is a branch decomposition of $G[X_i]$ for each i, then we can construct, in $O(1)$ time, a branch decomposition (T, δ) of G with

$$\mathrm{mimw}_G(T,\delta) \leq \max\{c\lfloor(\tfrac{p}{2})^2\rfloor, \max_{i \in \{1,\ldots,p\}} \{\mathrm{mimw}_G(T_i,\delta_i)\} + c(p-1)\}.$$

Mim-width of Some Basic Classes. Recall that Vatshelle [47] showed that the class of grids has unbounded mim-width. We next prove that the same holds for the class of walls, which we define momentarily. Thus, we obtain a class of graphs with maximum degree 3 having unbounded mim-width, and we will use this result in order to prove Lemma 11. Note that it also gives us a dichotomy, as graphs with maximum degree 2 have bounded clique-width and hence bounded mim-width.

A *wall of height h and width r* (an *$(h \times r)$-wall* for short) is the graph obtained from the grid of height h and width $2r$ as follows. Let $C_1, \ldots, C_{2r}$ be the set of vertices in each of the $2r$ columns of the grid, in their natural left-to-right order. For each column C_j, let $e^j_1, e^j_2, \ldots, e^j_{h-1}$ be the edges between two vertices of C_j, in their natural top-to-bottom order. If j is odd, we delete all edges e^j_i with i even. If j is even, we delete all edges e^j_i with i odd. We then remove all vertices of the resulting graph whose degree is 1. This final graph is an *elementary $(h \times r)$-wall* and any subdivision of the elementary $(h \times r)$-wall is an *$(h \times r)$-wall*. For an example, see Figure 2.

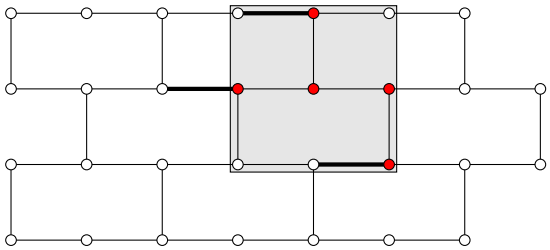


Figure 2 An elementary (4×4)-wall. We illustrate an example of the case where $h \geq \sqrt[4]{n(W)/3}$ and $r < 2n$ in the proof of Theorem 8: Q consists of the red vertices, B is the the grey box, and the thick edges are a matching in $W[A_e, \overline{A_e}]$.

▶ **Theorem 8.** *Let W be an elementary $(n \times n)$-wall with $n \geq 7$. Then* $\mathrm{mimw}(W) \geq \frac{\sqrt{n}}{50}$. *In particular, the class of walls has unbounded mim-width.*

Proof. We let $n(W) = |V(W)| = 2n^2 - 2$. Consider now a branch decomposition (T, δ) for W. There exists $e \in E(T)$ such that both partition classes A_e and $\overline{A_e}$ of $V(W)$ contain at least $n(W)/3$ vertices [38, Lemma 2.3]. Kanj et al. [39, Lemma 4.10] showed that if G is a graph such that each of its subgraphs has average degree at most d, then any matching M in G contains an induced matching in G of size at least $|M|/(2d-1)$. Since W is subcubic, it is sufficient to show that $W[A_e, \overline{A_e}]$ has a matching of size $\sqrt{n}/10$. We distinguish two cases, according to whether or not one of $W[A_e]$ and $W[\overline{A_e}]$ has a component of size at least $\sqrt{n(W)/3}$.

Suppose first that $W[A_e]$ has a component Q of size at least $\sqrt{n(W)/3}$. The component Q is contained in a rectangle of the underlying $n \times 2n$ grid. Consider the smallest such rectangle B, i.e., the rectangle whose horizontal sides contain the uppermost and lowermost vertex in Q and whose vertical sides contain the leftmost and rightmost vertex in Q. Let h and r be the height and width of B, respectively. Since $|V(Q)| \geq \sqrt{n(W)/3}$, one of h and r is at least $\sqrt[4]{n(W)/3}$.

Suppose first that $h \geq \sqrt[4]{n(W)/3}$. If $r < 2n$, say without loss of generality B does not intersect column C_1, we do the following. For each row of B, consider the leftmost vertex of Q in that row (since Q is connected, each row contains at least one vertex of Q). Clearly, the left neighbours of each such vertex belongs to $\overline{A_e}$, and so we have a matching in $W[A_e, \overline{A_e}]$ of size $h - 2 \geq \sqrt[4]{n(W)/3} - 2$, which is at least $\sqrt{n}/10$ when $n \geq 7$. If $r = 2n$, we distinguish two cases according to whether $h = n$ or not. In the first case (i.e., $r = 2n$ and $h = n$) we argue as follows. Since Q is connected, each row of B contains a vertex of $Q \subseteq A_e$. Moreover, there are at most $2n/3$ rows of B with all vertices contained in A_e, for otherwise $|A_e| > (2n/3) \cdot 2n \geq 2n(W)/3$. So there are at least $n/3$ rows of B containing a vertex of A_e and a vertex of $\overline{A_e}$. We can therefore find a matching in $W[A_e, \overline{A_e}]$ of size at least $n/3$. In the second case (i.e., $r = 2n$ and $h < n$), we proceed as follows. We assume, without loss of generality, that B does not intersect the uppermost row of the grid. We partition the columns of B into disjoint layers containing two consecutive columns each. For each layer, we consider its left column and the uppermost vertex $v \in A_e$ therein (since Q is connected, such a vertex exists). Let v_1 be the vertex on the grid above v, let v_2 be the vertex to the right of v and let v_3 be the vertex above v_2. By construction, $v_1 \in \overline{A_e}$ and if $vv_1 \in E(W)$, we select this edge. Otherwise, $vv_1 \notin E(W)$ and so $v_2v_3 \in E(W)$ and we have a path $vv_2v_3v_1$ in W with $v \in A_e$ and $v_1 \in \overline{A_e}$. We then select an edge of this path which belongs to $W[A_e, \overline{A_e}]$. Proceeding similarly for each layer, we obtain a matching in $W[A_e, \overline{A_e}]$ of size at least $r/2 = n$. Suppose finally that $h < \sqrt[4]{n(W)/3}$. We have that $r \geq \sqrt[4]{n(W)/3}$ and we proceed exactly as in the case $r = 2n$ and $h < n$ to obtain a matching in $W[A_e, \overline{A_e}]$ of size

at least $r/2 \geq \sqrt[4]{n(W)/3}/2$.

It remains to consider the situation in which all components of $W[A_e]$ and $W[\overline{A_e}]$ have size less than $\sqrt{n(W)/3}$. In particular, since $W[A_e]$ has more than $n(W)/3$ vertices, it has more than $\sqrt{n(W)/3}$ components. Let $Q_1, \ldots, Q_k$ be these components. For each $i \in \{1, \ldots, k\}$, there exists a vertex $u_i \in Q_i$ with a neighbour $v_i \in \overline{A_e}$, as W is connected. Let H be the subgraph of $W[A_e, \overline{A_e}]$ induced by $\{u_1, \ldots, u_k\} \cup \{v_1, \ldots, v_k\}$ (notice that we might have $v_i = v_j$ for some $i \neq j$). Let $H_1, \ldots, H_\ell$ be the components of H and let $n_i = |V(H_i)|$, for each $i \in \{1, \ldots, \ell\}$. By construction, $n_i \geq 2$, for each i. Moreover, since H_i is a connected subcubic graph, it has a matching of size at least $(n_i - 1)/3 \geq n_i/6$ [4]. But then H has a matching of size

$$\sum_{i=1}^{\ell} \frac{n_i}{6} = \frac{|V(H)|}{6} \geq \frac{k}{6} \geq \frac{1}{6} \cdot \sqrt{\frac{n(W)}{3}}.$$

As in all cases we find a matching in $W[A_e, \overline{A_e}]$ of size at least $\frac{\sqrt{n}}{10}$, this concludes the proof. ◀

▶ **Corollary 9.** *For an integer Δ, let $\mathcal{G}_\Delta$ be the class of graphs of maximum degree at most Δ. Then the mim-width of $\mathcal{G}_\Delta$ is bounded if and only if $\Delta \leq 2$.*

A *net-wall* is a graph that can be obtained from a wall G by performing a clique implant on each vertex of G having degree three. An example of part of a net-wall is given in Figure 4.

The following lemma is a straightforward consequence of Theorem 8 and Lemma 3.

▶ **Lemma 10.** *The class of net-walls has unbounded mim-width.*

Mengel [41] showed that strongly chordal split graphs, or equivalently $(\mathsf{sun}_3, \mathsf{sun}_4, \ldots)$-free split graphs, have unbounded mim-width. We find two more subclasses of split graphs with unbounded mim-width by using Lemmas 2 and 4.

▶ **Lemma 11.** *Let $\mathcal{G}$ be the class of split graphs, or equivalently $(C_4, C_5, 2P_2)$-free graphs, where one of the following properties is satisfied by every $G \in \mathcal{G}$:*

- **(i)** *G has a split partition (C, I) where each vertex in I has degree 2 and each vertex in C has at most three neighbours in I,*
- **(ii)** *G has a split partition (C, I) where each vertex in I has degree at most 3, and each vertex in C has two neighbours in I, or*
- **(iii)** *G is sun_t-free $t \geq 3$.*

Then $\mathcal{G}$ has unbounded mim-width.

Proof. Statement (iii) is due to Mengel [41]. To prove (i) and (ii), let G be a wall, and let G' be the graph obtained by 1-subdividing each edge of G. Partition $V(G')$ into (A, B), where B consists of the vertices of degree two introduced by the 1-subdivisions. Observe that G' is bipartite, with vertex bipartition (A, B). Let G'' be the graph obtained by making one of A or B a clique. By Lemmas 2 and 4, $\mathrm{mimw}(G'') \geq \mathrm{mimw}(G)/2$. The result now follows from Theorem 8. ◀

A graph is *chordal bipartite* if it is bipartite and every induced cycle has four vertices. Brault-Baron et al. [11] showed that the class of chordal bipartite graphs has unbounded mim-width. Combining their result with Lemma 4, after adding all edges in a colour class, yields the following:

▶ **Lemma 12.** *The class of co-bipartite graphs, or equivalently $(3P_1, C_5, \overline{C_7}, \overline{C_9}, \ldots)$-free graphs, has unbounded mim-width.*

As the last result in this section we consider hereditary classes defined by one forbidden induced subgraph. It is folklore that the class of H-free graphs has bounded clique-width if and only if $H \subseteq_i P_4$ (see [26] for a proof). It turns out that the same dichotomy holds for mim-width.

▶ **Theorem 13.** *The class of H-free graphs has bounded mim-width if and only if $H \subseteq_i P_4$.*

Proof. If $H \subseteq_i P_4$, then H-free graphs form a subclass of P_4-free graphs. Every P_4-free graph has clique-width at most 2 [19] and so mim-width at most 2 [47]. Suppose now that H is a graph such that the class of H-free graphs has bounded mim-width. Recall that chordal bipartite graphs have unbounded mim-width [11] (see also Section 5). Hence, H is C_3-free. As co-bipartite graphs, and thus $3P_1$-free graphs, and split graphs, or equivalently, $(C_4, C_5, 2P_2)$-free graphs, have unbounded mim-width by Lemmas 11 and 12, this means that H is a $(3P_1, 2P_2)$-free forest. It follows that $H \subseteq_i P_4$. ◀

4 New Bounded Cases

In this section, we present three general classes, and two further specific classes, of (H_1, H_2)-free graphs having bounded mim-width, but unbounded clique-width. First, we present the three infinite families of classes of (H_1, H_2)-free graphs. We show that for a class in one of these three families, there exists a constant k such that for every graph G in the class, and every $X \subseteq V(G)$, we have that $\text{cutmim}_G(X, \overline{X}) \leq k$. This implies that every branch decomposition of G has mim-width at most k. Thus, *for a graph in one of these classes, a branch decomposition of constant mim-width is quickly computable*: any branch decomposition will suffice. Finally, we present two more classes of (H_1, H_2)-free graphs having bounded mim-width, which do not have this property, but for which we prove that a branch decomposition of constant width can be computed in polynomial-time.

We make use of Ramsey theory. By Ramsey's Theorem, for all positive integers a and b, there exists an integer $R(a, b)$ such that if G is a graph on at least $R(a, b)$ vertices, then G has either a clique of size a, or an independent set of size b.

Recall that $K_r \boxminus K_r$ is the graph obtained from $2K_r$ by adding a perfect matching and that $K_r \boxminus rP_1$ is the graph obtained from $K_r \boxminus K_r$ by removing all the edges in one of the complete graphs. We let $K_r \boxminus P_1$ denote the graph obtained from K_r by adding a single vertex, attached to K_r by a single pendant edge. We also denote $\overline{C_4 + P_1}$ as bowtie. Examples of these graphs are given in Figure 3.

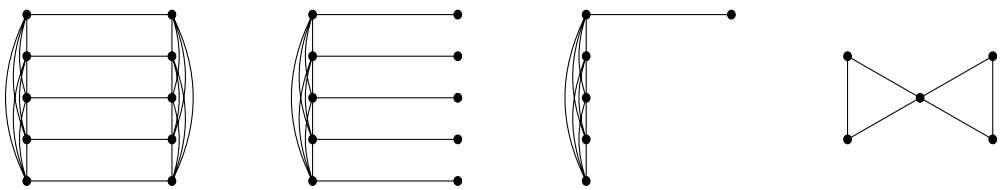


Figure 3 The graphs $K_5 \boxminus K_5$, $K_5 \boxminus 5P_1$, $K_5 \boxminus P_1$, and bowtie $= \overline{C_4 + P_1}$.

▶ **Theorem 14.** *Let G be a $(K_r \boxminus rP_1, 2P_2)$-free graph for $r \geq 3$. Then $\text{cutmim}_G(X, \overline{X}) < \max\{6, r\}$ for every $X \subseteq V(G)$. In particular, $\text{mimw}(G) < \max\{6, r\}$.*

Proof. Let $k = \max\{6, r\}$ and let (T, δ) be a branch decomposition of G. Towards a contradiction, suppose that there exists $X \subseteq V(G)$ such that $G[X, \overline{X}]$ has an induced matching of size at least k. Let $X' = \{x_1, x_2, \ldots, x_k\} \subseteq X$ and $Y' = \{y_1, y_2, \ldots, y_k\} \subseteq \overline{X}$ such that $x_i y_i$ is an edge of the induced matching for each $i \in \{1, 2, \ldots, k\}$.

First, observe that for any distinct $i, j \in \{1, 2, \ldots, k\}$, either x_ix_j or y_iy_j is an edge, otherwise $G[\{x_i, x_j, y_i, y_j\}] \cong 2P_2$. We claim that X' or Y' contains a clique of size 3. Since $|X'| = k \geq 6 = R(3,3)$, the set X' contains either a clique on 3 vertices, or an independent set on 3 vertices. So we may assume that X' contains an independent set on 3 vertices, $\{x_i, x_j, x_\ell\}$ say. Then $\{y_i, y_j, y_\ell\}$ is a clique of size 3 contained in Y', proving the claim.

Without loss of generality, we may now assume that X' contains a clique of size 3. Suppose X' is not a clique. Then there exist distinct $i, j \in \{1, 2, \ldots, k\}$ such that x_i is not adjacent to x_j. Now y_iy_j is an edge, since G is $2P_2$-free. Let X'' be a maximum-sized clique contained in X', so $|X''| \geq 3$. Note that $\{x_i, x_j\} \not\subseteq X''$, since X'' is a clique, so we may assume that $x_j \notin X''$. As any pair in $X'' \setminus \{x_i\}$ induces an edge that is anticomplete to the edge y_iy_j, we see that G contains an induced $2P_2$, a contradiction. We deduce that X' is a clique of size k. Now, since G is $(K_r \boxminus rP_1)$-free, there exist distinct $i, j \in \{1, 2, \ldots, k\}$ such that y_iy_j is an edge. Note that since $k \geq 6$, there exist distinct $s, t \in \{1, 2, \ldots, k\} \setminus \{i, j\}$. But now x_sx_t is anticomplete to y_iy_j, contradicting that G is $2P_2$-free. ◀

The class of $(K_r \boxminus rP_1, 2P_2)$-free graphs for $r \in \{1, 2\}$ is a subclass of P_4-free graphs, and thus has bounded clique-width and mim-width. However, for $r \geq 3$, the class of $(K_r \boxminus rP_1, 2P_2)$-free graphs has unbounded clique-width [23, Theorem 4.18], whereas Theorem 14 shows it has bounded mim-width. In particular, $(\mathsf{net}, 2P_2)$-free graphs and $(\mathsf{bull}, 2P_2)$-free graphs have bounded mim-width but unbounded clique-width.

In our next two results, we present two other new classes of bounded mim-width. We omit the proof of the second result.

▶ **Theorem 15.** *Let G be a $(K_r \boxminus P_1, tP_2)$-free graph for $r \geq 1$ and $t \geq 1$. Then* $\mathrm{cutmim}_G(X, \overline{X}) < R(r, R(r, t))$ *for every $X \subseteq V(G)$. In particular,* $\mathrm{mimw}(G) < R(r, R(r, t))$.

Proof. Let $k = R(r, R(r, t))$ and let (T, δ) be a branch decomposition of G. Towards a contradiction, suppose that there exists $X \subseteq V(G)$ such that $G[X, \overline{X}]$ has an induced matching of size at least k. Let $X' = \{x_1, x_2, \ldots, x_k\} \subseteq X$ and $Y' = \{y_1, y_2, \ldots, y_k\} \subseteq \overline{X}$ such that x_iy_i is an edge of the induced matching for each $i \in \{1, 2, \ldots, k\}$.

Since $|X'| = k = R(r, R(r, t))$, the set X' contains either a clique of size r, or an independent set of size $R(r, t)$. Suppose there is some $J \subseteq \{1, 2, \ldots, k\}$ such that $X_J = \{x_i : i \in J\}$ is a clique of size r. Then, for an arbitrarily chosen $j \in J$, the vertices $X_J \cup \{y_j\}$ induce a $K_r \boxminus P_1$, a contradiction. So X' contains an independent set of size $R(r, t)$. Let $I \subseteq \{1, 2, \ldots, k\}$ such that $X_I = \{x_i : i \in I\}$ is an independent set of size $R(r, t)$, and consider the set $Y_I = \{y_i : i \in I\}$. Since $|Y_I| = R(r, t)$, the set Y_I either contains a clique of size r, or an independent set of size t. In the former case, G contains an induced $K_r \boxminus P_1$, while in the latter case, G contains an induced tP_2, a contradiction. ◀

▶ **Theorem 16.** *Let G be a $(K_r \boxminus K_r, sP_1 + P_2)$-free graph for $r \geq 1$ and $s \geq 0$. Then* $\mathrm{cutmim}_G(X, \overline{X}) < R(R(r, s+1), s+1)$ *for every $X \subseteq V(G)$. In particular,* $\mathrm{mimw}(G) < R(R(r, s+1), s+1)$.

Note that $(K_r \boxminus P_1, tP_2)$-free graphs have unbounded clique-width if and only if $r \geq 3, t \geq 3$, or $r \geq 4$, $t \geq 2$ [23, Theorem 4.18]. Note also that $(K_r \boxminus K_r, sP_1 + P_2)$-free graphs have unbounded clique-width if and only if $r = 2$, $s \geq 3$, or $r \geq 3$, $s \geq 2$ [23, Theorem 4.18].

Our final results of the section resolve the remaining cases where $|V(H_1)| + |V(H_2)| \leq 8$ (we omit their proofs). For these results, we employ the following approach. Suppose we wish to show that the class of (H_1', H_2')-free graphs is bounded, where $H_1' \subseteq_i H_1$ for one of the pairs (H_1, H_2) appearing in Theorems 14 to 16. If G is a H_2-free graph in the class, then we can compute a branch decomposition of constant mim-width by one of Theorems 14

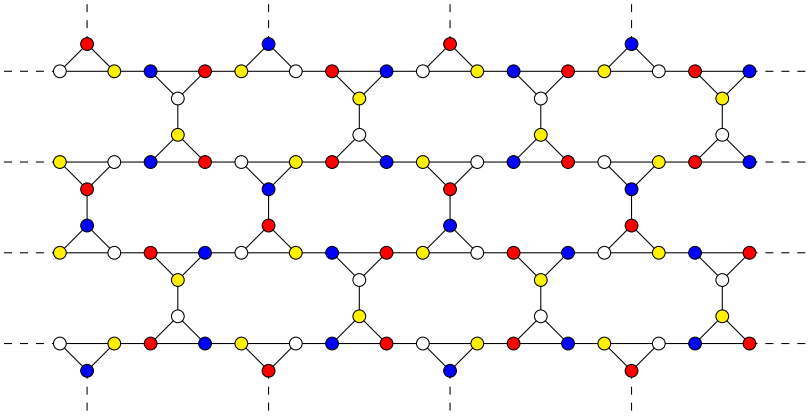


Figure 4 A particular 4-colouring of a net-wall, used in the proof of Theorem 21.

to 16. So it remains only to show that we can compute a branch decomposition of constant mim-width for (H_1', H_2')-free graphs having an induced subgraph isomorphic to H_2. For example, when $H_1' = 2P_2$ and $H_2' = K_{1,3}$, we exploit the structure of $(2P_2, K_{1,3})$-free graphs having an induced $K_3 \boxminus 3P_1$ to prove the next lemma, and, together with Theorem 14, obtain Theorem 18. Similarly, when $H_1' = 2P_1 + P_2$ and $H_2' = \mathsf{bowtie}$ (see Figure 3), we use Lemma 19 and Theorem 16 to obtain Theorem 20.

▶ **Lemma 17.** *Let G be a connected $(2P_2, K_{1,3})$-free graph. Given $X \subseteq V(G)$ such that $G[X] \cong K_r \boxminus rP_1$ for some $r \geq 3$, where X is maximal, we can construct, in $O(n)$ time, a branch decomposition (T, δ) of G such that $\mathrm{mimw}_G(T, \delta) = 1$.*

▶ **Theorem 18.** *Let G be a $(2P_2, K_{1,3})$-free graph. Then $\mathrm{mimw}(G) < 6$, and one can construct, in polynomial time, a branch decomposition (T, δ) of G with $\mathrm{mimw}_G(T, \delta) < 6$.*

▶ **Lemma 19.** *Let G be a $(2P_1 + P_2, \mathsf{bowtie})$-free graph. Given $X \subseteq V(G)$ such that $G[X] \cong K_r \boxminus K_r$ for some $r \geq 5$, where X is maximal, we can construct, in $O(n)$ time, a branch decomposition (T, δ) of G such that $\mathrm{mimw}_G(T, \delta) = 2$.*

▶ **Theorem 20.** *Let G be a $(2P_1 + P_2, \mathsf{bowtie})$-free graph. Then $\mathrm{mimw}(G) < R(14, 3)$, and one can construct, in polynomial time, a branch decomposition (T, δ) of G with $\mathrm{mimw}_G(T, \delta) < R(14, 3)$.*

5 New Unbounded Cases

We present a number of graph classes of unbounded mim-width, starting with following two theorems (we omit the proof of the second theorem).

▶ **Theorem 21.** *The class of $(\mathsf{diamond}, 5P_1)$-free graphs has unbounded mim-width.*

Proof. For every integer k, we will construct a $(\mathsf{diamond}, 5P_1)$-free graph G such that $\mathrm{mimw}(G) > k$. By Lemma 10, for any integer k there exists a net-wall W such that $\mathrm{mimw}(W) > 4k$. We partition the vertex set $V(W)$ into four colour classes (V_1, V_2, V_3, V_4) as illustrated in Figure 4. Observe that, for each $i \in \{1, 2, 3, 4\}$, the set V_i is independent, and no two distinct vertices $v, v' \in V_i$ have a common neighbour; that is, $N_W(v) \cap N_W(v') = \varnothing$.

Let G be the graph obtained from W by making each of V_1, V_2, V_3 and V_4 into a clique. By Lemma 4, $\mathrm{mimw}(G) \geq \mathrm{mimw}(W)/4 > k$. Since any set of five vertices of G contains at least two vertices in one of V_1, V_2, V_3, and V_4, and each of these four sets is a clique, G is $5P_1$-free.

It remains to show that G is diamond-free. First, observe that if $G[X] \cong K_3$ for some $X \subseteq V(G)$ with $|X \cap V_i| \geq 2$ for some $i \in \{1,2,3,4\}$, then, since no two vertices in V_i have a common neighbour in W, it follows that $X \subseteq V_i$. Now, towards a contradiction, suppose $G[Y] \cong$ diamond for some $Y \subseteq V(G)$. Then Y is the union of two sets X' and X'' that induce triangles in G, and $|X' \cap X''| = 2$. Since W is diamond-free, we may assume that $W[X']$ is not a triangle. Then X' contains at least two vertices of V_i for some $i \in \{1,2,3,4\}$. By the earlier observation, $X' \subseteq V_i$. Since $|X' \cap X''| = 2$, we then have $|X'' \cap V_i| \geq 2$, so $X'' \subseteq V_i$, and hence $Y \subseteq V_i$. But this implies that Y is a clique in G; a contradiction. So G is diamond-free. ◀

▶ **Theorem 22.** *The class of* $(4P_1, \overline{3P_1 + P_2}, \overline{P_1 + 2P_2})$*-free graphs has unbounded mim-width.*

Next we use the construction of a chordal bipartite graph G' from a graph G, given in [11][2]. Let $G = (V, E)$ be a graph. We take two copies of V labelled as follows: $X = \{x_v : v \in V\}$ and $Y = \{y_v : v \in V\}$. To construct G', start with a complete bipartite graph with vertex bipartition (X, Y), and add, for each edge $e \in E$ with endpoints u and v, two paths: an $x_u y_v$-path $x_u q_e t_e y_v$, and an $x_v y_u$-path $x_v q'_e t'_e y_u$. For convenience, we let $Q = \bigcup_{e \in E(G)} \{q_e, q'_e\}$ and $T = \bigcup_{e \in E(G)} \{t_e, t'_e\}$. Observe that (X, Y, Q, T) partitions $V(G')$; see also Figure 5.

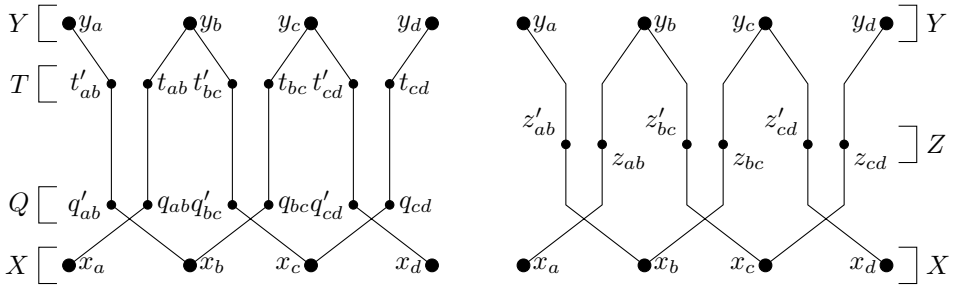


Figure 5 The graphs G' and G'', excluding edges between X and Y.

We need two lemmas. The first one is due to Baron, Capelli and Mengel. We omit the proof of the second one.

▶ **Lemma 23** ([11, Lemmas 15 and 16])**.** *For any graph G, the graph G' is chordal bipartite, Moreover, if G is bipartite, then* $\mathrm{mimw}(G') \geq \mathrm{tw}(G)/6$*, where* $\mathrm{tw}(G)$ *denotes the treewidth of G.*

▶ **Lemma 24.** *For any graph G, the chordal bipartite graph G' is $(P_8, P_3 + P_6, S_{1,1,5})$-free.*

Lemma 24 is tight in the following sense: for some graph G, the graph G' can contain, as an induced subgraph, $tP_2 + P_7$ or tP_5 for any non-negative integer t, or $S_{2,2,4}$.

Theorem 25 now follows from Lemmas 23 and 24 and the fact that bipartite graphs can have arbitrarily large treewidth (see, e.g., [47]). We use Lemma 4 to obtain Theorems 26 and 27 (proofs omitted).

▶ **Theorem 25.** *The class of chordal bipartite $(P_8, P_3 + P_6, S_{1,1,5})$-free graphs has unbounded mim-width.*

[2] Alternatively, we could take a wall, which has bipartition classes A and B; 2-subdivide all of its edges; and make A complete to B. The resulting graph has the same structure as G' and can have arbitrarily large mim-width due to Theorem 8 and Lemmas 2 and 4.

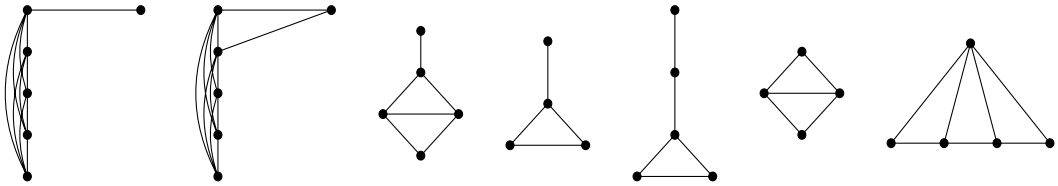


Figure 6 The graphs $K_5 \boxminus P_1 = \overline{K_{1,4} + P_1}$, $\overline{K_{1,3} + 2P_1}$, $\overline{S_{1,1,2}}$, paw, hammer, diamond and gem.

▶ **Theorem 26.** *The class of* $(4P_1, \mathsf{gem}, \overline{P_1 + 2P_2})$*-free graphs has unbounded mim-width.*

▶ **Theorem 27.** *The class of* $(\mathsf{diamond}, 2P_3)$*-free graphs has unbounded mim-width.*

We now describe the construction of a graph G'' from a graph $G = (V, E)$. This construction is similar to the construction of G'; we adapt the approach taken by [11] to construct graphs with arbitrarily large mim-width. Take two copies of V labelled as follows: $X = \{x_v : v \in V\}$ and $Y = \{y_v : v \in V\}$. Construct a graph G'' on vertex set $X \cup Y \cup Z$ where $Z = \bigcup_{e \in E(G)} \{z_e, z'_e\}$. Start with a complete bipartite graph with vertex bipartition (X, Y), and add, for each edge $e \in E$ with endpoints u and v, two paths $x_u z_e y_v$ and $x_v z'_e y_u$. Observe that G'' is 3-partite, with colour classes (X, Y, Z); see also Figure 5.

The following lemma is proven by modifying the proof of Lemma 23 given in [11]. Alternatively, we could take the $n \times n$ wall W, which has bipartition classes A and B; 1-subdivide each edge of W; and make A complete to B. By applying Theorem 8 and Lemmas 2 and 4, we obtain a lower bound on the mim-width in terms of n.

▶ **Lemma 28.** *If G is a bipartite graph, then* $\mathrm{mimw}(G'') \geq \mathrm{tw}(G)/6$.

We use Lemma 28 to show the following theorem (proof omitted).

▶ **Theorem 29.** *The class of* $(K_4, \mathsf{diamond}, P_6, P_2 + P_4)$*-free graphs has unbounded mim-width.*

6 State of the Art

We show the consequences of the results from Sections 3–5 for the boundedness and unboundedness of mim-width of classes of (H_1, H_2)-free graphs. We will also make a comparison between the results for mim-width and clique-width. In contrast to the situation where only one induced subgraph is forbidden, we note many differences when two induced subgraphs H_1 and H_2 are forbidden. Figure 6 illustrates a number of graphs used in the section.

Our first summary theorem follows from combining our results with known results from [5, 6, 7, 8, 9, 10, 12, 21, 22, 24, 25, 26, 37, 42] (proof details omitted). It gives all pairs (H_1, H_2) for which the mim-width of the class of (H_1, H_2)-free graphs is bounded. This theorem gives more bounded cases than the corresponding summary theorem for boundedness of *clique-width* of classes of (H_1, H_2)-free graphs, which can be found in [23] and which we need for our proof. To get the summary theorem for clique-width, replace Cases (x)–(xv) of Theorem 30 by the more restricted case where $H_1 = K_s$ and $H_2 = tP_1$ for some $s, t \geq 1$.

▶ **Theorem 30.** *For graphs H_1 and H_2, the mim-width of the class of (H_1, H_2)-free graphs is* bounded *and* quickly computable *if one of the following holds:*

- **(i)** H_1 *or* $H_2 \subseteq_i P_4$,
- **(ii)** $H_1 \subseteq_i \mathsf{paw}$ *and* $H_2 \subseteq_i K_{1,3} + 3P_1$, $K_{1,3} + P_2$, $P_1 + P_2 + P_3$, $P_1 + P_5$, $P_1 + S_{1,1,2}$, $P_2 + P_4$, P_6, $S_{1,1,3}$ *or* $S_{1,2,2}$,

(iii) $H_1 \subseteq_i P_1+P_3$ *and* $H_2 \subseteq_i \overline{K_{1,3}+3P_1}$, $\overline{K_{1,3}+P_2}$, $\overline{P_1+P_2+P_3}$, $\overline{P_1+P_5}$, $\overline{P_1+S_{1,1,2}}$, $\overline{P_2+P_4}$, $\overline{P_6}$, $\overline{S_{1,1,3}}$ *or* $\overline{S_{1,2,2}}$,
(iv) $H_1 \subseteq_i$ diamond *and* $H_2 \subseteq_i P_1+2P_2$, $3P_1+P_2$ *or* P_2+P_3,
(v) $H_1 \subseteq_i 2P_1+P_2$ *and* $H_2 \subseteq_i \overline{P_1+2P_2}$, $\overline{3P_1+P_2}$ *or* $\overline{P_2+P_3}$,
(vi) $H_1 \subseteq_i$ gem *and* $H_2 \subseteq_i P_1+P_4$ *or* P_5,
(vii) $H_1 \subseteq_i P_1+P_4$ *and* $H_2 \subseteq_i \overline{P_5}$,
(viii) $H_1 \subseteq_i K_3+P_1$ *and* $H_2 \subseteq_i K_{1,3}$,
(ix) $H_1 \subseteq_i 2P_1+P_3$ *and* $H_2 \subseteq_i \overline{2P_1+P_3}$,
(x) $H_1 \subseteq_i 2P_1+P_2$ *and* $H_2 \subseteq_i$ bowtie,
(xi) $H_1 \subseteq_i K_{1,3}$ *and* $H_2 \subseteq_i 2P_2$,
(xii) $H_1 \subseteq_i K_r$ *for* $r \geq 1$ *and* $H_2 \subseteq_i sP_1+P_5$ *for* $s \geq 0$,
(xiii) $H_1 \subseteq_i K_r \boxminus rP_1$ *for* $r \geq 1$ *and* $H_2 \subseteq_i 2P_2$,
(xiv) $H_1 \subseteq_i K_r \boxminus P_1$ *for* $r \geq 1$ *and* $H_2 \subseteq_i tP_2$ *for* $t \geq 1$, *or*
(xv) $H_1 \subseteq_i K_r \boxminus K_r$ *for* $r \geq 1$ *and* $H_2 \subseteq_i sP_1+P_2$ *for* $s \geq 0$.

Our second summary theorem, on the unbounded cases of mim-width, follows from combining results in previous sections (proof details omitted). We let $\mathcal{S}$ be the class of graphs every connected component of which is either a subdivided claw or a path. We let $\mathcal{N}$ denote the class of graphs that contain a connected component with either a cycle of length at least 4 or at least two (not necessarily vertex-disjoint) triangles; note, for example, that $\mathcal{N}$ contains C_4, diamond, and K_4.

▶ **Theorem 31.** *For graphs* H_1 *and* H_2, *the class of* (H_1, H_2)*-free graphs has* unbounded *mim-width if one of the following holds:*

(i) $H_1 \notin \mathcal{S}$ *and* $H_2 \notin \mathcal{S}$,
(ii) $H_1 \supseteq_i C_3$ *and* $H_2 \supseteq_i P_3+P_6$, P_8 *or* $S_{1,1,5}$,
(iii) $H_1 \supseteq_i K_{1,3}$ *and* $H_2 \in \mathcal{N}$,
(iv) $H_1 \supseteq_i$ diamond *and* $H_2 \supseteq_i 5P_1$, P_2+P_4, $2P_3$ *or* P_6,
(v) $H_1 \supseteq_i 3P_1$ *and* $H_2 \supseteq_i 3P_1$, C_5 *or* $\overline{C_{2s+1}}$ *for* $s \geq 3$,
(vi) $H_1 \supseteq_i 4P_1$ *and* $H_2 \supseteq_i$ gem, $\overline{3P_1+P_2}$ *or* $\overline{P_1+2P_2}$,
(vii) $H_1 \supseteq_i 2P_2$ *and* $H_2 \supseteq_i C_4$, C_5, $K_{1,4}$, $2P_2$, $\overline{3P_1+P_2}$ *or* sun_t *for* $t \geq 3$, *or*
(viii) $H_1 \supseteq_i K_4$ *and* $H_2 \supseteq_i P_2+P_4$ *or* P_6.

We note that the situation for the unbounded cases is again different from the situation for the unbounded cases of clique-width. For example, (H_1, H_2)-free graphs have unbounded clique-width if both $\overline{H_1} \notin \mathcal{S}$ and $\overline{H_2} \notin \mathcal{S}$ (see, for example, [26]). Take, for instance, $H_1 = 4P_1$ and $H_2 = 2P_2$. Then $\overline{H_1} = K_4$ and $\overline{H_2} = C_4$, and thus $\overline{H_1} \notin \mathcal{S}$ and $\overline{H_2} \notin \mathcal{S}$, so (H_1, H_2)-free graphs have unbounded clique-width. However, by Theorem 30-(xiii), (H_1, H_2)-free graphs have bounded mim-width. As $(\overline{H_1}, \overline{H_2})$-free graphs have unbounded mim-width by Theorem 31-(i), this example also shows that the complementation operation, a standard tool for working with clique-width, cannot be used for mim-width. Consequently, for mim-width there are many more open cases than the only five open cases for clique-width [23]. In order to get a handle on the open cases for mim-width, we now present some consequences of Theorems 30 and 31.

We first note that Theorems 30 and 31 cover all pairs (H_1, H_2) with $|V(H_1)|+|V(H_2)| \leq 8$.

▶ **Corollary 32.** *Let* H_1 *and* H_2 *be graphs with* $|V(H_1)| + |V(H_2)| \leq 8$. *Then the pair* (H_1, H_2) *satisfies Theorem 30 or Theorem 31.*

We now present two open problems involving some particular open cases arising from Theorems 30 and 31.

▶ **Open Problem 1.** *Let H_1 and H_2 be* forests. *Then (un)boundedness of mim-width of (H_1, H_2)-free graphs is open if and only if*

1. $H_1 = 2P_2$ *and* $H_2 = K_{1,3} + sP_1$ *for* $s \geq 1$, *or*
2. $H_1 = 2P_2$ *and* $H_2 = S_{1,1,2} + sP_1$ *for* $s \geq 0$.

▶ **Open Problem 2.** *Let H_1 and H_2 be* connected *graphs. Then (un)boundedness of mim-width of (H_1, H_2)-free graphs is open if and only if*

1. $H_1 = P_5$ *and* $H_2 = \overline{S_{1,1,2}}$ *or* $\overline{K_{1,r} + sP_1}$ *for* $r \geq 3$ *and* $s \in \{1, 2\}$,
2. $H_1 = P_7$ *or* $S_{h,i,j}$ *for* $h \leq i \leq j \leq 4$ *with* $i + j \leq 6 \leq h + i + j$ *and* $H_2 = C_3$ *or* paw, *or*
3. $H_1 = K_{1,3}$ *or* $S_{1,1,2}$ *and* $H_2 =$ hammer.

7 Conclusion

We extended the toolkit for proving (un)boundedness of mim-width of hereditary graph classes. Using the extended toolkit, we found new classes of (H_1, H_2)-free graphs of bounded and unbounded mim-width. We showed that the situation for mim-width of hereditary graph classes is different from the situation for clique-width, even when only two induced subgraphs H_1 and H_2 are forbidden. For future work, Open Problems 1 and 2 deserve attention. In particular, the class of $(P_5, \overline{K_{1,r} + sP_1})$-free graphs, for $r \geq 3$ and $s \in \{1, 2\}$ (Case 1 of Open Problem 2), is the only remaining infinite family. Moreover, a similar approach to Theorem 18 might be conducive to resolving further open cases where $H_1 = 2P_2$.

Another interesting case is when $H_1 = K_r$ for some r. For $r \geq 4$, Theorems 30 and 31 imply that the mim-width of the class of (K_r, H_2)-free graphs is bounded and quickly computable when $H_2 \subseteq_i sP_1 + P_5$ or tP_2, and unbounded when $H_2 \supseteq_i K_{1,3}$, $P_2 + P_4$, or P_6, or $H_2 \notin \mathcal{S}$. It can be shown that all remaining cases belong to one infinite family: when $H_2 = tP_2 + uP_3$ for $u \geq 1$ and $t + u \geq 2$. For any H_2 such that mim-width is bounded and quickly computable for the class of (K_r, H_2)-free graphs, k-Colouring is polynomial-time solvable for all $k < r$ (for example, see [12] when $H_2 \subseteq_i sP_1 + P_5$). For problems having polynomial-time algorithms when mim-width is bounded and quickly computable, we obtain $n^{f(\omega(G))}$-time algorithms, for some function f, when restricted to H_2-free graphs; that is, XP algorithms parameterized by $\omega(G)$ (the size of the largest clique in G). Recently, Chudnovsky et al. [17] showed that for P_5-free graphs, there exists an $n^{O(\omega(G))}$-time algorithm for Max Partial H-Colouring, a problem generalizing Maximum Independent Set and Odd Cycle Transversal, and which is polynomial-time solvable when mim-width is bounded and quickly computable.

One could also consider the class of (rP_1, H_2)-free graphs, for an integer r and a graph H_2, and similarly obtain, for many problems, XP algorithms parameterized by $\alpha(G)$ for the class of H_2-free graphs, where $\alpha(G)$ is the size of the largest independent set in G. For $r \geq 5$, Theorems 30 and 31 imply that the mim-width of the class of (rP_1, H_2)-free graphs is bounded and quickly computable when $H_2 \subseteq_i K_t \boxminus K_t$ for some t, and unbounded when H_2 is not co-bipartite, or $H_2 \supseteq_i$ diamond. All unresolved cases belong to the infinite family $H_2 = \overline{K_{s,t} + P_1}$ for $s, t \geq 2$ (note that if $s = t = 2$, then $H_2 =$ bowtie).

References

1 Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoretical Computer Science*, 511:54–65, 2013.

2 Benjamin Bergougnoux and Mamadou Moustapha Kanté. More applications of the d-neighbor equivalence: Connectivity and acyclicity constraints. *Proc. ESA 2019, LIPIcs*, 144:17:1–17:14, 2019.

3 Benjamin Bergougnoux, Charis Papadopoulos, and Jan Arne Telle. Node multiway cut and subset feedback vertex set on graphs of bounded mim-width. *Proc. WG 2020, LNCS*, to appear, 2020.

4 Therese C. Biedl, Erik D. Demaine, Christian A. Duncan, Rudolf Fleischer, and Stephen G. Kobourov. Tight bounds on maximal and maximum matchings. *Discrete Mathematics*, 285:7–15, 2004.

5 Alexandre Blanché, Konrad K. Dabrowski, Matthew Johnson, Vadim V. Lozin, Daniël Paulusma, and Viktor Zamaraev. Clique-width for graph classes closed under complementation. *SIAM Journal on Discrete Mathematics*, 34:1107–1147, 2020.

6 Rodica Boliac and Vadim V. Lozin. On the clique-width of graphs in hereditary classes. *Proc. ISAAC 2002, LNCS*, 2518:44–54, 2002.

7 Andreas Brandstädt, Tilo Klembt, and Suhail Mahfud. P_6-and triangle-free graphs revisited: structure and bounded clique-width. *Discrete Mathematics & Theoretical Computer Science*, 8:173–188, 2006.

8 Andreas Brandstädt, Hoàng-Oanh Le, and Raffaele Mosca. Gem-and co-gem-free graphs have bounded clique-width. *International Journal of Foundations of Computer Science*, 15:163–185, 2004.

9 Andreas Brandstädt, Hoàng-Oanh Le, and Raffaele Mosca. Chordal co-gem-free and (P_5, gem)-free graphs have bounded clique-width. *Discrete Applied Mathematics*, 145:232–241, 2005.

10 Andreas Brandstädt and Suhail Mahfud. Maximum weight stable set on graphs without claw and co-claw (and similar graph classes) can be solved in linear time. *Information Processing Letters*, 84:251–259, 2002.

11 Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding model counting for beta-acyclic CNF-formulas. *Proc. STACS 2015, LIPIcs*, 30:143–156, 2015.

12 Nick Brettell, Jake Horsfield, and Daniël Paulusma. Colouring $(sP_1 + P_5)$-free graphs: a mim-width perspective. *CoRR*, abs/2004.05022, 2020. `arXiv:2004.05022`.

13 Nick Brettell, Andrea Munaro, and Daniël Paulusma. List k-Colouring P_t-free graphs with no induced 1-subdivision of $K_{1,s}$: a mim-width perspective. *CoRR*, abs/2008.01590, 2020. `arXiv:2008.01590`.

14 Nick Brettell, Andrea Munaro, and Daniël Paulusma. Solving problems on generalized convex graphs via mim-width. *CoRR*, abs/2008.09004, 2020. `arXiv:2008.09004`.

15 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-width of graphs. *Theoretical Computer Science*, 412:5187–5204, 2011.

16 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoretical Computer Science*, 511:66–76, 2013.

17 Maria Chudnovsky, Jason King, Michal Pilipczuk, Pawel Rzazewski, and Sophie Spirkl. Finding large H-colorable subgraphs in hereditary graph classes. *Proc. ESA 2020, LIPIcs*, 173:35:1–35:17, 2020.

18 Maria Chudnovsky, Sophie Spirkl, and Mingxian Zhong. List-3-coloring P_t-free graphs with no induced 1-subdivision of $K_{1,s}$. *Discrete Mathematics*, 343:112086, 2020.

19 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101:77–114, 2000.

20 Jean-François Couturier, Petr A. Golovach, Dieter Kratsch, and Daniël Paulusma. List coloring in the absence of a linear forest. *Algorithmica*, 71:21–35, 2015.

21 Konrad K. Dabrowski, François Dross, and Daniël Paulusma. Colouring diamond-free graphs. *Journal of Computer and System Sciences*, 89:410–431, 2017.

22 Konrad K. Dabrowski, Shenwei Huang, and Daniël Paulusma. Bounding clique-width via perfect graphs. *Journal of Computer and System Sciences*, 104:202–215, 2019.

23 Konrad K. Dabrowski, Matthew Johnson, and Daniël Paulusma. Clique-width for hereditary graph classes. *London Mathematical Society Lecture Note Series*, 456:1–56, 2019.

24 Konrad K. Dabrowski, Vadim V. Lozin, and Daniël Paulusma. Clique-width and well-quasi-ordering of triangle-free graph classes. *Joournal of Computer and System Sciences*, 108:64–91, 2020.

25 Konrad K. Dabrowski, Vadim V. Lozin, Rajiv Raman, and Bernard Ries. Colouring vertices of triangle-free graphs without forests. *Discrete Mathematics*, 312:1372–1385, 2012.

26 Konrad K. Dabrowski and Daniël Paulusma. Clique-width of graph classes defined by two forbidden induced subgraphs. *The Computer Journal*, 59:650–666, 2016.

27 Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on H-graphs. *Proc. ESA 2018, LIPIcs*, 30:1–14, 2018.

28 Esther Galby and Andrea Munaro. Approximating Independent Set and Dominating Set on VPG graphs. *CoRR*, abs/2004.07566, 2020. `arXiv:2004.07566`.

29 Esther Galby, Andrea Munaro, and Bernard Ries. Semitotal domination: New hardness results and a polynomial-time algorithm for graphs of bounded mim-width. *Theoretical Computer Science*, 814:28–48, 2020.

30 Georg Gottlob, Petr Hliněný, Sang-il Oum, and Detlef Seese. Width parameters beyond tree-width and their applications. *The Computer Journal*, 51:326–362, 2008.

31 Frank Gurski. The behavior of clique-width under graph operations and graph transformations. *Theory of Computing Systems*, 60:346–376, 2017.

32 Venkatesan Guruswami and C. Pandu Rangan. Algorithmic aspects of clique-transversal and clique-independent sets. *Discrete Applied Mathematics*, 100:183–202, 2000.

33 Lars Jaffke, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. Mim-width III. Graph powers and generalized distance domination problems. *Theoretical Computer Science*, 796:216–236, 2019.

34 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width I. Induced path problems. *Discrete Applied Mathematics*, 278:153–168, 2020.

35 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width II. The Feedback Vertex Set problem. *Algorithmica*, 82:118–145, 2020.

36 Öjvind Johansson. Clique-decomposition, NLC-decomposition, and modular decomposition - relationships and results for random graphs. *Congressus Numerantium*, 132:39–60, 1998.

37 Marcin Kamiński, Vadim V. Lozin, and Martin Milanič. Recent developments on graphs of bounded clique-width. *Discrete Applied Mathematics*, 157:2747–2761, 2009.

38 Dong Yeap Kang, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. A width parameter useful for chordal and co-comparability graphs. *Theoretical Computer Science*, 704:1–17, 2017.

39 Iyad Kanj, Michael J. Pelsmajer, Marcus Schaefer, and Ge Xia. On the induced matching problem. *Journal of Computer and System Sciences*, 77(6):1058–1070, 2011.

40 Vadim V. Lozin and Dieter Rautenbach. On the band-, tree-, and clique-width of graphs with bounded vertex degree. *SIAM Journal on Discrete Mathematics*, 18:195–206, 2004.

41 Stefan Mengel. Lower bounds on the mim-width of some graph classes. *Discrete Applied Mathematics*, 248:28–32, 2018.

42 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96:514–528, 2006.

43 Andrzej Proskurowski and Jan Arne Telle. Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal on Discrete Mathematics*, 10:529–550, 1997.

44 Michaël Rao. Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics*, 308:6157–6165, 2008.

45 Neil Robertson and Paul D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52:153–190, 1991.

46 Sigve Hortemo Sæther and Martin Vatshelle. Hardness of computing width parameters based on branch decompositions over the vertex set. *Theoretical Computer Science*, 615:120–125, 2016.

47 Martin Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, 2012.

Fixed-Parameter Algorithms for Longest Heapable Subsequence and Maximum Binary Tree

Karthekeyan Chandrasekaran
University of Illinois, Urbana-Champaign, IL, USA
karthe@illinois.edu

Elena Grigorescu
Purdue University, West Lafayette, IN, USA
elena-g@purdue.edu

Gabriel Istrate
West University of Timişoara, Romania
e-Austria Research Institute, Timişoara, Romania
gabrielistrate@acm.org

Shubhang Kulkarni[1]
University of Illinois, Urbana-Champaign, IL, USA
smkulka2@illinois.edu

Young-San Lin
Purdue University, West Lafayette, IN, USA
lin532@purdue.edu

Minshen Zhu
Purdue University, West Lafayette, IN, USA
zhu628@purdue.edu

Abstract

A heapable sequence is a sequence of numbers that can be arranged in a *min-heap data structure*. Finding a longest heapable subsequence of a given sequence was proposed by Byers, Heeringa, Mitzenmacher, and Zervas (ANALCO 2011) as a generalization of the well-studied longest increasing subsequence problem and its complexity still remains open. An equivalent formulation of the longest heapable subsequence problem is that of finding a maximum-sized binary tree in a given permutation directed acyclic graph (permutation DAG). In this work, we study parameterized algorithms for both longest heapable subsequence and maximum-sized binary tree. We introduce *alphabet size* as a new parameter in the study of computational problems in permutation DAGs and show that this parameter with respect to a fixed topological ordering admits a complete characterization and a polynomial time algorithm. We believe that this parameter is likely to be useful in the context of optimization problems defined over permutation DAGs.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

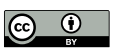
Keywords and phrases maximum binary tree, heapability, permutation directed acyclic graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.7

Related Version `www.cs.purdue.edu/homes/lin532/file/FPT_algorithms_for_MBT_IPEC.pdf`

Funding *Karthekeyan Chandrasekaran*: Supported by NSF CCF-1814613 and NSF CCF-1907937.
Elena Grigorescu: Supported by NSF CCF-1910659 and NSF CCF-1910411.
Gabriel Istrate: Supported by a grant of the Romanian Ministry of Research and Innovation, CNCS – UEFISCDI project number PN-III-P4-ID-PCE-2016-0842, within PNCDI III.
Young-San Lin: Supported by NSF CCF-1910411.
Minshen Zhu: Supported by NSF CCF-1910659.

[1] Work done while at Purdue University, USA.



1 Introduction

The longest increasing subsequence is a fundamental computational problem that has led to numerous discoveries in algorithms as well as combinatorics. The motivation behind this work is a generalization of the longest increasing subsequence problem, known as the *longest heapable subsequence* problem, introduced by Byers, Heeringa, Mitzenmacher, and Zervas [5]. We begin by defining this problem. A rooted tree whose nodes are labeled with values has the *heap property* if the value of every node is at least that of its parent; a sequence of natural numbers is *heapable* if the elements can be sequentially placed one at a time to form a binary tree with the heap property. For example, the sequence $1, 5, 3, 2, 4$ is not heapable while the sequence $1, 3, 3, 2, 4$ is heapable. Throughout this work, we will be interested in sequences whose elements are natural numbers. In the longest heapable subsequence problem, the goal is to find a longest heapable subsequence of a given sequence. Although the longest *increasing* subsequence problem is solvable in polynomial-time, the complexity of the longest *heapable* subsequence problem is still open.

The problem of verifying if a given sequence is heapable, although non-trivial, is solvable efficiently using a greedy approach [5]. In order to address the longest heapable subsequence problem, Porfilio [12] observed a connection to a graph problem on directed acyclic graphs (DAGs). The permutation DAG associated with a sequence $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n))$, denoted $\mathsf{PermDAG}(\sigma)$, is obtained by introducing a vertex t_i for every sequence element $i \in [n]$, and arcs (t_j, t_i) for every $i, j \in [n]$ such that $i < j$ and $\sigma(i) \leq \sigma(j)$. We recall that a directed graph G is a *permutation DAG* if there exists a sequence τ such that G is isomorphic to $\mathsf{PermDAG}(\tau)$. We need the notion of a binary tree in a given directed graph G: a subgraph T of G is an r-rooted binary tree if r is the unique vertex in T with no outgoing edges, every vertex in T has a unique directed path to r in T, and every vertex in T has in-degree at most 2 in T; the size of T is the number of vertices in T. Porfilio showed that a longest heapable subsequence of a given sequence σ is equivalent to a maximum-sized binary tree in $\mathsf{PermDAG}(\sigma)$. This result raises the question of whether one can efficiently find a maximum-sized binary tree in a given permutation DAG. The complexity of this problem also remains open.

In an earlier work [6], we showed that maximum-sized binary tree in arbitrary input directed graphs is fixed-parameter tractable when parameterized by the solution size: we gave a $2^k n^{O(1)}$ time algorithm, where k is the size of the largest binary tree and n is the number of vertices in the input graph. This also implies that the longest heapable subsequence problem is fixed-parameter tractable when parameterized by the solution size. In this work, we consider two alternative parameterizations for the maximum-sized binary tree/longest heapable subsequence problem.

Firstly, we show that the longest heapable subsequence problem is fixed-parameter tractable when parameterized by the number of distinct values in the input sequence. Next, we introduce *alphabet size* as a new parameter in the study of computational problems in permutation DAGs. Our algorithmic result for longest heapable subsequence problem implies that the maximum-sized binary tree problem in a given permutation DAG is fixed-parameter tractable when parameterized by the alphabet size. We currently do not know how to compute the alphabet size of a given permutation DAG. As a stepping stone towards computing alphabet size, we show that alphabet size *with respect to a fixed topological ordering* can be computed efficiently and it also admits a min-max relation and a polyhedral description. Our results suggest that alphabet size is an interesting parameterization for computational problems defined on permutation DAGs and merits a thorough study. Finally, we design a fixed-parameter algorithm for the maximum-sized binary tree problem in undirected graphs when parameterized by treewidth. We elaborate on our contributions now.

1.1 Results

Our first result shows that the longest heapable subsequence problem is fixed-parameter tractable when parameterized by the number of distinct values in the sequence.

▶ **Theorem 1.** *There exists an algorithm that takes as input an n-length sequence τ with k distinct values and returns a longest heapable subsequence of τ in time $(k+1)! \cdot k \cdot O(n)$. Equivalently, our algorithm returns a maximum-sized binary tree in* $\mathsf{PermDAG}(\tau)$.

We emphasize that our algorithm also works in the streaming model of computation – i.e., when the input sequence arrives one by one and the algorithm has to find the longest heapable subsequence of the input that has arrived so far with sublinear memory (in particular, the algorithm does not have the capability to store the entire input sequence seen so far). The space complexity of our algorithm is $(k+1)! \cdot k \cdot O(\log n)$ and is logarithmic for constant k.

Theorem 1 can also be viewed as a fixed-parameter algorithm to find a maximum-sized binary tree in a given permutation DAG when parameterized by *alphabet size*. We define this parameter now. We note that for a fixed permutation DAG G, there could be several sequences τ such that $\mathsf{PermDAG}(\tau)$ is isomorphic to G (e.g., see Figure 1). In fact, there could be sequences τ_1 and τ_2 such that the difference between the number of distinct symbols in τ_1 and τ_2 may be arbitrarily large – e.g., consider an n-vertex tournament DAG G in its unique topological ordering (all arcs oriented in the backward direction) which is isomorphic to $\mathsf{PermDAG}(\tau_1)$ as well as $\mathsf{PermDAG}(\tau_2)$ where $\tau_1 = (1, 2, \ldots, n)$ and $\tau_2 = (1, 1, \ldots, 1)$. This motivates our parameterization for permutation DAGs: The *alphabet size* of a n-vertex permutation DAG G, denoted $\alpha(G)$, is defined as follows (see Figure 1 for an example):

$$\alpha(G) := \min\{k : \exists \text{ sequence } \tau \in [k]^n \text{ with } \mathsf{PermDAG}(\tau) \text{ being isomorphic to } G\}.$$

We recall that directed graphs $G = (V, A)$ and $G' = (V,', A')$ are isomorphic if there exists a bijection $\phi : V' \to V$ such that $(u', v') \in A'$ if and only if $(\phi(u'), \phi(v')) \in A$. Theorem 1 also implies that there exists an algorithm that takes as input, an n-vertex permutation DAG G and a sequence τ with $\alpha(G) = k$ distinct values such that $\mathsf{PermDAG}(\tau)$ is isomorphic to G and returns a maximum-sized binary tree in G in time $(k+1)! \cdot k \cdot O(n)$, i.e., a fixed-parameter algorithm for maximum-sized binary tree in permutation DAGs when parameterized by alphabet size.

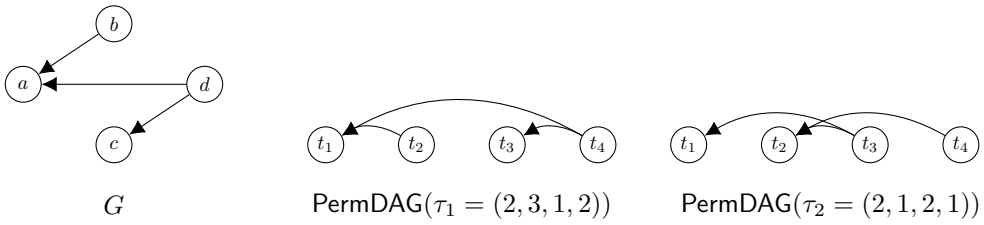


Figure 1 Let G be the input permuation DAG. The graph G is isomorphic to $\mathsf{PermDAG}(\tau_1)$ and $\mathsf{PermDAG}(\tau_2)$. The sequence $\tau_1 = (2, 1, 2, 1)$ uses only two distinct values, which turns out to be the minimum, so $\alpha(G) = 2$.

Next, we explore algorithmic aspects of our newly defined parameter, namely the alphabet size. A natural question is whether the alphabet size of a given permutation DAG can be computed in polynomial-time. Currently, we do not know the answer to this question. However, there is a natural related problem that seems like a stepping stone towards resolving the complexity of computing the alphabet size of permutation DAGs. We define this related problem now.

We recall that every DAG $G = (V, A)$ admits a topological ordering – a bijection $\gamma : V \to [n]$ corresponding to a permutation of its n vertices such that every arc $(v, u) \in A$ has $\gamma(u) < \gamma(v)$ (i.e., all edges are oriented in the backward direction with respect to the ordering defined by γ). For a fixed topological ordering $\gamma : V \to [n]$ of an n-vertex permutation DAG G, we define the *γ-alphabet size of G*, denoted $\alpha(G, \gamma)$, as follows (see Figure 2 for an example illustrating the definition):

$$\alpha(G,\gamma) := \min\Big\{k : \exists \text{ sequence } \tau \in [k]^n \text{ with } \mathsf{PermDAG}(\tau) = (\{t_1,\ldots,t_n\}, A') \text{ being isomorphic to } G \text{ under the mapping } \phi : \{t_1,\ldots,t_n\} \to V(G) \text{ given by } \phi(t_i) = \gamma^{-1}(i)\ \forall\ i \in [n]\Big\}.$$

We note that the optimization problem $\alpha(G, \gamma)$ may be infeasible in which case, we use $\alpha(G, \gamma) := \infty$ as the convention. The following relationship between alphabet size and γ-alphabet size is immediate for a permutation DAG G:

$$\alpha(G) = \min\{\alpha(G,\gamma) : \gamma \text{ is a topological ordering of } G\}.$$

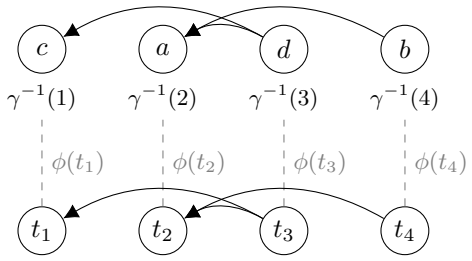


Figure 2 The graph at the top corresponds to G in topological order γ where $\gamma(c) = 1$, $\gamma(a) = 2$, $\gamma(d) = 3$, and $\gamma(b) = 4$. The graph at the bottom corresponds to $\mathsf{PermDAG}(\tau = (2, 1, 2, 1))$. Note that the two graphs are isomorphic under the mapping $\phi : \{t_1, t_2, t_3, t_4\} \to V(G)$ given by $\phi(t_1) = \gamma^{-1}(1) = c$, $\phi(t_2) = \gamma^{-1}(2) = a$, $\phi(t_3) = \gamma^{-1}(3) = d$, and $\phi(t_4) = \gamma^{-1}(4) = b$ (shown by dotted lines). We have that $\alpha(G, \gamma) = 2$ and is achieved by the sequence τ.

As a stepping stone towards understanding $\alpha(G)$, we show that $\alpha(G, \gamma)$ for a given topological ordering γ of G (i.e., the γ-alphabet size of G) can be computed in polynomial time.

▶ **Theorem 2.** *There exists a polynomial-time algorithm that takes a permutation DAG G and a topological ordering γ of G as input and detects if $\alpha(G, \gamma)$ is finite and if so, then returns a sequence that achieves $\alpha(G, \gamma)$.*

Our algorithm underlying Theorem 2 also reveals a min-max relation for γ-alphabet size that we describe now. Let $G = (V, A)$ be a permutation DAG with n vertices and let $\gamma : V \to [n]$ be a topological ordering of G such that $\alpha(G, \gamma)$ is finite. Let $\overrightarrow{E} := \{(u, v) : \gamma(u) < \gamma(v) \text{ and } (v, u) \notin A\}$ and $H(G, \gamma) := (V, A \cup \overrightarrow{E})$. We note that $H(G, \gamma)$ is a tournament.[2] Also, let $w : A \cup \overrightarrow{E} \to \{0, 1\}$ be an arc weight function for $H(G, \gamma)$ defined as follows:

$$w(e) := \begin{cases} 0 & \text{if } e \in A, \\ 1 & \text{if } e \in \overrightarrow{E}. \end{cases}$$

[2] A tournament is a directed graph $H = (V, A)$ in which we have exactly one of the two arcs (v, u) and (u, v) for every pair of distinct vertices $u, v \in V$.

Then, we have the following min-max relation for the minimization problem corresponding to $\alpha(G, \gamma)$.

▶ **Theorem 3.** *Let $G = (V, A)$ be a permutation DAG and γ be a topological ordering of V such that $\alpha(G, \gamma)$ is finite. Then,*

$$\alpha(G, \gamma) = 1 + \max \left\{ \sum_{e \in P} w(e) : P \text{ is a path in } H(G, \gamma) \right\}.$$

In addition to the algorithm and the min-max relation, we give a polyhedral description (see Theorem 19 in Section 3.3) that also leads to an LP-based algorithm to compute $\alpha(G, \gamma)$. We believe that alphabet size, as a parameter, is likely to be useful in the context of permutation DAGs and consequently, merits a thorough study. We view Theorems 2 and 3 as stepping stones towards the problem of efficiently computing the alphabet size of a given permutation DAG and Theorem 1 to be an application of this parameter. Resolving the complexity of computing the alphabet size is an intriguing open problem.

Next, we address the maximum binary tree problem in undirected graphs with bounded treewidth. Here, we are given an undirected graph G and the goal is to find a subgraph that is a binary tree with maximum number of nodes. An undirected graph is said to be a *binary tree* if the graph is acyclic and every vertex has degree at most 3. We observe that the existence of a binary tree can be expressed as a monadic second order logic property and hence, extensions of Courcelle's theorem [7] can be used to obtain an algorithm for maximum-sized binary tree that runs in time $f(w)n$ for some function $f(w)$, where n is the number of vertices and w is the treewidth of the input graph. However, the run-time dependence $f(w)$ is at least doubly exponential on the treewidth w in this approach. We improve this dependence substantially.

▶ **Theorem 4.** *Given a tree decomposition of an n-vertex undirected graph G with treewidth w, there exists an algorithm to find a maximum-sized binary tree in G in time $w^{O(w)}n$.*

1.2 Related Work

Heapability of integer sequences was introduced in [5] and has been investigated further in [9, 12, 10, 2, 3, 4, 1]. Heapability of integer sequences can be decided by a simple greedy algorithm [5] (see also [10] for an alternate approach based on integer programming, and [1] for connections with Dilworth's theorem and an algorithm based on network flows). Besides introducing the longest heapable subsequence problem, [5] also showed that deciding if a sequence can be arranged in a *complete* binary heap is NP-complete.

Heapable sequences of integers can be regarded as "loosely increasing". The celebrated *Ulam-Hammesley problem* aims to understand the length of the longest increasing sequence of a random permutation. This has a long history with deep connections to many areas of science (e.g., see [13]). [5] studied the counterpart of this problem for heapability: they showed that the longest heapable subsequence of a random permutatoin of length n is of size $n - o(n)$ with high probability and it can also be found in an online fashion.

As mentioned earlier, Porfilio [12] showed that the longest heapable subsequence is equivalent to solving the maximum-sized binary tree problem in permutation DAGs. In an earlier work [6], we showed that the maximum-sized binary tree problem is NP-hard in DAGs and showed further inapproximability results. We also gave a fixed-parameter algorithm for the maximum binary tree problem when parameterized by the solution size. Furthermore, we

designed a polynomial-time algorithm to solve the maximum-sized binary tree problem in the special class of bipartite permutation graphs. It is also known that maximum-sized binary tree problem in DAGs induced by sets of intervals can be solved in polynomial time [1].

Organization. In Section 2, we present the fixed-parameter algorithm for longest heapable subsequence when parameterized by the alphabet size and prove Theorem 1. In Section 3, we address the problem of computing γ-alphabet size and present a min-max relation. We present a polyhedral description for γ-alphabet size that leads to an LP-based algorithm for $\alpha(G, \gamma)$ in Section 3.3. Due to page limits, we present our fixed-parameter algorithm for computing a maximum-sized binary tree in bounded treewidth graphs in the full version.

2 Longest heapable subsequence parameterized by alphabet size

In this section, we prove Theorem 1 by giving a $(k+1)! \cdot k \cdot O(n)$-time algorithm to compute the longest heapable subsequence of a given n-length sequence containing k distinct values. We begin with certain useful definitions. Given a rooted non-empty binary tree T, we define the extended binary tree $Ext(T)$ by introducing new leaf nodes in a way that makes every node in T have exactly 2 children. The nodes in T are also referred to as *internal nodes*, and the new leaf nodes are referred to as *external nodes* (see Figure 3). We will denote a directed binary tree where each node is labeled by some number in $[k] \coloneqq \{1, 2, \ldots, k\}$ such that the labels on every leaf to root path is non-increasing as a *heap over alphabet* $[k]$.

▶ **Definition 5** (Shape). *Given a heap H over alphabet $[k]$, we define its* shape *as a tuple $\mathbf{x} = (x_0, x_1, \ldots, x_{k-1}, x_k)$, where x_i is the number of external nodes whose parents have label i in $Ext(H)$. We also follow the convention that the shape of an empty heap is $(1, 0, \ldots, 0)$.*

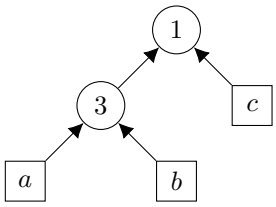


Figure 3 Given a binary tree T composed of two nodes 1 and 3, the extended binary tree $Ext(T)$ has two internal nodes 1 and 3, and three new external nodes a, b, and c. Suppose $k = 4$. The shape of the heap above is $\mathbf{x} = (x_0, x_1, x_2, x_3, x_4) = (0, 1, 0, 2, 0)$ because the parent of a and b has label 3 and the parent of c has label 1.

Intuitively, an external node represents the location of a potential future insertion into the heap. Since an insertion is effectively replacing an external node with a new internal node (thus introducing two new external nodes), it is captured by simple manipulations of shapes. This naturally leads us to defining insertions with respect to shapes. Given a shape $\mathbf{x} = (x_0, \ldots, x_k)$ and labels $a \leq b$, the shape obtained by inserting b under a, denoted $\mathbf{x}(a \leftarrow b)$, is defined as

$$\mathbf{x}\,(a \leftarrow b) := \begin{cases} (x_0, \ldots, x_{a-1}, x_a + 1, x_{a+1}, \ldots, x_k) & \text{if } x_a > 0 \text{ and } a = b, \\ (x_0, \ldots, x_{a-1}, x_a - 1, x_{a+1}, \ldots, x_{b-1}, x_b + 2, x_{b+1}, \ldots, x_k) & \text{if } x_a > 0 \text{ and } a < b, \\ \bot & \text{if } x_a = 0. \end{cases}$$

For example, consider the shape $\mathbf{x} = (0, 1, 0, 2, 0)$. The shape $\mathbf{x}\,(1 \leftarrow 2)$ is $(0, 0, 2, 2, 0)$, and the shape $\mathbf{x}(2 \leftarrow 3)$ is $\bot$. Given a heap H and a sequence $a = (a_1, \ldots, a_n)$, consider a longest subsequence of a which can be sequentially inserted to H as leaf nodes while maintaining

the heap property. We will call such a subsequence a *longest heapable subsequence starting from H*. We observe that the longest heapable subsequence of a given sequence starting from H depends only the shape of H and not the precise structure of H (i.e., the optimum does not change for two different heaps H_1 and H_2 sharing the same shape). Therefore, it is equivalent and also convenient to consider the longest heapable subsequence problem starting from an initial shape instead of an initial heap. This line of thought also suggests a natural dynamic programming approach where the subproblems are specified by shapes.

To analyze the running time, we need to upper bound the number of subproblems, which is the same as the number of distinct shapes. As a starting point, the number of distinct shapes can be upper bounded by $n^{O(k)}$. This is because in any n-node heap H there are exactly $n+1$ external nodes in $Ext(H)$ (an elementary property of binary trees). Therefore, the number of shapes is bounded by the number of non-negative integral solutions to $x_0 + x_1 + \ldots + x_k = n+1$, which is $n^{O(k)}$. Although this estimate seems like a very crude upper bound, bringing down the estimate into the fixed-parameter regime (i.e., $f(k)n^{O(1)}$) seems very difficult. We employ additional ideas to design a fixed-parameter algorithm.

Consider the longest heapable subsequence problem starting from initial shape $\mathbf{x} = (x_0, x_1, \ldots, x_k)$. Suppose that the initial shape also satisfies the condition that $x_j \geq k - j + 1$ for some $j \in [k]$. Our key observation is that all elements with labels at least j are heapable from $\mathbf{x}$: we can reserve an external node attached to j for each label $v \in \{j, j+1, \ldots, k\}$, which can then be used to form a chain of elements with the same label v. Essentially, once we have reached the shape $\mathbf{x}$, there are "infinitely" many external nodes available for future elements with label at least j, and hence, we no longer need to keep track of the precise values of $x_j, x_{j+1}, \ldots, x_k$. This motivates the following notion of refined shapes.

▶ **Definition 6** (Refined shapes). *A tuple $(x_0, x_1, \ldots, x_k)$ is a* refined shape *(over alphabet size k) if for each $j \in \{0, 1, \ldots, k\}$ we have $x_j \in \{0, 1, \ldots, k-j\} \cup \{\infty\}$, and $x_j = \infty$ implies $x_\ell = \infty$ for all $\ell > j$. We will write $\mathcal{X}_k$ for the set of all refined shapes over alphabet size k.*

We are going to see later that the total number of refined shapes is bounded by $O((k+1)!)$. The operation refine$(\cdot)$ introduced below formalizes the intuition discussed earlier.

▶ **Definition 7.** *Let $\mathbf{x} = (x_0, \ldots, x_k)$ be such that $x_j \in \mathbb{N} \cup \{\infty\}$ for all j. Let*

$$\text{refine}(\mathbf{x}) := \begin{cases} \mathbf{x} & \text{if } x_j \leq k - j \text{ for all } j, \\ (x_0, \ldots, x_{j_0 - 1}, \infty, \infty, \ldots) & j_0 \text{ is the smallest } j \text{ such that } x_j \geq k - j + 1. \end{cases}$$

We remark that refine$(\mathbf{x}) \in \mathcal{X}_k$ for any $\mathbf{x}$. Next we define insertions with respect to refined shapes. Given a refined shape $\mathbf{x} = (x_0, \ldots, x_k)$ and labels $a \leq b$, the shape obtained by inserting b under a, denoted $\mathbf{x}(a \leftarrow b)$, is defined as

$$\mathbf{x}(a \leftarrow b) := \begin{cases} \text{refine}\,(x_0, \ldots, x_{a-1}, x_a + 1, x_{a+1}, \ldots, x_{k-1}) & \text{if } x_a > 0 \text{ and } a = b, \\ \text{refine}\,(x_0, \ldots x_{a-1}, x_a - 1, x_{a+1}, \ldots, x_{b-1}, x_b + 2, x_{b+1}, \ldots, x_{k-1}) & \text{if } x_a > 0 \text{ and } a < b, \\ \bot & \text{if } x_a = 0. \end{cases}$$

where we followed the convention that $\infty > 0$ and $\infty + c = \infty$ for any constant c.

Now we are ready to state the dynamic programming algorithm. In the following, we fix $(a_1, a_2, \ldots, a_n)$ as the input sequence. For $\mathbf{x} \in \mathcal{X}_k$ and $i \in [n]$ define $\mathsf{LHS}[i, \mathbf{x}]$ to be the length of the longest heapable subsequence in the prefix sequence $(a_1, a_2, \ldots, a_i)$, with an additional constraint that the refined shape of the heap constructed from the subsequence should be $\mathbf{x}$.

We write $\mathsf{LHS}[i, \mathbf{x}] = -\infty$ if there is no feasible solution (i.e. shape $\mathbf{x}$ is not reachable by any subsequence of $(a_1, \ldots, a_i)$). With this definition, the longest heapable subsequence of the given sequence has length $\max_{\mathbf{x} \in \mathcal{X}_k} \mathsf{LHS}[n, \mathbf{x}]$. Our goal now is to compute $\mathsf{LHS}[n, \mathbf{x}]$ for each $\mathbf{x} \in \mathcal{X}_k$.

For a label $v \in \{1, 2, \ldots, k\}$ and two refined shapes $\mathbf{x}$ and $\mathbf{x}'$, we say that $\mathbf{x}$ *is reachable from* $\mathbf{x}'$ *via an insertion of* v if there exists $b \leq v$ such that $\mathbf{x}'(b \leftarrow v) = \mathbf{x}$. We denote by $\mathtt{prev}(\mathbf{x}, v)$ the set of refined shapes from which $\mathbf{x}$ is reachable via an insertion of v. We show that LHS satisfies the following recurrence relation.

▶ **Lemma 8.** *For every $i \in [n]$ and $\mathbf{x} \in \mathcal{X}_k$, we have that*

$$\mathit{LHS}[i, \mathbf{x}] = \max \left\{ \mathit{LHS}[i-1, \mathbf{x}], \max_{\mathbf{x}' \in \mathit{prev}(\mathbf{x}, a_i)} \left\{ \mathit{LHS}[i-1, \mathbf{x}'] \right\} + 1 \right\}.$$

Proof. We will show that

$$\mathsf{LHS}[i, \mathbf{x}] \leq \max \left\{ \mathsf{LHS}[i-1, \mathbf{x}], \max_{\mathbf{x}' \in \mathtt{prev}(\mathbf{x}, a_i)} \left\{ \mathsf{LHS}[i-1, \mathbf{x}'] \right\} + 1 \right\}$$

as the other direction is trivial. Let us fix an optimal heapable subsequence s of $(a_1, \ldots, a_i)$. If a_i does not belong to s, it must be the case that s is also an optimal heapable subsequence of $(a_1, \ldots, a_{i-1})$. In this case $\mathsf{LHS}[i, \mathbf{x}] = \mathsf{LHS}[i-1, \mathbf{x}]$. If a_i belongs to s, we further fix an optimal heap H (with refined shape $\mathbf{x}$) and assume that a_i is inserted under an element with value b in H. Removing a_i from H results in a heap H' with a shape $\mathbf{x}'$ satisfying $\mathbf{x}'(b \leftarrow a_i) = \mathbf{x}$. In particular, $\mathbf{x}' \in \mathtt{prev}(\mathbf{x}, a_i)$. In this case, $\mathsf{LHS}[i, \mathbf{x}] = \mathsf{LHS}[i-1, \mathbf{x}'] + 1 \leq \max_{\mathbf{x}' \in \mathtt{prev}(\mathbf{x}, a_i)} \left\{ \mathsf{LHS}[i-1, \mathbf{x}'] \right\} + 1$. ◀

Proof of Theorem 1. Given Lemma 8, it remains to show that the recurrence relation can be implemented in time $(k+1)! \cdot k \cdot O(n)$. We observe that the number of subproblems is bounded by $O(n|\mathcal{X}_k|)$. The set $\mathtt{prev}(\mathbf{x}, a_i)$ can be enumerated in time $O(k)$ by inverting the operation $\mathbf{x}'(b \leftarrow a_i)$ for each $b \leq a_i$. Therefore, it suffices to show that $|\mathcal{X}_k| = O((k+1)!)$.

In order to bound the size of $\mathcal{X}_k$, we observe that for every $\mathbf{x} = (x_0, x_1, \ldots, x_k) \in \mathcal{X}_k$, we have that $x_0 = 0$ unless $\mathbf{x} = (1, 0, \ldots, 0)$, and that $x_j \in \{0, 1, \ldots, k-j\} \cup \{\infty\}$ for $j \geq 1$. Therefore $|\mathcal{X}_k| \leq 1 + \prod_{j=1}^{k} (k - j + 2) = (k+1)! + 1$. ◀

Algorithm 1 gives an implementation of this dynamic programming algorithm. This implementation requires space complexity $O((k+1)! n \cdot \log n)$, which can be optimized to $O((k+1)! \cdot \log n)$ using a standard rolling array technique: we observe that in the recurrence relation, $\mathsf{LHS}[i, \mathbf{x}]$ depends only on $\mathsf{LHS}[i-1, \mathbf{x}']$ but not on $\mathsf{LHS}[j, \mathbf{x}']$ for any $j < i-1$. Therefore the values $\mathsf{LHS}[i-2, \mathbf{x}]$ become obsolete and the space can be recycled to store new values. Essentially, we only need two arrays $\mathsf{LHS}_1[\mathbf{x}]$ and $\mathsf{LHS}_2[\mathbf{x}]$ and store new values alternately between them.

▶ Remark. We note that our dynamic programming algorithm also works in the streaming model, where the elements of the input sequence have to be processed one by one without storing all of them in memory and the goal is to find the length of a longest heapable subsequence of the input that has arrived so far. For constant alphabet size k, the space complexity of our algorithm is $O(\log n)$.

Algorithm 1 Longest Heapable Subsequence for Alphabet Size k.

Input: A sequence $a = (a_1, \ldots, a_n)$ such that $\forall i \in [n]$, $a_i \in \{1, 2, \ldots, k\}$.
Output: The length of longest heapable subsequence in a.

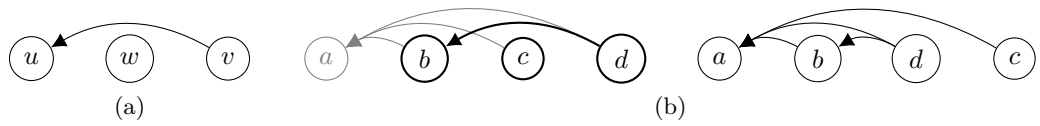
$\texttt{LHS}(a_1, a_2, \ldots, a_n)$:
$\mathcal{X} \leftarrow \{(1, 0, \ldots, 0)\}$ ▷ $\mathcal{X}$ maintains a set of reachable refined shapes
$\texttt{LHS} \leftarrow$ integer array of size $n \times (k+1) \times k \times \ldots 2 \times 1$
$\texttt{LHS}[0, (1, 0, \ldots, 0)] \leftarrow 0$
for $i \leftarrow 1$ to n **do** ▷ DP main body
  for $\mathbf{x} \in \mathcal{X}$ **do**
    $\texttt{LHS}[i, \mathbf{x}] \leftarrow \texttt{LHS}[i-1, \mathbf{x}]$ ▷ Discard a_i
  for $\mathbf{x} \in \mathcal{X}$ **do**
    for $b \in \{b' : 0 \le b' \le a_i, x_{b'} > 0\}$ **do**
      $\mathbf{x}' \leftarrow \mathbf{x}(b \leftarrow a_i)$ ▷ Insert a_i under b to reach refined shape $\mathbf{x}'$
      if $\mathbf{x}' \notin \mathcal{X}$ **then** ▷ First time reaching shape $\mathbf{x}'$
        $\mathcal{X} \leftarrow \mathcal{X} \cup \{\mathbf{x}'\}$
        $\texttt{LHS}[i, \mathbf{x}'] \leftarrow \texttt{LHS}[i-1, \mathbf{x}] + 1$
      else if $\texttt{LHS}[i, \mathbf{x}'] < \texttt{LHS}[i-1, \mathbf{x}] + 1$ **then**
        $\texttt{LHS}[i, \mathbf{x}'] \leftarrow \texttt{LHS}[i-1, \mathbf{x}] + 1$
return $\max \{\texttt{LHS}[n, \mathbf{x}] : \mathbf{x} \in \mathcal{X}\}$

3 γ-Alphabet Size of Permutation DAGs

In this section, we consider the problem of computing the γ-alphabet size of a permutation DAG G, where γ is a given topological ordering of G. We give an efficient algorithm in Section 3.1 and a min-max relation in Section 3.2. We also give a polyhedral description in Section 3.3. We begin with some useful background on permutation DAGs.

We recall that a directed graph G is a permutation DAG if there exists a sequence σ such that $\mathsf{PermDAG}(\sigma)$ is isomorphic to G. We note that permutation DAGs are *transitively closed*, i.e., for a permutation DAG $G = (V, A)$, if $(u, v), (v, w) \in A$, then $(u, w) \in A$. In order to recognize if a given DAG is a permutation DAG, we need the notion of *umbrella-free ordering* defined below (see Figure 4 for an example). This notion will also help us recognize if $\alpha(G, \gamma)$ is finite.

▶ **Definition 9** (Umbrella-free Order). *Let $G = (V, A)$ be an n-vertex DAG. An order $\gamma : V \to [n]$ of V is* umbrella-free *if for all $(v, u) \in A$ and for every vertex $w \in V$ with $\gamma(u) < \gamma(w) < \gamma(v)$, either $(w, u) \in A$ or $(v, w) \in A$ (or both).*



(a) (b)

Figure 4 (a) Scenario when the triple (u, w, v) is an *umbrella*. (b) Two topological orderings of the same DAG. The order (a, b, c, d) is not umbrella-free due to the (highlighted) umbrella (b, c, d), while the the order (a, b, d, c) is umbrella-free.

The following lemma characterizes permutation DAGs in terms of the existence of an umbrella-free topological ordering.

▶ **Lemma 10** ([11, 8]). *Let $G = (V, A)$ be a transitively closed DAG. Then G is a permutation DAG if and only if there exists an umbrella-free topological ordering of G. Moreover, there exists a polynomial-time algorithm to verify if a given DAG G is a permutation DAG and if so, then construct an umbrella-free topological ordering of G.*

Lemma 10 implies that $\alpha(G, \gamma)$ is finite if and only if γ is an umbrella-free topological ordering of G.

3.1 Algorithm

In this section, we will prove Theorem 2 – we will give an algorithm to compute the γ-alphabet size of a given permutation DAG G, i.e., $\alpha(G, \gamma)$, where γ is a topological ordering of G.

We note that umbrella-freeness of a given topological ordering can be verified in polynomial-time, so we may henceforth assume that the input γ is in fact an umbrella-free topological ordering of G. We will give an iterative algorithm to compute $\alpha(G, \gamma)$. We observe that computing $\alpha(G, \gamma)$ involves assigning a value to each vertex of G such that the sequence obtained by ordering the values of the vertices in the same order as γ gives the same permutation DAG as G. At each iteration, our algorithm will choose a vertex of G and assign a value to it. The next definition will allow us to formally define the choice of this vertex.

▶ **Definition 11** (Fully Suffix Connected Vertex). *Let $G = (V, A)$ be a permutation DAG and γ be a topological ordering of G. A vertex $u \in V$ is* fully suffix connected *if for all $v \in V$ such that $\gamma(v) > \gamma(u)$, we have $(v, u) \in A$. The* γ-least fully suffix connected (γ-LFSC) vertex *is the fully suffix connected vertex u with smallest $\gamma(u)$.*

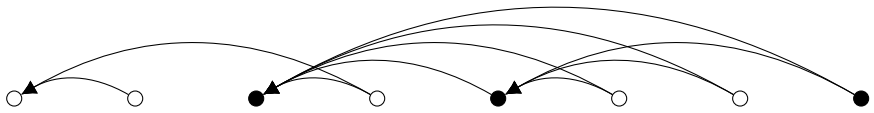


Figure 5 The DAG in the given topological order γ has 3 fully suffix connected vertices that are depicted as filled circles. The leftmost fully suffix connected vertex is the (unique) γ-LFSC vertex.

See Figure 5 for an example showing fully suffix connected vertices. We note that γ-LFSC is unique. The following lemma states a useful property of the γ-LFSC vertex.

▶ **Lemma 12.** *Let $G = (V, A)$ be a permutation DAG and γ be an umbrella-free topological ordering of G. Then, the γ-LFSC vertex has no outgoing arcs in G.*

Proof. Let $v \in V$ be the γ-LFSC and suppose for contradiction that v has an outgoing arc in G. Let u be the vertex with largest $\gamma(u)$ such that $(v, u) \in A$. We note that $\gamma(u) < \gamma(v)$ since γ is a topological ordering. We will show that such a vertex u is fully suffix connected and hence contradicts the γ-least fully suffix connected property of vertex v.

We first show that for every vertex $w \in V$ such that $\gamma(w) \geq \gamma(v)$, we have $(w, u) \in A$. For $w = v$, this follows since $(v, u) \in A$ by the choice of u. Let w be a vertex such that $\gamma(w) > \gamma(v)$. Since v is fully suffix connected, we have that $(w, v) \in A$. Also, since G is a permutation DAG, it is transitively closed. Hence, $(v, u) \in A$ implies that $(w, u) \in A$.

Next, we show that for every vertex $w \in V$ such that $\gamma(u) < \gamma(w) < \gamma(v)$, we have $(w, u) \in A$. Let w be a vertex such that $\gamma(u) < \gamma(w) < \gamma(v)$. By assumption, the ordering γ is umbrella-free. Thus, at least one of (w, u) or (v, w) must exist in A. However, $(v, w) \notin A$ as otherwise, w will contradict the choice of vertex u. Therefore, $(w, u) \in A$. ◀

We now discuss a high level overview of our iterative greedy algorithm for computing $\alpha(G,\gamma)$. During the first iteration, the algorithm greedily chooses the γ-LFSC vertex v_1 (say) in $G_1 := G$ to assign the smallest alphabet, namely $\sigma(v_1) = 1$. The vertex v_1 and its incident edges are deleted from G_1 to form G_2, and the remaining $n-1$ vertices $V\backslash\{v\}$ are ordered in the same relative order as γ – denote this ordering as γ_2. In the second iteration, our algorithm greedily chooses the γ_2-LFSC vertex v_2 (say) in G_2 to assign the next smallest alphabet – the next smallest alphabet is chosen based on whether v_2 lies to the left or right of v_1: if v_2 lies to the left of v_1 with respect to γ, then we set $\sigma(v_2) = \sigma(v_1)+1$, otherwise we set $\sigma(v_2) = \sigma(v_1)$. This iterative removal and assignment process continues for n iterations, i.e., until all vertices are removed from G. The final output sequence will just be the sequence of assigned values in the order of vertices in γ. Before presenting our complete algorithm (Algorithm 2), we introduce a definition to formalize the reordering of vertices after removing a vertex from G – this will allow us to obtain γ_{i+1} from γ_i.

▶ **Definition 13** (Projected order). *Let $G = (V, A)$ be an n-vertex DAG, and γ be a topological ordering of V. Let $H = G - v$. Then the* projection of γ onto H, *denoted by* $\mathsf{Proj}_H[\gamma] : V\backslash\{v\} \to [n-1]$, *is defined as*

$$\mathsf{Proj}_H[\gamma](u) = \begin{cases} \gamma(u) & \text{if } \gamma(u) < \gamma(v), \\ \gamma(u) - 1 & \text{if } \gamma(u) > \gamma(v). \end{cases}$$

Armed with the notions of fully suffix connected vertices and projected order, we state our algorithm below.

Algorithm 2 `GreedyAssign` algorithm to compute $\alpha(G,\gamma)$.

Input: Permutation DAG $G = (V, A)$ on n vertices in umbrella-free topological order $\gamma : V \to [n]$
Output: Sequence σ of length n

`GreedyAssign`(G,γ):
1: Initialize $\alpha \leftarrow 1$; $G_1 \leftarrow G$; $\gamma(v_0) \leftarrow -\infty$; $\gamma_1 \leftarrow \gamma$
2: **for** $i \leftarrow 1$ to n **do**
3: $v_i \leftarrow \gamma_i$-LFSC in G_i
4: **if** $\gamma(v_i) < \gamma(v_{i-1})$ **then** $\alpha \leftarrow \alpha + 1$
5: $G_{i+1} \leftarrow G_i - v_i$
6: $\gamma_{i+1} \leftarrow \mathsf{Proj}_{G_{i+1}}[\gamma_i]$
7: $\sigma(v_i) \leftarrow \alpha$
8: **Return** $\sigma \leftarrow (\sigma(\gamma^{-1}(1)) \dots \sigma(\gamma^{-1}(n)))$

The algorithm can be implemented to run in polynomial-time since a γ-LFSC vertex in G can be computed in polynomial-time. We now prove the correctness of the algorithm. Let $G = (V, A)$ be an n-vertex permutation DAG, and γ be an umbrella-free topological ordering of G. Let $v_1, \dots, v_n$ be the sequence of vertices chosen in the execution of `GreedyAssign`(G,γ). Let α_i, G_i and γ_i denote the alphabet size α at the end of the i^{th} iteration, the remaining subgraph at the start of the i^{th} iteration, and γ projected onto G_i respectively. Finally, let σ be the sequence returned by `GreedyAssign`(G,γ). We have the following observations about the execution of the algorithm.

▶ **Observation 14.** *The vertex v_i has no outgoing arcs in G_i for all $i \in [n]$.*

▶ **Observation 15.** *If $\gamma(v_{i+1}) < \gamma(v_i)$ then $\sigma(v_{i+1}) = \sigma(v_i)+1$ and $(v_i, v_{i+1}) \notin A$, otherwise $\sigma(v_{i+1}) = \sigma(v_i)$ and $(v_{i+1}, v_i) \in A$. Thus, alphabet assignments by* `GreedyAssign` *are non-decreasing with increasing iterations i.e. $\sigma(v_i) \le \sigma(v_j)$ for all $i, j \in [n]$ with $i < j$.*

Observation 14 directly follows from Lemma 12. Observation 15 is due to the conditional increment of the alphabet size, α, in `GreedyAssign`. The next two lemmas show feasibility and optimality of `GreedyAssign` respectively. Theorem 2 then immediately follows from Lemmas 16 and 17.

▶ **Lemma 16** (Feasibility of `GreedyAssign`). *Let* $\mathsf{PermDAG}(\sigma) = (\{t_1 \dots t_n\}, A')$. *Then* $\mathsf{PermDAG}(\sigma)$ *is isomorphic to* G *under the mapping* $\phi : \{t_1 \dots t_n\} \to V$ *given by* $\phi(t_i) = \gamma^{-1}(i)$.

Proof. We will prove isomorphism of the two graphs under ϕ by showing that $(u, v) \in A$ if and only if $(\phi^{-1}(u), \phi^{-1}(v)) \in A'$.

For the forward direction, it suffices to show that $\sigma(u) \leq \sigma(v)$ whenever $(u, v) \in A$. We observe that if $(u, v) \in A$, then $\gamma(v) < \gamma(u)$. By Observation 14, `GreedyAssign` must assign $\sigma(v)$ before $\sigma(u)$. Observation 15 then implies that $\sigma(u) \leq \sigma(v)$.

Next we show the contrapositive of the converse direction. Assume that $(u, v) \notin A$. We first consider the case when $\gamma(v) > \gamma(u)$. Let $\phi^{-1}(u) = t_{\gamma(u)}$ and $\phi^{-1}(v) = t_{\gamma(v)}$. By definition of permutation DAGs, $\mathsf{PermDAG}(\sigma)$ does not have arc (t_i, t_j) when $i < j$. Thus $(t_{\gamma(u)}, t_{\gamma(v)}) \notin A$. Next, we consider the case when $\gamma(v) < \gamma(u)$. For this, it suffices to show that $\sigma(u) < \sigma(v)$. Since $(u, v) \notin A$, the vertex v will never become fully suffix connected before the removal of u. Thus `GreedyAssign` sets $\sigma(u)$ before $\sigma(v)$. Thus, by Observation 15, we have that $\sigma(u) \leq \sigma(v)$. Let $u = v_i$ and $v = v_j$, where $i, j \in [n]$ are the iteration numbers during which `GreedyAssign` assigns $\sigma(u)$ and $\sigma(v)$ respectively. Then, there exists k such that $i \leq k < j$ and $\gamma(v_{k+1}) < \gamma(v_k)$ as otherwise, Observation 15 would imply that $\gamma(v_i) < \gamma(v_j)$, a contradiction. Thus, $\sigma(u) < \sigma(v)$. ◀

▶ **Lemma 17** (Optimality of `GreedyAssign`). *Let* σ^* *be a sequence achieving* $\alpha(G, \gamma)$. *Then,* $\sigma(v_i) \leq \sigma^*(v_i)$ *for all* $i \in [n]$.

Proof. We will show this by induction on i. For the base case of $i = 1$, `GreedyAssign` always sets $\sigma(v_1) = 1$, the smallest possible alphabet assignment. Thus $\sigma(v_1) \leq \sigma^*(v_1)$ holds. For the induction step, let $i \geq 2$. We have the following two cases based on whether `GreedyAssign` incremented the alphabet size while assigning v_i.

1. Suppose $\sigma(v_i) = \sigma(v_{i-1})$. By the description of the algorithm `GreedyAssign`, we have that v_{i-1} is fully suffix connected in G_{i-1} and $\gamma(v_i) > \gamma(v_{i-1})$. Thus, the arc (v_i, v_{i-1}) must exist in G_{i-1} and so also in G. It follows that

   $$\sigma(v_i) = \sigma(v_{i-1}) \leq \sigma^*(v_{i-1}) \leq \sigma^*(v_i).$$

   Here, the first inequality is by the induction hypothesis, while the second inequality is due to the observation that $(v_i, v_{i-1}) \in A$.
2. Suppose $\sigma(v_i) \neq \sigma(v_{i-1})$. By the description of the algorithm `GreedyAssign`, we have that $\gamma(v_i) < \gamma(v_{i-1})$. Thus by Observation 14, the arc (v_{i-1}, v_i) does not exist in G_{i-1} and hence, does not exist in G. It follows that

   $$\sigma(v_i) = \sigma(v_{i-1}) + 1 \leq \sigma^*(v_{i-1}) + 1 \leq \sigma^*(v_i).$$

   The equality relation is due to Observation 15. The first inequality is due to the induction hypothesis while the second inequality is due to our observation that $(v_{i-1}, v_i) \notin A$. ◀

▶ Remark. Algorithm 2 can be implemented to run in $O(|V| + |A|)$ time. This can be done by using a *priority queue* data structure initialized as a *stack*. All fully suffix connected vertices should be added to the priority queue with priorities being position in γ. The choice of vertex to assign is the vertex with the *minimum* priority. The alphabet size should be incremented whenever a vertex removal results in new vertices becoming fully suffix connected.

3.2 Min-Max Relation

Min-max relations are significant in optimization literature as they are strong indicators for the existence of a polynomial-time algorithm. In the context of algorithm design, min-max relations bring the optimization problem into $\mathsf{NP} \cap \mathsf{coNP}$, thus providing strong evidence for the existence of polynomial-time algorithms. In this section, we prove the min-max relation for $\alpha(G, \gamma)$, i.e., Theorem 3. A consequence of our min-max relation will be an alternative linear time algorithm for computing $\alpha(G, \gamma)$. We believe that the min-max relation could be a useful tool towards computing $\alpha(G)$. We restate and prove the min-max relation below (see Section 1.1 for the definition of the graph $H(G, \gamma)$ and weights w for this graph).

▶ **Theorem 3.** *Let $G = (V, A)$ be a permutation DAG and γ be a topological ordering of V such that $\alpha(G, \gamma)$ is finite. Then,*

$$\alpha(G, \gamma) = 1 + \max \left\{ \sum_{e \in P} w(e) : P \text{ is a path in } H(G, \gamma) \right\}.$$

Proof. We will show the equation by showing inequality in both directions. We begin by showing the lower bound on $\alpha(G, \gamma)$. Let P be any path in $H(G, \gamma)$, and σ be any sequence such that $\mathsf{PermDAG}(\sigma) = (\{t_1, \ldots, t_n\}, A')$ is isomorphic to G under the mapping $\phi : \{t_1, \ldots, t_n\} \to V$ given by $\phi(t_i) = \gamma^{-1}(i)$. For every arc $(\phi(t_i), \phi(t_j)) \in P$ such that $(\phi(t_i), \phi(t_j)) \in \overrightarrow{E}$, we have the following two observations. First, the arc $(\phi^{-1}(t_j), \phi^{-1}(t_i)) \notin A'$ as the arc $(\phi(t_j), \phi(t_i)) \notin A$. Second, $\sigma(\phi(t_i)) \geq \sigma(\phi(t_j)) + 1$ as $i < j$. It follows that

$$w(P) = \sum_{(u,v) \in P} w(u, v) = \sum_{(u,v) \in P \cap \overrightarrow{E}} w(u, v) \leq \sum_{(u,v) \in P \cap \overrightarrow{E}} \sigma(u) - \sigma(v) \leq \alpha(G, \gamma) - 1.$$

The first and second equations are by definition of $w(P)$ and the weight function w respectively. The first inequality is due to our observation that $\sigma(u) \geq \sigma(v) + 1$ whenever $(u, v) \in \overrightarrow{E}$. Let a and b be the first and last vertices on P. Then the final inequality follows from $\sigma(a) \geq 1$ and $\sigma(b) \leq \alpha(G, \gamma)$.

Next, we show the upper bound on $\alpha(G, \gamma)$. We recall that $v_1, \ldots, v_n$ is the order in which `GreedyAssign` processes vertices of G. Consider $P = (v_n, v_{n-1}, \ldots, v_1)$. To prove the upper bound, it suffices to show that (1) P is a path in $H(G, \gamma)$; and (2) $w(P) \geq \alpha(G, \gamma) - 1$. To prove (1), we show that $(v_i, v_{i-1}) \in A \cup \overrightarrow{E}$ for each $i \geq 2$. Consider the case when $\gamma(v_i) > \gamma(v_{i-1})$. Since v_{i-1} was γ_{i-1}-LFSC in G_i, the arc $(v_i, v_{i-1}) \in A$. Next, consider the case when $\gamma(v_i) < \gamma(v_{i-1})$. By Observation 14, we have that the arc $(v_{i-1}, v_i) \notin A$. Thus, the arc $(v_i, v_{i-1}) \in \overrightarrow{E}$ by definition of $\overrightarrow{E}$. We now prove (2). By Observation 15, and definitions of w and $\overrightarrow{E}$, we have $w(v_i, v_{i-1}) = \sigma(v_i) - \sigma(v_{i-1})$ It follows that

$$w(P) = \sum_{i=2}^{n} w(v_i, v_{i-1}) = \sum_{i=2}^{n} \sigma(v_i) - \sigma(v_{i-1}) = \alpha(G, \gamma) - 1.$$

The second equality is due to our previous observation. The final equality is due to the `GreedyAssign` assignments $\sigma(v_n) = \alpha(G, \gamma)$ and $\sigma(v_1) = 1$. ◀

We remark that although the RHS problem in the min-max relation given in Theorem 3 is the longest path problem in a directed graph, it can be solved in the graph $H(G, \gamma)$ owing to the following lemma. Lemma 18 allows the optimization problem in the RHS of Theorem 3 to be solved in $O(|V| + |A|)$ time by the classical dynamic programming algorithm for maximum weight path in a DAG. This leads to an alternative algorithm for computing $\alpha(G, \gamma)$.

▶ **Lemma 18.** *$H(G,\gamma)$ is a DAG.*

Proof. Suppose for contradiction that $H(G,\gamma)$ contains a cycle. Let $C = (u_1, u_2, \dots u_k, u_1)$ be a cycle with the smallest number of vertices. If $(u_1, u_3) \in A \cup \overrightarrow{E}$, then $C' = (u_1, u_3, \dots, u_k, u_1)$ is a cycle, contradicting our choice of C. Since $H(G,\gamma)$ is a tournament, the arc $(u_3, u_1) \in A \cup \overrightarrow{E}$, and $C' = (u_1, u_2, u_3, u_1)$ is also a cycle i.e. $k = 3$. We recall that the subgraph (V, A) is transitively closed. Thus, at most one edge of C can belong to A. To get the required contradiction, it suffices to show that the subgraph $(V, \overrightarrow{E})$ is transitively closed. Suppose for contradiction that $\overrightarrow{E}$ is not transitively closed. Then, there exist arcs $(u,v), (v,w) \in \overrightarrow{E}$ such that the arc $(u,w) \notin \overrightarrow{E}$. By definition of $\overrightarrow{E}$, we have that $\gamma(u) < \gamma(v) < \gamma(w)$. It follows that the arc $(w,u) \in A$, and the triple (u,v,w) is an umbrella in G ordered by γ. This contradicts that γ is umbrella-free. ◀

3.3 Polyhedral Description for γ-alphabet size

In this section, we give a polyhedral description for the convex-hull of sequences that are feasible for $\alpha(G,\gamma)$. As a consequence, it leads to an LP-based algorithm to compute $\alpha(G,\gamma)$. We emphasize that our polyhedral result is stronger than giving an LP-based algorithm to compute $\alpha(G,\gamma)$: it implies that one can efficiently compute an *integer-valued* sequence $\sigma = (\sigma(1), \dots, \sigma(n))$ with minimum weight $\sum_{i=1}^{n} w_i \sigma(i)$ for any given non-negative weights $w_1, \dots, w_n$ such that $\mathsf{PermDAG}(\sigma)$ is isomorphic to G under the mapping $\phi : \{t_1, \dots, t_n\} \to V$ given by $\phi(t_i) = \gamma^{-1}(i)$ for every $i \in [n]$. The following is the main result of this section.

▶ **Theorem 19.** *Let $G = (V, A)$ be an n-vertex permutation DAG and γ be an umbrella-free topological ordering of its vertices. Let $Q(G,\gamma)$ be the convex-hull of indicator vectors of $\mathbf{x} \in \mathbb{N}^n$ whose sequence $\sigma := (x_1, \dots, x_n)$ is such that $\mathsf{PermDAG}(\sigma) = (\{t_1, \dots, t_n\}, A')$ is isomorphic to G under the mapping $\phi : \{t_1, \dots, t_n\} \to V$ given by $\phi(t_i) = \gamma^{-1}(i)$ for all $i \in [n]$. Then,*

$$Q(G,\gamma) = \left\{ x \in \mathbb{R}^n \;\middle|\; \begin{array}{ll} x_{\gamma(u)} \le x_{\gamma(v)} & \forall (v,u) \in A, \\ x_{\gamma(v)} \le x_{\gamma(u)} - 1 & \forall (v,u) \notin A \text{ with } \gamma(u) < \gamma(v), \text{ and} \\ x_i \ge 1 & \forall\ i \in [n] \end{array} \right\}.$$

For notational convenience, let $P(G,\gamma)$ denote the polyhedron defined in the RHS of Theorem 19. Before proving Theorem 19, we describe how $\alpha(G,\gamma)$ can be obtained by optimizing over $P(G',\gamma')$ for a graph G' and an ordering γ' obtained from G and γ. Let $G' = (V', A')$ be obtained from G by adding a vertex t with edges (t,u) for all $u \in V$ and $\gamma' : V' \to [n+1]$ be defined as $\gamma'(u) = \gamma(u)$ if $u \in V$ and $\gamma'(t) = n+1$. We note that if G is a permutation DAG and γ is an umbrella-free topological ordering of G, then G' is also a permutation DAG and γ' is an umbrella-free topological ordering of G'. Moreover, we also have that

$$\alpha(G,\gamma) = \min \left\{ x_{\gamma'(n+1)} : x \in Q(G',\gamma') \right\}.$$

Thus, by Theorem 19, the γ-alphabet size of G, i.e., $\alpha(G,\gamma)$, can be computed by optimizing along the objective direction $(0, \dots, 0, 1) \in \mathbb{R}^{n+1}$ over the polyhedron $P(G',\gamma')$.

We now prove Theorem 19.

Proof of Theorem 19. We recall that a point $\mathbf{x}$ is an *extreme point* of a polyhedron if $\mathbf{x}$ cannot be expressed as a convex combination of any two distinct points in the polyhedron. Any extreme point x of $Q(G,\gamma)$ satisfies the constraints defining $P(G,\gamma)$. Thus, $Q(G,\gamma) \subseteq$

$P(G,\gamma)$. In order to show equality, it suffices to show that all extreme points of $P(G,\gamma)$ are integral. Lemma 20 shows that all extreme points of $P(G,\gamma)$ are integral, thus completing the proof of Theorem 19. ◀

▶ **Lemma 20.** *Let $G=(V,A)$ be an n-vertex DAG and γ be a topological ordering of its vertices. If $\mathbf{x}$ is an extreme point of $P(G,\gamma)$, then $\mathbf{x}\in\mathbb{Z}^n$.*

Proof. Suppose for contradiction that $\mathbf{x}$ is non-integral. We will show the existence of two points in $P(G,\gamma)$ such that $\mathbf{x}$ is a convex combination of these points. Let $S:=\{i : x_i \notin \mathbb{Z}\}$. We note that the set S is non-empty due to our choice of $\mathbf{x}$. Let $\epsilon\in\mathbb{R}$ be as follows

$$\epsilon := \min_{i\in S}\left\{\min(x_i-\lfloor x_i\rfloor, \lceil x_i\rceil - x_i)\right\}.$$

Since S is non-empty, we have $\epsilon > \frac{\epsilon}{2} > 0$. Let $\mathbf{y}\in\mathbb{R}^n$ be defined as follows:

$$y_i := \begin{cases} \epsilon/2 & \text{if } i\in S,\\ 0 & \text{otherwise.}\end{cases}$$

We note that $\mathbf{x}=\frac{1}{2}(\mathbf{x}+\mathbf{y})+\frac{1}{2}(\mathbf{x}-\mathbf{y})$. It suffices to show that $\mathbf{x}+\mathbf{y}, \mathbf{x}-\mathbf{y}\in P$. We will show that the point $\mathbf{x}+\mathbf{y}\in P$ and remark that the proof of $\mathbf{x}-\mathbf{y}\in P$ is along very similar lines. We observe that $\mathbf{y}\geq 0$.

Constraint (3) is always satisfied as $x_i+y_i\geq x_i\geq 1$. We first focus on constraint (1). Consider any arc $(v,u)\in A$. Since $\mathbf{y}\geq 0$, the constraint is easily seen to be satisfied in the cases where (1) $x_{\gamma(u)}, x_{\gamma(v)}\in\mathbb{Z}$; (2) $x_{\gamma(u)}, x_{\gamma(v)}\notin\mathbb{Z}$; and (3) $x_{\gamma(u)}\in\mathbb{Z}$ but $x_{\gamma(v)}\notin\mathbb{Z}$. Consider the case when $x_{\gamma(u)}\notin\mathbb{Z}$ but $x_{\gamma(v)}\in\mathbb{Z}$. Then, we have that

$$x_{\gamma(u)}+y_{\gamma(u)} < x_{\gamma(u)}+\epsilon \leq \left\lceil x_{\gamma(u)}\right\rceil \leq x_{\gamma(v)} = x_{\gamma(v)}+y_{\gamma(v)}.$$

The first inequality is by $y_i\leq\epsilon/2$ for all $i\in[n]$. The second inequality is by definition of ϵ. The third inequality is due to $\mathbf{x}\in P$ and our case assumption that $x_{\gamma(v)}\in\mathbb{Z}$. The equality relation is by definition of $\mathbf{y}$.

Next, we consider constraint (2). Let $\gamma(u)<\gamma(v)$ but $(v,u)\notin A$. Similar to the above analysis, the constraint is easily seen to be satisfied in the cases where (1) $x_{\gamma(u)}, x_{\gamma(v)}\in\mathbb{Z}$; (2) $x_{\gamma(u)}, x_{\gamma(v)}\notin\mathbb{Z}$; and (3) $x_{\gamma(u)}\notin\mathbb{Z}$ but $x_{\gamma(v)}\in\mathbb{Z}$. Consider the case when $x_{\gamma(u)}\in\mathbb{Z}$ but $x_{\gamma(v)}\notin\mathbb{Z}$. Then, we have that

$$x_{\gamma(v)}+y_{\gamma(v)} < x_{\gamma(v)}+\epsilon \leq \left\lceil x_{\gamma(u)}\right\rceil \leq x_{\gamma(u)} = x_{\gamma(u)}+y_{\gamma(u)}.$$

The first inequality is due to $y_i\leq\epsilon/2$ for all $i\in[n]$. The second inequality is by definition of ϵ. The third inequality is due to $\mathbf{x}\in P$ and our case assumption that $x_{\gamma(u)}\in\mathbb{Z}$. The equality relation is by definition of $\mathbf{y}$. ◀

Based on Lemma 20, it is natural to wonder if the integral extreme points of $P(G,\gamma)$ have any combinatorial interpretation when G is an arbitrary DAG and γ is an arbitrary topological ordering of G. The following lemma shows that integrality of $P(G,\gamma)$ is useful only when G is a *permutation* DAG and γ is an *umbrella-free* topological ordering of G.

▶ **Lemma 21.** *Let G be a DAG and γ be a topological ordering of G. Then, $P(G,\gamma)$ is non-empty if and only if G is a permutation DAG and γ is umbrella-free.*

Proof. The reverse direction follows from the correctness of `GreedyAssign` (Lemma 16). We focus on proving the forward direction. Let $\mathbf{x} \in P(G, \gamma)$ be a feasible point. It suffices to show that G is transitively closed and γ is umbrella-free.

First, assume for contradiction that G is not transitively closed. Then, there exist arcs $(u, v), (v, w) \in A$ such that the arc $(u, w) \notin A$. Since $\mathbf{x}$ is feasible, we have the following: (1) $x_{\gamma(v)} \leq x_{\gamma(u)}$; (2) $x_{\gamma(w)} \leq x_{\gamma(v)}$; and (3) $x_{\gamma(u)} \leq x_{\gamma(w)} - 1$. However, these inequalities do not admit any feasible solution, a contradiction.

Next, assume for contradiction that γ is not umbrella-free. Then, there exists a triple (u, v, w) such that $\gamma(u) < \gamma(v) < \gamma(w)$, and the arc $(w, u) \in A$, but the arcs $(v, u), (w, v) \notin A$. Since the point $\mathbf{x}$ is feasible, we have the following: $x_{\gamma(w)} \leq x_{\gamma(v)} - 1$; (2) $x_{\gamma(v)} \leq x_{\gamma(u)} - 1$; and (3) $x_{\gamma(u)} \leq x_{\gamma(w)}$. However, these inequalities do not admit any feasible solution, a contradiction. ◀

References

1 János Balogh, Cosmin Bonchiş, Diana Diniş, Gabriel Istrate, and Ioan Todinca. The heapability of finite partial orders. *Discrete Mathematics and Theoretical Computer Science*, 22(1), 2020.

2 Anne-Laure Basdevant, Lucas Gerin, Jean-Baptiste Gouéré, and Arvind Singh. From Hammersley's lines to Hammersley's trees. *Probability Theory and Related Fields*, pages 1–51, 2016.

3 Anne-Laure Basdevant and Arvind Singh. Almost-sure asymptotic for the number of heaps inside a random sequence. *Electronic Communications in Probability*, 23(17), 2018.

4 Cosmin Bonchiş, Gabriel Istrate, and Vlad Rochian. The language (and series) of Hammersley-type processes. In *Proceedings of the Eighth Conference on Machines Computation and Universality (MCU'18)*, volume 10881 of *Lecture Notes in Computer Science*, 2018.

5 John Byers, Brent Heeringa, Michael Mitzenmacher, and Georgios Zervas. Heapable sequences and subseqeuences. In *Proceedings of the Eighth Workshop on Analytic Algorithmics and Combinatorics*, ANALCO '11, pages 33–44, 2011.

6 Karthekeyan Chandrasekaran, Elena Grigorescu, Gabriel Istrate, Shubhang Kulkarni, Young-San Lin, and Minshen Zhu. The maximum binary tree problem. *In Proceedings of the 32nd European Symposium on Algorithms (ESA'20), to appear*, 2020. arXiv preprint: 1909.07915. `arXiv:1909.07915`.

7 Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Verlag, 2012.

8 Martin Charles Golumbic. Chapter 7 - permutation graphs. In Martin Charles Golumbic, editor, *Algorithmic Graph Theory and Perfect Graphs*, pages 157–170. Academic Press, 1980.

9 Gabriel Istrate and Cosmin Bonchiş. Partition into heapable sequences, heap tableaux and a multiset extension of Hammersley's process. In *Proceedings of the 26th Annual Symposium on Combinatorial Pattern Matching (CPM'15), Ischia, Italy*, volume 9133 of *Lecture Notes in Computer Science*, pages 261–271. Springer, 2015.

10 Gabriel Istrate and Cosmin Bonchiş. Heapability, interactive particle systems, partial orders: Results and open problems. In *Proceedings of the 18th International Conference on Descriptional Complexity of Formal Systems (DCFS'2016), Bucharest, Romania*, volume 9777 of *Lecture Notes in Computer Science*, pages 18–28. Springer, 2016.

11 A. Pnueli, A. Lempel, and S. Even. Transitive orientation of graphs and identification of permutation graphs. *Canadian Journal of Mathematics*, 23(1):160–175, 1971.

12 Jaclyn Porfilio. A combinatorial characterization of heapability. Master's thesis, Williams College, 2015.

13 Dan Romik. *The surprising mathematics of longest increasing subsequences*. Cambridge University Press, 2015.

Recognizing Proper Tree-Graphs

Steven Chaplick
Maastricht University, The Netherlands
s.chaplick@maastrichtuniversity.nl

Petr A. Golovach
Department of Informatics, University of Bergen, Norway
petr.golovach@uib.no

Tim A. Hartmann
RWTH Aachen, Germany
hartmann@algo.rwth-aachen.de

Dušan Knop
Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Czech Republic
dusan.knop@fit.cvut.cz

Abstract

We investigate the parameterized complexity of the recognition problem for the proper H-graphs. The H-graphs are the intersection graphs of connected subgraphs of a subdivision of a multigraph H, and the properness means that the containment relationship between the representations of the vertices is forbidden. The class of H-graphs was introduced as a natural (parameterized) generalization of interval and circular-arc graphs by Biró, Hujter, and Tuza in 1992, and the proper H-graphs were introduced by Chaplick et al. in WADS 2019 as a generalization of proper interval and circular-arc graphs. For these graph classes, H may be seen as a structural parameter reflecting the distance of a graph to a (proper) interval graph, and as such gained attention as a structural parameter in the design of efficient algorithms. We show the following results.

- For a tree T with t nodes, it can be decided in $2^{\mathcal{O}(t^2 \log t)} \cdot n^3$ time, whether an n-vertex graph G is a proper T-graph. For yes-instances, our algorithm outputs a proper T-representation. This proves that the recognition problem for proper H-graphs, where H required to be a tree, is fixed-parameter tractable when parameterized by the size of T. Previously only NP-completeness was known.
- Contrasting to the first result, we prove that if H is not constrained to be a tree, then the recognition problem becomes much harder. Namely, we show that there is a multigraph H with 4 vertices and 5 edges such that it is NP-complete to decide whether G is a proper H-graph.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Fixed parameter tractability

Keywords and phrases intersection graphs, H-graphs, recognition, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.8

Related Version `https://arxiv.org/abs/2011.11670`

Funding *Petr A. Golovach*: Author supported by the project MULTIVAL of the Research Council of Norway.
Dušan Knop: Supported by the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16 019/0000765 "Research Center for Informatics".

Acknowledgements We thank Peter Zeman for initial discussions on this work, particularly relating to the NP-completeness of proper H-graph recognition. We also thank an anonymous reviewer to point to a mistake in an earlier version.

1 Introduction

An *intersection representation* of a graph $G = (V, E)$ is a collection of nonempty sets $\{M_v \mid v \in V(G)\}$ over a given universe such that $\{u, v\}$ is an edge of G if and only if $M_u \cap M_v \neq \emptyset$. A large area of research in graph algorithms is the study of restricted families of graphs arising from specialized intersection representations, e.g., the *interval* graphs are the graphs with an intersection representation where the sets are intervals of $\mathbb{R}$, and the *circular-arc* graphs are intersection graphs of families of arcs of the circle. The interval graphs and similarly defined graph classes are often motivated from application areas such as circuit layout problems [24, 4], scheduling problems [22], biological problems [19], or the study of wireless networks [16]. We refer to the books [5, 15] for an introduction and survey of the known results on the related graph classes.

A key feature of these specialized intersection representations is that they can often be used to obtain efficient algorithms for standard combinatorial optimization problems, e.g., it is well-known [5] that the Clique and Independent Set problems, as well as various coloring and Hamiltonicity problems are all efficiently solvable on interval graphs, and the algorithms often leverage on the intersection representation. This led Biró et al. [2] to introduce an elegant family of intersection graph classes, called *H-graphs*, over universes that may be seen as (multi) graphs. Formally, the parameter H is a multigraph, and a graph G is an *H-graph* when there is a *subdivision* H_{sub} of H[1] and a collection $\mathcal{M} = \{M_v \subseteq V(H_{\text{sub}}) \mid v \in V(G)\}$ of sets, where we refer to M_v as the *model of* v, such that

- for every $v \in V(G)$, its model M_v induces a connected subgraph of H_{sub}, and
- $\{u, v\} \in E(G)$ if and only if $M_u \cap M_v \neq \emptyset$.

In this context, H_{sub} *represents* G. Observe that, for any interval graph G, there is a path P (i.e., a subdivision of K_2) such that P represents G meaning that the interval graphs are precisely the K_2-graphs. Similarly, the circular-arc graphs are C-graphs for any cycle C, and every chordal graph is a T-graph for some tree T, i.e., indeed, H-graphs can be seen as a parameterized generalization of several important families of intersection graphs, where H is a parameter reflecting the distance of a graph to an interval graph. Biró et al. [2] provided polynomial-time algorithms (via treewidth-based techniques) for coloring problems on H-graphs for fixed H, but left many interesting problems open.

The classes of H-graphs have seen renewed interest in recent years concerning their structure and recognition [8], relation to other graph parameters [8, 12], and primarily regarding the computational complexity of standard algorithmic problems when parameterized by the size of H [1, 7, 8, 9, 12, 17, 18]. Of particular relevance to our paper is the work on Hamiltonicity problems [7] as it introduces proper H-graphs, which are to *proper interval graphs* as H-graphs are to interval graphs. Namely, for a graph G, a subdivision H_{sub} of H *properly represents* G when H_{sub} represents G using models $\{M_v \subseteq V(H_{\text{sub}}) \mid v \in V(G)\}$ such that for each $u, v \in V(G)$, neither $M_u \subseteq M_v$ nor $M_v \subseteq M_u$. In particular, on proper H-graphs polynomial size kernels (in the size of H) were developed for various Hamiltonicity problems [7], but the recognition problems were left open.

The cornerstone problem for every graph class is recognizability, and we focus on the recognition problem for proper H-graphs both when H is part of the input and when H is fixed. It is important to note that the problem of testing whether for a given graph G and given tree T, the graph G is a T-graph is NP-complete [20, Theorem 4]. In fact, the reduction [20, Theorem 4] also implies that testing whether G is a *proper* T-graph

[1] H_{sub} is obtained from H by iteratively replacing an edge $\{u, v\}$ by a path uwv, where w is a new vertex.

is NP-complete. In contrast to this, it is known that when T is fixed, testing whether a given graph is a T-graph can be done in polynomial-time [8], i.e., XP in the size of T; but it is not known whether the problem is FPT in the size of T. When going beyond trees the recognition problem becomes much harder. Namely, for each fixed non-cactus graph H, H-graph recognition is NP-complete [8]. However, for fixed H, it seems that only two cases of proper H-graph recognition have been studied: The proper interval graphs (proper K_2-graphs) [10, 11] and the proper circular-arc graphs (proper C-graphs, for any cycle C) [11] can each be recognized in linear-time.

Our Contribution. In our main result, we show that the recognition of proper T-graphs is fixed-parameter tractable (FPT) with respect to the size of T by proving the following.

▶ **Theorem 1.** *There is an algorithm that, given an n-vertex graph G and a tree T with t nodes, decides whether G is a proper T-graph, and if yes, outputs a proper T-representation, in $2^{\mathcal{O}(t^2 \log t)} \cdot n^3$ time.*

To obtain our FPT algorithm for proper T-graph recognition, we first observe that the problem can be reduced to the case when the input graph G is connected and chordal. We proceed in the following three key steps.

In Section 3, we introduce compact representations which are an analog to the clique-trees of chordal graphs that incorporates the properness condition. We characterize the proper T-graphs via these compact representations. This allows us to work with maximal cliques of the input graph that can be listed in linear-time due to the chordalilty of G.

In Subsections 4.1 and 4.2, independent of the tree T, we partition the maximal cliques into a collection of the so-called *chains* each one necessarily forming a path in any proper T-representation, and the remaining singleton cliques that are marked and treated separately. We show that having a compact T-representation means there are, in terms of the size of T, at most quadratically many of these marked cliques and chains altogether.

In Subsections 4.3 and 4.4, we combine these ideas to form our FPT algorithm for proper T-graph recognition. First, our algorithm guesses a layout of the chains and the marked maximal cliques. The remaining non-trivial task is to decide whether there is a compact representation corresponding to the guessed layout. We select a root of the tree and show a combinatorial result that if any compact representation realizes some layout, it can be assumed to have some special properties concerning the usage of the nodes of degree at least three of the tree by the models with respect to the root. We call representations satisfying these properties *normalized*. Our algorithm follows the layout bottom-up and constructs a normalized representation if it exists.

We complement our algorithmic result from Theorem 1 by proving that if H is not constrained to be a tree, the recognition problem for proper H-graphs becomes NP-complete even if H has bounded size. This negative result employs a reduction quite similar to the one used for (non-proper) H-graphs in [8], and as such is discussed and proven in the full version.

▶ **Theorem 2** ($\star$). *There is a 4-vertex, 5-edge multigraph $\mathfrak{D}$ (defined by $V(\mathfrak{D}) = \{a, b, c, d\}$ and $E(\mathfrak{D}) = \{ab, bc, bc, bc, cd\}$) such that proper $\mathfrak{D}$-graph recognition is NP-complete.*

Note that this and further statements proven in the full version are marked with ($\star$).

2 Preliminaries

General Notation. We consider undirected graphs G with vertex set $V(G)$ and edge set $E(G)$. Usually we denote an edge as a set $\{u, v\}$. However, when needed, we also denote an edge an ordered pair (u, v). For any subset W of $V(G)$, we use $N(W)$ to denote the *open neighborhood* of W, i.e., $N(W) := \{u \in V(G) \setminus W \mid \{u, w\} \in E(G), w \in W\}$, and for a single vertex $w \in V(G)$, $N(w) := N(\{w\})$. We denote the set of maximal cliques of a graph as $\mathcal{C}(G)$. The shorthand $[n]$ denotes the set $\{1, \ldots, n\}$ of integers.

A *subdivision* H' of a graph H at an edge $\{u, w\}$ is the graph resulting from replacing edge $\{u, w\}$ with a path u, v, w where v is new vertex. A *contraction* of a graph H at an edge $\{u, w\}$ is the graph resulting from removing edge $\{u, v\}$ and identifying the two vertices u and w. Then H_{sub} is a *re-subdivision* of H if it can be obtained by a series of contractions of H (possibly none) followed by a series of subdivisions. In particular a graph G is a (proper) H-graph if and only if there is a re-subdivision that (properly) represents G.

Let T be a tree. For any pair x, y of nodes of T, we denote by $T[x, y]$ the set of nodes of the unique path from x to y in T. Note that $T[x, y] = T[y, x]$. We similarly define $T(x, y] := T[x, y] \setminus \{x\}$ and $T(x, y) := T[x, y] \setminus \{x, y\}$. A tuple of nodes $(x_1, \ldots, x_s)$ is *T-ordered* if there exists a path in the graph T from x_1 to x_s where the nodes $x_1, \ldots, x_s$ occur in this order, i.e., $T[x_1, x_s]$ is the path $x_1, \ldots, x_s$.

***H*-graphs.** Consider a re-subdivision H_{sub} of a graph H that (properly) represents a graph G using models $\{M_v \subseteq V(H_{\text{sub}}) \mid v \in V(G)\}$. For clarity, we refer to each $x \in V(H_{\text{sub}})$ as a *node* and to each $v \in V(G)$ as a *vertex*. We further refer to each node $x \in V(H_{\text{sub}})$ as:

- a *subdivision node* when it has degree two,
- a *branching node* when it has degree more than two, and
- a *leaf node* if it has degree one.

For a set of nodes $X \subseteq V(H_{\text{sub}})$, let $V_X := \{v \in V(G) \mid M_v \cap X \neq \emptyset\}$. When $X = \{x\}$, we also write V_x to mean $V_{\{x\}}$. For a subset of vertices $\Gamma \subseteq V(G)$, let $M_\Gamma := \bigcup_{v \in \Gamma} M_v$. We say that a set Γ of vertices (or nodes) is *connected* if the graph induced by Γ is connected.

▶ **Observation 3.** *Let H_{sub} (properly) represent a graph G. For any connected subset Γ of $V(G)$, the model M_Γ of Γ is connected in H_{sub}.*

Chordal Graphs and Clique Trees. A graph is *chordal* when it does not contain an induced k-vertex cycle for any $k \geq 4$. The chordal graphs are well known to be characterized as the intersection graphs of subtrees of a tree, i.e., for every chordal graph G, there is a tree T that represents G (G is a T-graph) [6, 14, 25]. In fact, G is chordal if and only if there is a tree T with models $\{M_v \subseteq V(T) \mid v \in V(G)\}$ where, for each node $x \in V(T)$, V_x is a maximal clique of G and for every node $y \in V(T)$ with $y \neq x$, $V_y \neq V_x$ [6, 14, 25]. These special representations of G are called *clique trees*, and one can be constructed in linear-time [3, 13]. Note that chordal graphs have a simpler linear-time recognition algorithm [23]. Finally, every chordal graph G has at most n maximal cliques where $n = |V(G)|$ and the sum of the sizes of the maximal cliques of G is $O(n + m)$ [15]. In particular, the total size of a clique tree of G is $O(n + m)$. Clearly, the latter two properties of chordal graphs also apply to (proper) T-graphs independently of T, and we will use them implicitly throughout our discussions.

Each chordal graph G is also a proper T-graph for a tree T. Namely, if a tree T represents G via models $\{M_v \mid v \in V(G)\}$, any tree T' built from T as follows properly represents G: Extend each model M_v by a new node x_v and add $\{x, x_v\}$ to $E(T)$ for some $x \in M_v$.

3 Compact Representations of Proper T-Graphs

In this section we introduce an analogue of clique trees for proper T-graphs. Ideally, G being a proper T-graph would imply a clique tree with the topology of T representing G which satisfies properness; in other words: a re-subdivision T_{sub} of T with models satisfying properness (i.e., forbidding $M_u \subseteq M_v$ for every pair $u, v \in V(G)$) such that every node x represents a unique maximal clique V_x. However, a proper tree-representation of a graph G may use a lot of nodes just to ensure that the models M_u and M_v obey properness; which is already the case for K_2 and its interval representation. Fortunately we may guarantee that almost all nodes represent a unique maximal clique by relaxing the properness condition. Instead of forbidding containment, we require that when M_u intersects M_v, there is a *place* where M_u may be extended (as needed) to break containment. That place is an edge $\{x, y\}$ in the tree T_{sub} where u *strongly escapes* v, that is, $u, v \in V_x$ and $v \notin V_y$. Actually, a weaker version of *escape* suffices. A vertex u *escapes* v if $u \in V_x$ and $v \notin V_y$.

▶ **Definition 4.** *Let a tree T_{sub} with models $\{M_u \mid u \in V(G)\}$ represent a connected graph G. We say that T_{sub} is a* compact representation of G *if*

(C1) *for every leaf node $x \in V(T_{\text{sub}})$, $V_x = \emptyset$,*

(C2) *there is a bijection between the non leaves of $V(T_{\text{sub}})$ and the maximal cliques $\mathcal{C}(G)$, and*

(C3) *for every ordered pair (u, v) with $u, v \in V(G)$, there is an edge $\{x, y\} \in E(T_{\text{sub}})$ where u escapes v.*

▶ **Observation 5 (⋆).** *Let a tree T_{sub} with models $\{M_u \mid u \in V(G)\}$ represent a connected graph G and satisfy condition (C1). For any vertices u, v of G, u and v satisfy the condition (C3) if and only if u and v satisfy condition*

(C3') *if $M_u \cap M_v \neq \emptyset$, then u strongly escapes v.*

Note that, the non-leaves of a compact representation are in one-to-one correspondence with the maximal cliques $\mathcal{C}(G)$. Namely, we identify the non-leaves with the maximal cliques, which implicitly defines the models. Thus, we often omit the explicit statement of the models.

▶ **Observation 6.** *Let G be a connected graph. For any compact representation T_{sub} of G,*

1. *for every distinct non-leaves $x, y \in V(T_{\text{sub}})$ there is a vertex $u \in V_x \setminus V_y$, and*
2. *for every edge $\{x, y\} \in E(T_{\text{sub}})$ of non-leaves x, y, there is a vertex $u \in V_x \cap V_y$.*

We (constructively) show that properness and compactness are essentially equivalent. To obtain compactness from properness, we carefully contract edges where a node was used solely to assure properness. This can involve contracting edges of T when the vertex sets of the nodes of an edge are comparable, e.g., if they are the same maximal clique. To obtain properness from compactness, we subdivide the tree and appropriately extend the models.

▶ **Theorem 7 (⋆).** *For any connected graph G and tree $T \neq K_1$, the graph G is a proper T-graph if and only if there is re-subdivision T_{sub} of T that is a compact representation of G.*

Thus, instead of finding a proper representation, we search for a compact representation. The actual "properness" is hidden in the condition (C3), and we may refer to this condition as *properness*. See also examples in Figure 1.

Our algorithm further relies on the following property of the models M_Γ of the (connected) components of $G - V_y$ for some non-leaf y; see also Figure 2(a). Let $\mathbf{\Gamma}(y)$ (w.r.t. graph G) be the vertex sets of the components of $G - V_y$. We note that $N(\Gamma) \subseteq V_y$ for every $\Gamma \in \mathbf{\Gamma}(y)$. Let a node y be an *eye* if it is a neighbor of a leaf or if it is a branching node.

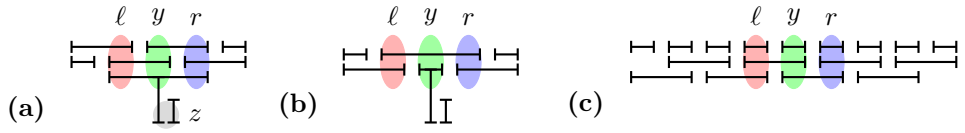


Figure 1 (a) A proper $K_{1,3}$-graph. Triple (ℓ, y, r) is surrounding. Any representation positions y between ℓ and r. Component $V_z \setminus V_\ell$ complies with condition (2B). Further, edge $\{z, y\}$ may be replaced by edge $\{z, \ell\}$ or $\{z, r\}$. (b) A proper $K_{1,3}$-graph. Triple (ℓ, y, r) is not surrounding. A "private" vertex in $V_y \setminus N(\Gamma_\ell) \cup N(\Gamma_r)$ contradicts condition (2A). Indeed, any of ℓ, y, r may realize the branching node. (c) Triple (ℓ, y, r) is surrounding. For $\{\Gamma_\ell, \Gamma_r\} = \mathbf{\Gamma}(y)$ condition (1) allows private vertices in V_y; otherwise, this proper interval graph would have no surrounded nodes.

▶ **Lemma 8** (⋆). *Let G be a connected graph. For any compact representation T_{sub} of G and any non-leaf node $y \in V(T_{\text{sub}})$,*

1. *$\{y\}$ and M_Γ for $\Gamma \in \mathbf{\Gamma}(y)$ partition the non-leaves of T_{sub}, and*
2. *each partition M_Γ contains an eye, hence $|\mathbf{\Gamma}(y)| \leq |V(T)|$.*

4 Finding a Compact Representation

In this section, we prove Theorem 1; namely, we establish our FPT algorithm. Throughout the discussion, we assume G is connected, and handle disconnected graphs within the final proof. From Section 3, it suffices to check for a compact representation T_{sub}. In Subsection 4.1, we establish the concept of *surrounded* nodes, which leads, in Subsection 4.2, to the *chains* that necessarily form paths in any compact tree representation. We establish that the chains (composed of surrounded nodes), and the remaining non-surrounded nodes are only quadratically many in the size of the desired tree T. In Subsection 4.3, we formalize the way these pieces fit together as *templates*. Finally, Subsection 4.4 contains the algorithm establishing Theorem 1. It proceeds by enumerating candidate templates and (non-trivially) testing whether a template admits a compact representation via a bottom-up procedure.

4.1 Surrounded Nodes

We establish conditions for arbitrary nodes ℓ, y, r that determines the relative position of ℓ, y, r in any representation T_{sub}, a relation which we denote as (ℓ, y, r) surrounding. Clearly, this positioning is unlikely to be possible for every triple (ℓ, y, r) since this would yield a polynomial-time algorithm. However, by carefully crafting our first two requirements, we may still relatively position almost all nodes ℓ, y, r. We only fail for a few nodes y, at most quadratic in the size of the host tree T, hence our parameter.

▶ **Definition 9.** *Consider non-leaves ℓ, y, r of T_{sub}. There is a component $\Gamma_\ell \in \mathbf{\Gamma}(y)$ containing $V_\ell \setminus V_y$, likewise a component $\Gamma_r \in \mathbf{\Gamma}(y)$ containing $V_r \setminus V_y$. Then (ℓ, y, r) is a* surrounding triple*, if the following conditions are met:*

(1) *If $\{\Gamma_\ell, \Gamma_r\} = \mathbf{\Gamma}(y)$, then $V_y = N(\Gamma_\ell) \cup N(\Gamma_r)$ or $N(\Gamma_\ell) \cap N(\Gamma_r) = \emptyset$;*

(2) *if $\{\Gamma_\ell, \Gamma_r\} \subsetneq \mathbf{\Gamma}(y)$,*

- ***(2A)*** *$V_y = N(\Gamma_\ell) \cup N(\Gamma_r)$, and*
- ***(2B)*** *for every $\Gamma \in \mathbf{\Gamma}(y) \setminus \{\Gamma_\ell, \Gamma_r\}$ we have: $N(\Gamma) \subseteq N(\Gamma_\ell) \cap N(\Gamma_r)$; and*

(3) *for every ℓ', r' that satisfy (1), (2A), and (2B) where $\Gamma_\ell = \Gamma_{\ell'}$ and $\Gamma_r = \Gamma_{r'}$, we have $V_{\ell'} \cap V_y \subseteq V_\ell \cap V_y$ and $V_{r'} \cap V_y \subseteq V_r \cap V_y$.*

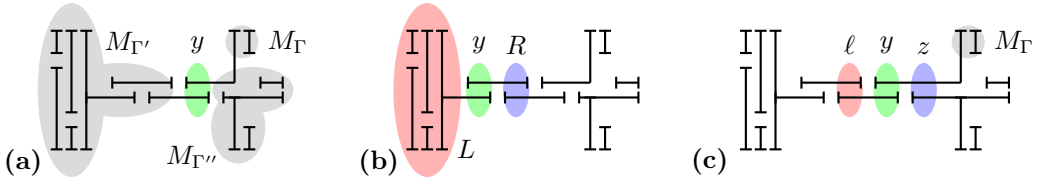


Figure 2 (a) Graph $G - V_y$ splits into three connected components. Their models partition the non-leaves of $T_{\text{sub}} - \{y\}$. Each model contains an eye (of this subdivision of a double-star). (b) Sets L, R exactly capture the nodes ℓ, r that (together with y) form a surrounding triple (ℓ, y, r). Here, y neighbors a subdivision node r to its right. In such a case a y-guard $R \supseteq \{r\}$ may only contain r. (c) Subdivision node y is not surrounded. A remote component Γ falsifies condition (2B). Note that $\mathbf{\Gamma}^{-1}(\Gamma) = \{y, z\}$. Thus Γ does not falsify (2B) for another subdivision node y'.

Note, that the definition does *not* depend on the considered representation T_{sub}. Importantly, $\Gamma_\ell \neq \Gamma_r$ is (implicitly) required by condition (1). We say that y *is surrounded*, if a triple (ℓ, y, r) is a surrounding triple for some nodes ℓ, r. See Figure 1 for examples.

For each node y, the connected components Γ_ℓ and Γ_r satisfy or falsify the first two conditions independently of the precise maximal cliques V_ℓ and V_r. However, condition (3) requires V_ℓ and V_r to be the closest ones to V_y. In many cases condition (3) implies that ℓ and r directly neighbor y. In fact, for a surrounded node y, there are sets of nodes L and R that exactly localize the nodes ℓ and r forming a surrounding triple with y. Formally, L, R are *y-guards*: A set of non-leaves $L \subseteq V(T_{\text{sub}})$ is a *y-guard* if $L \cup \{y\}$ is connected, and y is adjacent to a node $\ell \in L$ such that $\{\ell\} = L$ or ℓ is a branching node of T_{sub}; see Figure 2(b).

▶ **Lemma 10** (⋆). *Let a tree T_{sub} be a compact representation of a connected graph G. Let y be surrounded. There are distinct y-guards L and R such that (ℓ, y, r) is surrounding if and only if $(\ell, r) \in (L \times R) \cup (R \times L)$. Moreover there is an $\mathcal{O}(t^3 n^3)$-time algorithm that determines sets L, R for every surrounded node y; where $t = |V(T)|$ and $n = |V(G)|$.*

The *guards of y* are such distinct y-guards L and R that precisely characterize its surrounding triples. It is worth noting that a node y that is surrounded by subdivision nodes has singleton guards $\{\ell\}$ and $\{r\}$, which then must be neighbors y in any representation.

Surprisingly there is also a quadratic bound in $|V(T)|$ on the number of *not* surrounded nodes. To show this, the main difficulty is that the conditions (2A) and (2B) fail for a subdivision node y due to some *remote* component Γ, which is a connected component $\Gamma \in \mathbf{\Gamma}(y) \setminus \{\Gamma_\ell, \Gamma_r\}$. Here, let $y \in T_{\text{sub}}(x, z)$ for some neighbors $x, z \in V(T)$. To cope with these remote components, we use three ingredients:

- A component $\Gamma \in \mathbf{\Gamma}(y)$ that falsifies the conditions in question relates to $\mathbf{\Gamma}(x)$ or $\mathbf{\Gamma}(z)$: Its model M_Γ must be outside of $T_{\text{sub}}[x, z]$. By examining the nodes y' that have Γ as a component, it follows that either $\Gamma \in \mathbf{\Gamma}(x)$ or $\Gamma \in \mathbf{\Gamma}(z)$.
- We use Lemma 8 on components $\Gamma \in \mathbf{\Gamma}(x)$, likewise for $\mathbf{\Gamma}(z)$: The models of the components $\Gamma \in \mathbf{\Gamma}(x)$ partition the non-leaves T_{sub}, and each of its models M_Γ contains an eye. That means that $\mathbf{\Gamma}(x)$ contains at most $|E(T)|$ components.
- A component $\Gamma \in \mathbf{\Gamma}(x)$ can be remote for at most one node y on the path $T_{\text{sub}}(x, z)$, and hence falsify the surround conditions for at most one y on that path. This yields a simple $2|E(T_{\text{sub}})|$-bound for not surrounded nodes on that path; see also Figure 2(c).

By incorporating these ideas in a more careful manner we obtain the following bound:

▶ **Lemma 11** (⋆). *Let subdivision T_{sub} of a tree T be a compact representation of a connected graph G. There are at most $|E(T)|^2 + 1$ non-leaves of $V(T_{\text{sub}})$ that are not surrounded.*

4.2 Chains: Paths in any Representation

As observed previously, singleton guards $\{\ell\}$ and $\{r\}$ of a node y must neighbor y. If a path of nodes $y_1, \ldots, y_s$ is made of aligned guards, i.e., $\{y_{i-1}\}$ and $\{y_{i+1}\}$ are the guards of y_i, then it is a path in any representation T_{sub}. In this subsection we define such paths as *chains*. A *chain* captures a maximum length path $y_1, \ldots, y_s$ with aligned guards. They also include the initial and final guard Y_0 and Y_{s+1}, their *terminals*.

▶ **Definition 12.** *A* chain *is a maximal length $s \geq 1$ sequence of sets of non-leaf nodes*

$$\mathcal{Y} = \langle Y_0, \{y_1\}, \ldots, \{y_s\}, Y_{s+1} \rangle$$

where y_i has guards Y_{i-1} and Y_{i+1} for every $i \in [s]$; and where $Y_i \coloneqq \{y_i\}$ for $i \in [s]$.

To avoid lengthy statements, let us implicitly use $Y_i \coloneqq \{y_i\}$ from now on. Let $I(\mathcal{Y}) = \{y_1, \ldots, y_s\}$ be the set of inner nodes of a chain $\mathcal{Y}$. Let $\mathcal{H}(G)$ be the set of chains of a (connected) graph G.

By Lemma 10 such a chain implies that $y_1, \ldots, y_s$ is a path in any representation T_{sub}. Also there are unique realizations of the terminals $y_0 \in Y_0 \cap N_{T_{\text{sub}}}(y_1)$ and $y_{s+1} \in Y_{s+t} \cap N_{T_{\text{sub}}}(y_s)$ in a given T_{sub}. Let $y_0 y_1 \ldots y_s y_{s+1}$ be the *corresponding* path of $\mathcal{Y}$ in the tree T_{sub}.

The terminals define how the chains may attach to each other. In the very simple case a terminal Y_0 may only consist of a single non-surrounded node, and hence any other chain must attach to that node. Otherwise, as we show later, only the following option remains: Terminal Y_0 contains some surrounded node y_i' which is part of another chain $\langle Y_0', \ldots, Y_{s'+1}' \rangle$. Interestingly Y_0 then contains the whole path $y_1', \ldots, y_s'$. This allows us to freely change the attachment of the chain $\langle Y_0, \{y_1\}, \ldots \rangle$ to the chain $\langle \ldots, \{y_i'\}, \ldots \rangle$ without breaking the connectivity of the representation, though possibly the properness. Similarly, the intersection $V_x \cap V_{y_i}$ for an outside-of-path node x is equal for every $i \in [s]$, and therefore may be reattached in the same sense.

▶ **Lemma 13 (⋆).** *Consider a chain $\langle Y_0, \ldots, \{y_i\}, \ldots, Y_{s+1} \rangle$. A neighbor $x \in N(y_i) \setminus (Y_{i-1} \cup Y_{i+1})$ has $V_x \cap V_{y_i} = V_x \cap V_{y_j}$, for every $j \in [s]$. Furthermore, $Y' \cap I(\mathcal{Y}) \in \{\emptyset, I(\mathcal{Y})\}$ for every terminal Y' of any chain.*

In the following subsection we aim for a bound on the number of chains. We note here that chains behave in a reasonable way: Each surrounded node y is part of exactly one chain, because otherwise it contradicts the classification by y-guards as seen in Lemma 10. Clearly, a chain does not contain a node more than once, since y_i-guards Y_{i-1}, Y_{i+1} are in different subtrees of y_i, for every $i \in [s]$. As the next step, we observe that terminals only consist of either a not-surrounded node or a non-singleton guard, i.e., are from

- $\overline{\mathcal{S}}(G) \coloneqq \{\{y_0\} \mid y_0 \in \mathcal{C}(G) \text{ is not surrounded}\}$, or
- $\overline{\mathcal{U}}(G) \coloneqq \{Y_0 \mid Y_0 \text{ is guard of some } y_1 \in \mathcal{C}(G), |Y_0| > 1\}$.

▶ **Lemma 14 (⋆).** *Let T_{sub} be a compact representation of a connected graph G. Then every terminal Y_0, Y_{s+1} of a chain of G is part of $\overline{\mathcal{S}}(G)$ or $\overline{\mathcal{U}}(G)$.*

Further, let the family of inner nodes be $\mathcal{I}(G) \coloneqq \{I(\mathcal{Y}) \mid \mathcal{Y} \in \mathcal{H}(G)\}$. Note here that $\mathcal{I}(G) \cup \overline{\mathcal{S}}(G)$ partition the maximal cliques $\mathcal{C}(G)$.

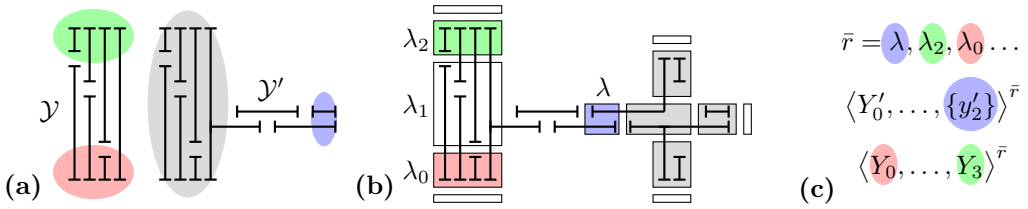


Figure 3 a) Depiction of the chains of almost the graph from Figure 2a) with marked terminals. b) A template prescribes the positioning of terminals, chains and $\overline{\mathcal{S}}$. Empty boxes represent leaves. The chain $\mathcal{Y}$ is mapped to path $\lambda_0\lambda_1\lambda_2$. The chain $\mathcal{Y}'$ is mapped to the single edge path $\lambda_1\lambda$ (an edge despite the gap in the picture). We have $t^0(\lambda_1) = I(\mathcal{Y})$, and indeed any attachment of $\mathcal{Y}'$ to an inner node of $\mathcal{Y}$ is possible. For the chain $\mathcal{Y}'$ a mapping to single edge suffices since here it is realized by a path of subdivision nodes. c) A sample root-ordering (colors mark the mapping of t^0), and resulting orientation of chains $\mathcal{Y}'$ and $\mathcal{Y}$. Here λ_2 (mapped to Y_3) is a tie-breaker for $\mathcal{Y}$.

4.3 Template: Fixing the Topology of Chains

The set of chains $\mathcal{H}(G)$ of a (connected) graph G already considerably prescribes many paths that are present in any proper representation T_{sub} of G. What remains are two problems of a more global flavor: For a chain there may be a vast range of possible connections. Simultaneously we have to assure properness, i.e., that any vertices u and v escape each other. To cope with these tasks we define a preliminary representation, a *template*. A template considerably fixes the topology of a tree T_{sub} representing G. It narrows down the possible representations such that we can focus on the properness. At the same time, our final algorithm has to guess a template, thus its possibilities should be bounded by our parameter, the size of T.

To fix the relative positions of chains, a template locates the terminals of a chain, Y_0 and Y_{s+1}, on some template tree T^0. A concrete realization T_{sub} of that template is a subdivision of T^0. It realizes a chain between its terminals as prescribed by the template. More precisely, t^0 maps the nodes λ of T^0 to the terminals of chains. To avoid ambiguity, let $t^0(\lambda)$ not map to a mere terminal Y_0, if $Y_0 \in \overline{\mathcal{U}}(G)$ (as it may be huge), but narrow down the mapping to some set of inner nodes of $\mathcal{I}(G)$. In other words we fix the neighborhood of a chain on the "chain-level". Note that any $Y_0 \in \overline{\mathcal{U}}(G)$ is a superset of some set of inner nodes, as seen in Lemma 13. For convenience, let us also fix a mapping h^0 of chains $\langle Y_0, \{y_1\}, \dots, \{y_s\}, Y_{s+1} \rangle$ onto T^0: Let h^0 map to paths $\lambda_0, \dots, \lambda_{s'+1}$ in T^0, which should be conforming with the terminals, which is $t^0(\lambda_0) \subseteq Y_0$ and $t^0(\lambda_{s'+1}) \subseteq Y_{s+1}$. If the chain does not contain any branching node, it suffices to represent the terminals. Then h^0 maps simply to the single-edge path $\lambda_0, \lambda_{s'+1}$ (for example $\mathcal{Y}'$ in Figure 3(a),(b)). In the other extreme, every inner node may be a branching node (respectively used as a terminal of another chain), thus possibly $s' = s$.

In more detail, a chain $\mathcal{Y}$ may correspond to a path $y_0 \dots y_{s+1}$ with an inner branching node y'_0 which is the endpoint of a path $y'_0 y'_1 \dots$ corresponding to another chain $\mathcal{Y}' = \langle Y'_0, \dots, \rangle$. Thus, the chain $\mathcal{Y}$ must be mapped to a path with an inner node λ_j that is an endpoint of the path $\lambda_j, \lambda'_1 \dots, \lambda'_{s''+1}$ that is the image of the other chain $\mathcal{Y}'$ (for example $\mathcal{Y}$ in Figure 3a)b)). As seen before, then $I(\mathcal{Y}) \subseteq Y'_0 \in \overline{\mathcal{U}}(G)$. Hence, the mapping of that terminal is $t^0(\lambda_j) = I(\mathcal{Y})$. We require this behavior for inner nodes like λ_j.

▶ **Definition 15.** *Let G be a connected chordal graph. A* template *of a tree T (w.r.t. G) is a triple (T^0, t^0, h^0) where*

- *T^0 is a re-subdivision of T,*
- *t^0 is a mapping of the non-leaves of T^0 to $\overline{\mathcal{S}}(G) \cup \mathcal{I}(G)$,*
- *h^0 is a bijection of the chains $\mathcal{H}(G)$ to an edge-disjoint set of non-trivial (i.e., containing at least one edge) paths between non-leaves of T^0, and*
- *for every chain $\langle Y_0, \dots, Y_{s+1} \rangle \in \mathcal{H}(G)$ mapped to a path $\lambda_0, \dots, \lambda_{s'+1}$ we have that $t^0(\lambda_0) \subseteq Y_0$ and $t^0(\lambda_{s'+1}) \subseteq Y_{s+1}$ and $t^0(\lambda_i) = I(\mathcal{Y})$ for every $i \in [s']$.*

Consider a tree T_{sub} where each non-leaf y is identified with a maximal clique V_y. Then T_{sub} realizes *the template (T^0, t^0, h^0) if T_{sub} results from subdividing T^0 and $V_\lambda \in t^0(\lambda)$ for every non-leaf λ of T^0.*

Notably the image of h^0 does not necessarily cover every edge between non-leaves. Namely, non-surrounded nodes y and y' might be neighbors. The next lemma establishes that every tree T_{sub} that is a compact representation realizes some template, as we intended.

▶ **Lemma 16 (⋆).** *If T_{sub} is a re-subdivision of a tree T and a compact representation of a connected graph G, then T_{sub} realizes some template (T^0, t^0, h^0) of T.*

As formalized in the next lemma, we can enumerate the possible templates in FPT, since the number of chains is quadratically bounded in $V(T)$.

▶ **Lemma 17 (⋆).** *There are $2^{\mathcal{O}(t^2 \log t)}$ possible templates of a tree T w.r.t. a connected chordal graph G, which can be enumerated in time $2^{\mathcal{O}(t^2 \log t)} \cdot n^3$; where $t = |V(T)|$, $n = |V(G)|$.*

4.4 Normalized Representation: Achieving Properness

We now consider a fixed template and focus on the properness. The remaining leeway is to locally change the branching nodes of a particular chain. We use a construction which only fails if the considered template does not allow a compact representation. The result is a *normalized* representation.

Any representation T_{sub} can be normalized by a bottom-up process: Move each branching node y_i up as much as possible within the local subtree, i.e. as long as the subtree remains compact. By moving up, we mean replacing y_i by y_j as a branching node that is closer to a global root in the chain. The set of nodes that potentially replace y_i behave in a linear fashion, and hence allow this greedy approach. Thus we may assume that a normalized representation exists for a yes-instance. Our algorithm though has to construct a representation from scratch. By incorporating this idea in a more careful manner we may assemble each subtree of a normalized T_{sub} bottom-up. Here we attach the inductive subtrees in the most conservative way; then the same normalization step as before yields the desired new subtree. Again, the linear behavior of the potential replacements enable this greedy approach.

To start, let us define the root of a template. Since chains may not "align" towards a picked root $\bar{r}_1$, we have to work with additional tie-breakers $\bar{r}_2, \bar{r}_3, \dots$.

▶ **Definition 18.** *A* root-ordering *$\bar{r}$ is an ordering $\bar{r}_1, \bar{r}_2, \dots$ of nodes $V(T^0) \cap \{\lambda \mid t^0(\lambda) \in \overline{\mathcal{S}}\}$.*

The specific root-ordering will not be of importance and we may pick one arbitrarily. We assume in the following that every tree and template comes with a root-ordering. See Figure 3 for an example.

▶ **Definition 19.** *A root-ordering $\bar{r}$ and a template (T^0, t^0, h^0) define an orientation for every chain $\mathcal{Y} = \langle Y_0, \dots, Y_{s+1} \rangle$ as follows. Let $h^0(\mathcal{Y})$ map to a path in T^0 with end nodes λ_0 and λ_{s+1} where $t^0(\lambda_0) \subseteq Y_0$ and $t^0(\lambda_{s+1}) \subseteq Y_{s+1}$. Let k be the smallest index such that $(\lambda_0, \lambda_{s+1}, \bar{r}_k)$ or $(\bar{r}_k, \lambda_0, \lambda_{s+1})$ is T^0-ordered. If $(\lambda_0, \lambda_{s+1}, \bar{r}_k)$ is T^0-ordered, then $\mathcal{Y}$* is oriented towards *Y_{s+1}, which we denote by writing $\langle Y_0, \dots, Y_{s+1} \rangle^{\bar{r}}$.*

Note that the index k always exists, since every neighbor of a leaf is not surrounded.

Let $R[y_i, y_j]T_{\text{sub}}$ be the tree resulting from replacing branching node y_i by y_j. Its local version is $\rho[y_i, y_j]T_{\text{sub}}$. The models living in the more restrict subtree $\rho^{\downarrow}[y_i, y_j]T_{\text{sub}}$ are critical: Their properness is at stake. We define the possible replacements of a node y_i resulting in a proper representation as the potential $\Phi(T_{\text{sub}}, y_i)$.

▶ **Definition 20.** *Let $\bar{r}$ be a root-ordering. Consider a branching node $y_i \in V(T_{\text{sub}})$ and its chain $\langle Y_0, \dots, \{y_i\}, \dots, \{y_j\}, \dots, Y_{s+1}\rangle^{\bar{r}}$ where y_0 realizes Y_0. For integers $i \leq j < s$, let*

- *$R[y_i, y_j]T_{\text{sub}}$ be the tree T_{sub} where y_i replaces y_j as a branching node, i.e., edge $\{y_i, z\}$ is replaced by a new edge $\{y_j, z\}$, for every node $z \in N_{T_{\text{sub}}}(y_i) \setminus (Y_{i-1} \cup Y_{y+1})$;*
- *$\rho[y_i, y_j]T_{\text{sub}}$ be the tree consisting of the subtree of $R[y_i, y_j]T_{\text{sub}}$ rooted at y_0 (w.r.t. global root $\bar{r}_1$) and path $y_0, \dots, y_s$ where for every chain node $y_{i'} \in \{y_1, \dots, y_s\}$ and non-chain neighbor $z' \in N_{R[y_i,y_j]T_{\text{sub}}} \setminus (Y_{i'-1} \cap Y_{i'+1})$ a new leaf node $y'_{i'}$ added adjacent to $y_{i'}$;*
- *$\rho^{\downarrow}[y_i, y_j]T_{\text{sub}}$ be the tree consisting of the subtree of $R[y_i, y_j]T_{\text{sub}}$ rooted at y_0 (w.r.t. global root $\bar{r}_1$) and path $y_0, \dots, y_{j-1}$.*

For convenience, let $\rho^{\downarrow}[y_i]T_{\text{sub}} \coloneqq \rho^{\downarrow}[y_i, y_i]T_{\text{sub}}$ as well as $\rho[y_i]T_{\text{sub}} \coloneqq \rho[y_i, y_i]T_{\text{sub}}$.

▶ **Definition 21.** *We define the* potential $\Phi(T_{\text{sub}}, y_i)$ *(w.r.t. a template (T^0, t^0, h^0) and root-ordering $\bar{r}$) of non-leaf node y_i.*

- *For a not-surrounded node y_i, let $\Phi(T_{\text{sub}}, y_i) = \{y_i\}$.*
- *For a surrounded branching node y_i, consider its chain $\langle \cdot, \dots, \{y_i\}, \dots, \{y_s\}, \cdot\rangle^{\bar{r}}$. The potential $\Phi(T_{\text{sub}}, y_i)$ contains every node $y_j \in \{y_i, \dots, y_s\}$ where the tree $R[y_i, y_j]T_{\text{sub}}$ is such that every vertex u with model $M_u \subseteq V(\rho^{\downarrow}[y_i, y_j]T_{\text{sub}})$ escapes every other vertex v.*

A simple example is that $y_i \in \Phi(T_{\text{sub}}, y_i)$ for compact representations T_{sub}, as the considered replacement does nothing. In contrast, $\Phi(T_{\text{sub}}, y_i) = \emptyset$ indicates non-properness for the subtree of y_{i-1}. Indeed, the potential of y_i captures exactly the possible replacements of y_i as a branching node.

If some replacement y_j of y_i already is a branching node, the topology changes and $R[y_i, y_j]T_{\text{sub}}$ does not realize the same template. To avoid such issues, we require (without loss of generality) a minimal representation: A tree T_{sub} is *minimal* if there is no compact representation T'_{sub} of G that is a re-subdivision of T_{sub} with fewer branching nodes. Clearly, if there is a representation T_{sub} of G, we may also assume that it is minimal. In particular, the contraction would result in different candidate re-subdivision of T, which we consider separately.

We may compute it locally, meaning it suffices to consider the subtree $\rho[y_i]$. Since the potential $\Phi(T_{\text{sub}}, y_i)$ is a connected subsequence of $\langle y_i, \dots, y_s\rangle$, we either view it as a set or as such a subsequence. Further, if the potential is $\langle y_i, \dots, y_j\rangle$, then replacing y_i with the last node y_j makes the resulting potential at y_j singleton. Finally, the potential is independent from later replacements, assuming a bottom-up (i.e., leaf-to-root) procedure.

▶ **Lemma 22 (⋆).** *Let T_{sub} be a minimal compact representation of a connected graph G. We observe the following for a chain $\langle \cdot, \dots, \{y_i\}, \dots, \{y_j\}, \dots, \{y_s\}, \cdot\rangle^{\bar{r}}$ for $i \leq j \leq s$:*

1. *If $y_j \in \Phi(T_{\text{sub}}, y_j)$, then $R[y_i, y_j]T_{\text{sub}}$ is a minimal compact representation of G.*
2. *locality, $\Phi(T_{\text{sub}}, y_i) = \Phi(\rho[y_i]T_{\text{sub}}, y_i)$,*
3. *connectivity, $\Phi(T_{\text{sub}}, y_i)$ is connected in T_{sub}, and hence some subsequence $\langle y_i, \dots, y_j\rangle$,*
4. *linearity, $\Phi(T_{\text{sub}}, y_i) = \langle y_i, \dots, y_j\rangle$ if and only if $\Phi(R[y_i, y_j]T_{\text{sub}}, y_j) = \langle y_j\rangle$.*
5. *independence, $\Phi(R[y_i, y_j]T_{\text{sub}}, x) \subseteq \Phi(T_{\text{sub}}, x)$ for every node $x \in V(T_{\text{sub}})$ where $(x, y_i, \bar{r}_1)$ is T_{sub}-ordered.*

Consider a tree T_{sub} that realizes a template (T^0, t^0, h^0), and has some root-ordering $\bar{r}$. We say T_{sub} is *normalized* for a node y (w.r.t. to (T^0, t^0, h^0) and $\bar{r}$) if $\Phi(T_{\text{sub}}, y) = \langle y \rangle$. By the locality property, this is equivalent to $\Phi(\rho[y]T_{\text{sub}}, y) = \langle y \rangle$, hence it suffices to consider the local subtree. The whole tree T_{sub} is normalized if it is normalized for every branching node. Now the independence of the potential as explored earlier allows normalizing any representation by a bottom-up procedure. Thus, in a yes-instance, we may assume a normalized representation.

▶ **Lemma 23** (⋆). *There is an $\mathcal{O}(n^3)$ time algorithm that, given a connected chordal n-vertex graph G and a template (T^0, t^0, h^0), decides whether there is a minimal compact representation of G that realizes (T^0, t^0, h^0), and if one exists, it outputs one that is also normalized.*

Proof (Sketch). Assuming a yes-instance, there is minimal compact representation T'_{sub} of G that realizes template (T^0, t^0, h^0). We may also assume that T'_{sub} is normalized (proven in the full version). Our algorithm outputs a representation isomorphic to T'_{sub}, thus a normalized one as desired. If, however, our construction fails at some point, we correctly conclude that no such representation exists. In the rest of the proof we fix an arbitrary root-ordering $\bar{r}$.

We fix an ordering $\sigma = \lambda_1, \lambda_2, \dots$ of the non-leaf nodes of the template tree T^0, which follows the ordering within in a chain and otherwise is bottom-up. Pick a node λ_k where every non-leaf child of λ_k has been added before, and append it to the ordering. If there is a chain $\langle Y_0, \dots, Y_{s'+1} \rangle^{\bar{r}}$ mapped by h^0 to a path of form $\lambda_0, \lambda_k, \lambda_{k,1}, \dots, \lambda_{k,s'+1}$, append nodes $\lambda_{k,1}, \dots, \lambda_{k,s'+1}$ as well. Then continue to picking a new node until all nodes are ordered.

For $k \geq 1$, let T_k be the subtree of T'_{sub} induced by $\lambda_1, \dots, \lambda_k$, every subdivision node between nodes from $\lambda_1, \dots, \lambda_k$ and leaves neighboring $\lambda_1, \dots, \lambda_k$. By induction over $k \geq 1$, we prove that a tree isomorphic to T_k is polynomial time computable given G, $(T^0, t^0, h^0,)$ and $\bar{r}$. Eventually this yields to a representation T_{sub} isomorphic to T'_{sub}, thus normalized minimal compact and realizing (T^0, t^0, h^0), as desired.

(Induction base, when λ_k neighbors a leaf (w.r.t. to root $\bar{r}_1$)) The node λ_k represents a not-surrounded node $t^0(\lambda_k) = \{\lambda_k\} \in \overline{\mathcal{S}}(G)$. Then T_k consists only of λ_k adjacent to some leaf. Thus, this tree is prescribed by (T^0, t^0, h^0) and hence no computation is required. The induction step where $\lambda_k \in V(T^0)$ is a node with $t^0(\lambda_k) = \{\lambda_k\} \in \overline{\mathcal{S}}(G)$ is similar, and omitted here.

(Induction step $\mathcal{I}(G)$) We consider the case where the template node λ_k is a surrounded branching node. This means that $t^0(\lambda_k) = \{y_1, \dots, y_s\} = I(\mathcal{Y})$ for some chain $\mathcal{Y}$. The template maps $\mathcal{Y}$ to a non-trivial path $\lambda_0, \dots, \lambda_{s'+1}$ in T^0 containing λ_k:

$$\lambda_0 \dots \lambda_{c_0} \lambda_k \lambda_{c'_0} \dots \lambda_{s'+1} \;=\; h^0\big(\langle Y_0, \{y_1\} \dots, \{y_i\}, \dots, \{y_s\}, Y_{s+1} \rangle^{\bar{r}}\big).$$

For each of those inner template nodes $\lambda_{i'}$, we have $t^0(\lambda_{i'}) = \{y_1, \dots, y_s\}$. Let us assume that $t^0(\lambda_0) \subseteq Y_0$ such that the directions of increasing indices match.

Note that λ_{c_0} is ordered before λ_k because of how the ordering σ is defined. Our algorithm may determine λ_{c_0} as the child in T^0 where $(\lambda_0, \lambda_{c_0}, \lambda_k, \lambda_{s'+1})$ is T^0-ordered (possibly $\lambda_0 = \lambda_{c_0}$). The tree T_k realizes λ_k with some inner node y_j with $j \in [s]$. Our task is to determine j without knowing T_k. Let $\lambda_{c_1}, \dots, \lambda_{c_z}$ be the (possibly non-existent, possibly containing $\lambda_{c'_0}$) remaining children of λ_k in T^0. By the induction hypothesis, the subtrees $T_{c_0}, T_{c_1}, \dots, T_{c_z}$ are polynomial time computable.

The tree T_{c_0} realizes λ_{c_0} with some node y_{i-1} for $i \in \{2, \dots, s\}$ where $y_0 \in Y_0$. Because T_{c_0} is a subtree of T_k, this limits the possible realizations of y_j to $\{y_i, \dots, y_s\}$. Let $\overrightarrow{c_0}$ be the path $(y_{i-1}, y_i, \dots, y_s)$.

Consider the adjacency $\lambda_{c_1}\lambda_k$. A simple case is that $t^0(\lambda_{c_1}) = \{\lambda_{c_1}\} \in \overline{\mathcal{S}}(G)$. Then in the tree T_k the two realizing nodes λ_{c_1} and λ_k must be adjacent, and we define $\overrightarrow{c_1}(\lambda_k)$ to be the path $\lambda_{c_1}\lambda_k$. Note that we define the path with a variable λ_k (as named to coincide with the template node since it shares the same variability). For example $\overrightarrow{c_1}(y_j)$ is the path contained in the tree T_k (which we aim to construct).

Otherwise, the node $t^0(\lambda_{c_1})$ is part of a chain with terminal $Y'_{s'+1}$ where $t^0(\lambda_k) \subseteq Y'_{s'+1}$. Now either λ_{c_1} is the other terminal of chain $\mathcal{Y}_1$, or it is the set of inner nodes $I(\mathcal{Y}_1)$ and hence realized as one of them. Thus the format of the chain is either

- $\langle Y'_0, \{y^1_{c_1}\}, \ldots, \{y^{s_1}_{c_1}\}, Y'_{s'+1}\rangle^{\bar{r}}$ where $t^0(\lambda_{c_1}) \subseteq Y'_0$, or
- $\langle \cdot, \ldots, \{y_{c_1}\}, \{y^1_{c_1}\}, \ldots, \{y^{s_1}_{c_1}\}, Y'_{s'+1}\rangle^{\bar{r}}$, where y_{c_1} is the realization of λ_{c_1} in the tree T_{c_1}.

Let $\overrightarrow{c_1}(\lambda_k)$ be the path $(\lambda_{c_1}, y^1_{c_1}, \ldots, y^{s_1}_{c_1}, \lambda_k)$, similarly as before with variable λ_k. For example $\overrightarrow{c_1}(y_j)$ is the path contained in the unknown tree T_k. We define the paths $\overrightarrow{c_2}(\lambda_k), \ldots, \overrightarrow{c_z}(\lambda_k)$ for the other children analogously. Clearly, the same observations apply.

We define the tree $T(\lambda_k)$ similarly. Namely, $T(\lambda_k)$ is the tree containing the subtrees $T_{c_0}, T_{c_1}, \ldots, T_{c_z}$ together with paths $\overrightarrow{c_1}(\lambda_k), \ldots, \overrightarrow{c_z}(\lambda_k)$ and path $y_{i-1}, y_i, \ldots, y_s$. Then T_k is the subtree of $T(y_j)$ rooted at y_j. Thus it remains to determine y_j without knowing T_k.

For that purpose, consider the tree $T(y_i)$, the tree with the most conservative realization of λ_k. Applying the rehang operation yields $R[y_i, y_j]T(y_i) = T(y_j)$. Assume that node y_k of all the nodes of T_k has the smallest distance to the global root $\bar{r}$ (the general case is handled by a slight modification to $T(y_i)$, see full version). Then, since T'_{sub} is normalized and because of locality, we have $\langle y_j\rangle = \Phi(T'_{\text{sub}}, y_j) = \Phi(\rho[y_i]T'_{\text{sub}}, y_j) = \Phi(\rho[y_i]T(y_j), y_j) = \Phi(T(y_j), y_j)$.

Then by the linearity of the potential we have that $\Phi(T(y_i), y_i) = \langle y_i, \ldots, y_j\rangle$. This is how we algorithmically determine y_j, assuming a yes-instance. Thus the desired tree $T_k(y_j)$ is polynomial time computable given graph G, template (T^0, t^0, h^0) and $\bar{r}$. If our algorithm observes that $\Phi(T(y_i), y_i) = \emptyset$ at some point, it contradicts the existence of a normalized representation T'_{sub}, and our algorithm returns no. ◀

Now we outline our FPT algorithm for the parameter $t = |V(T)|$. We assume without loss of generality that G is a chordal graph and $T \neq K_1$ as the problem is trivial otherwise. Note that chordality can be tested in linear time [23]. If G is not connected, each proper interval graph component always be represented using a subdivision of an edge incident to a leaf of T. Thus, these components, which can be recognized in linear time [10, 11], can be excluded from the further consideration. Each of the remaining components is not a proper interval graph and, as such, contains a vertex whose model includes a branching node of T. Thus, if these components number more than the number of branching nodes, G has no T-representation. Assume that this is not the case. We guess an assignment of the connected components of G to connected subtrees of T representing them. Two such subtrees may share an edge (which can be needed to represent both components of G using the end-nodes of this shared edge). Note that are at most $2^{\mathcal{O}(t \log t)}$ possible mappings, and then we can deal with every component of G and the corresponding subtree of T separately. From now on, we assume that G is connected. By Theorem 7, we may look for a compact representation T_{sub}; further, it suffices that T_{sub} is minimal. Therefore, there is a template (T^0, t^0, h^0) that allows a representation of G as seen in Lemma 16. We compute the chains of G and try every template in time $2^{\mathcal{O}(t^2 \log t)} \cdot n^3$ where $n = |V(G)|$, as seen in Lemma 17. Pick an arbitrary root-ordering $\bar{r}$. Then test in polynomial time whether a minimal compact representation of G realizing this template by using Lemma 23. In a positive case, applying Theorem 7 leads to a proper representation. This implies our main result (restated here).

▶ **Theorem 1.** *There is an algorithm that, given an n-vertex graph G and a tree T with t nodes, decides whether G is a proper T-graph, and if yes, outputs a proper T-representation, in $2^{\mathcal{O}(t^2 \log t)} \cdot n^3$ time.*

5 Concluding Remarks and Open Problems

Our recognition algorithm for proper tree-graphs provides the following side result on proper leafage (introduced by Lin et al. [21] analogously to leafage): The *proper leafage* $\ell^\star$ of a chordal graph G is the minimum number of leaf nodes of all trees T that properly represent G. The side result, as in Corollary 24, is that computing the proper leafage is FPT. For the decision version, if G is not a proper interval graph, we simply guess the host tree T_{sub} of minimal leafage and verify properness with our algorithm from Theorem 1. Of course, it still remains open whether computing proper leafage is NP-hard.

▶ **Corollary 24.** *Computing the proper leafage $\ell^\star$ of a chordal graph G is FPT w.r.t. $\ell^\star$.*

While we have shown that proper T-graph recognition is FPT, it remains open whether non-proper T-graph recognition is FPT. Perhaps most importantly, gaps remain concerning the precise conditions under which (proper) H-graph recognition is NP-complete for fixed H.

References

1 Deniz Ağaoğlu and Petr Hliněný. Isomorphism problem for S_d-graphs, 2019. arXiv:1907.01495.

2 Miklós Biró, Mihály Hujter, and Zsolt Tuza. Precoloring extension. I. Interval graphs. *Discrete Mathematics*, 100(1):267–279, 1992.

3 Jean R. S. Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In *Graph Theory and Sparse Matrix Computation*, pages 1–29, New York, NY, 1993. Springer New York.

4 M. L. Brady and M. Sarrafzadeh. Stretching a knock-knee layout for multilayer wiring. *IEEE Transactions on Computers*, 39(1):148–151, January 1990. doi:10.1109/12.46293.

5 Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999. doi:10.1137/1.9780898719796.

6 Peter Buneman. A characterisation of rigid circuit graphs. *Discrete Mathematics*, 9(3):205–212, 1974. doi:10.1016/0012-365X(74)90002-8.

7 Steven Chaplick, Fedor V. Fomin, Petr A. Golovach, Dusan Knop, and Peter Zeman. Kernelization of graph hamiltonicity: Proper H-graphs. In Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R. Salavatipour, editors, *WADS 2019*, volume 11646 of *LNCS*, pages 296–310. Springer, 2019. doi:10.1007/978-3-030-24766-9_22.

8 Steven Chaplick, Martin Toepfer, Jan Voborník, and Peter Zeman. On H-topological intersection graphs. In *WG 2017*, pages 167–179, 2017. doi:10.1007/978-3-319-68705-6_13.

9 Steven Chaplick and Peter Zeman. Combinatorial problems on H-graphs. In *EUROCOMB'17*, volume 61 of *ENDM*, pages 223–229. Elsevier, 2017. doi:10.1016/j.endm.2017.06.042.

10 Derek G. Corneil. A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs. *Discrete Applied Mathematics*, 138(3):371–379, 2004. doi:10.1016/j.dam.2003.07.001.

11 Xiaotie Deng, Pavol Hell, and Jing Huang. Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. *SIAM Journal on Computing*, 25(2):390–403, 1996. doi:10.1137/S0097539792269095.

12 Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on H-graphs. *Algorithmica*, 2020. doi:10.1007/s00453-020-00692-9.

13 Philippe Galinier, Michel Habib, and Christophe Paul. Chordal graphs and their clique graphs. In *WG 1995*, volume 1017 of *LNCS*, pages 358–371. Springer, 1995. doi:10.1007/3-540-60618-1_88.

14 Fănică Gavril. The intersection graphs of subtrees of trees are exactly the chordal graphs. *Journal of Combinatorial Theory Series B*, 16:47–56, 1974.

15 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57 of *Annals of Discrete Mathematics*. Elsevier Science B.V., Amsterdam, second edition, 2004. With a foreword by Claude Berge.

16 M. L. Huson and A. Sen. Broadcast scheduling algorithms for radio networks. In *Military Communications Conference, 1995. MILCOM '95, Conference Record, IEEE*, volume 2, pages 647–651 vol.2, November 1995. doi:10.1109/MILCOM.1995.483546.

17 Lars Jaffke, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. Mim-width III. graph powers and generalized distance domination problems. *Theoretical Computer Science*, 796:216–236, 2019. doi:10.1016/j.tcs.2019.09.012.

18 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width II. the feedback vertex set problem. *Algorithmica*, 82(1):118–145, 2020. doi:10.1007/s00453-019-00607-3.

19 Deborah Joseph, Joao Meidanis, and Prasoon Tiwari. Determining DNA sequence similarity using maximum independent set algorithms for interval graphs. In Otto Nurmi and Esko Ukkonen, editors, *SWAT '92*, volume 621 of *LNCS*, pages 326–337. Springer, 1992. doi:10.1007/3-540-55706-7_29.

20 Pavel Klavík, Jan Kratochvíl, Yota Otachi, and Toshiki Saitoh. Extending partial representations of subclasses of chordal graphs. *Theoretical Computer Science*, 576:85–101, 2015.

21 In-Jen Lin, Terry A. McKee, and Douglas B. West. The leafage of a chordal graph. *Discuss. Math. Graph Theory*, 18(1):23–48, 1998. doi:10.7151/dmgt.1061.

22 Fred S Roberts. *Graph theory and its applications to problems of society*. SIAM, 1978.

23 Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976. doi:10.1137/0205021.

24 F. W. Sinden. Topology of thin film rc circuits. *Bell System Technical Journal*, 45(9):1639–1662, 1966. doi:10.1002/j.1538-7305.1966.tb01713.x.

25 James R. Walter. Representations of chordal graphs as subtrees of a tree. *Journal of Graph Theory*, 2(3):265–267, 1978. doi:10.1002/jgt.3190020311.

New Algorithms for Mixed Dominating Set

Louis Dublois
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, Paris, France
louis.dublois@lamsade.dauphine.fr

Michael Lampis
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, Paris, France
michail.lampis@lamsade.dauphine.fr

Vangelis Th. Paschos
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, Paris, France
paschos@lamsade.dauphine.fr

Abstract

A mixed dominating set is a set of vertices and edges that dominates all vertices and edges of a graph. We study the complexity of exact and parameterized algorithms for MDS, resolving some open questions. In particular, we settle the problem's complexity parameterized by treewidth and pathwidth by giving an algorithm running in time $O^*(5^{tw})$ (improving the current best $O^*(6^{tw})$), and a lower bound showing that our algorithm cannot be improved under the SETH, even if parameterized by pathwidth (improving a lower bound of $O^*((2-\varepsilon)^{pw})$). Furthermore, by using a simple but so far overlooked observation on the structure of minimal solutions, we obtain branching algorithms which improve the best known FPT algorithm for this problem, from $O^*(4.172^k)$ to $O^*(3.510^k)$, and the best known exact algorithm, from $O^*(2^n)$ and exponential space, to $O^*(1.912^n)$ and polynomial space.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases FPT Algorithms, Exact Algorithms, Mixed Domination

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.9

1 Introduction

Domination problems in graphs are one of the most well-studied topics in theoretical computer science. In this paper we study a variant called Mixed Dominating Set: we are given a graph $G = (V, E)$ and are asked to select $D \subseteq V$ and $M \subseteq E$ such that $|D \cup M|$ is minimized and the set $D \cup M$ dominates $V \cup E$, where a vertex dominates itself, its neighbors, and its incident edges and an edge dominates itself, its endpoints, and all edges with which it shares an endpoint.

Mixed Dominating Set is a natural variation of domination in graphs as it can be seen as a *mix* between four standard problems: Dominating Set, where vertices dominate vertices; Edge Dominating Set, where edges dominate edges; Vertex Cover, where vertices dominate edges; and Edge Cover, where edges dominate vertices. In Mixed Dominating Set we are asked to select vertices *and* edges in a way that dominates all vertices *and* edges. As only the last of these four problems is in P, it is not surprising that Mixed Dominating Set is NP-hard. We are therefore motivated to study approximation, exponential-time and parameterized algorithms for this problem, and indeed this has been the topic of several recent papers. On the approximation algorithms side, the problem is well-understood: Hatami [9] gave a 2-approximation algorithm, while more recently Dudycz et al. [5] showed that (under the UGC) no algorithm can achieve a ratio better than 2 for Edge Dominating Set. As we explain (Proposition 1) this hardness result easily carries over to Mixed Dominating Set, thus essentially settling the problem's approximability. Hence, in this paper we focus on parameterized and exact algorithms.

Mixed Dominating Set has recently been the focus of several works in this context. With respect to the natural parameter (the size k of the solution), an $O^*(7.465^k)$[1] algorithm was given by Jain et al. [12], more recently improved to $O^*(4.172^k)$ by Xiao and Sheng [26]. With respect to treewidth and pathwidth, Jain et al. gave algorithms running in $O^*(6^{tw})$ time and $O^*(5^{pw})$ time, improving upon the $O^*(3^{tw^2})$ time algorithm of [24]. Furthermore, Jain et al. showed that no algorithm can solve the problem in $O^*((2-\varepsilon)^{pw})$ time under the Set Cover Conjecture. These works observed that it is safe to assume that the optimal solution has a nice structure: the selected edges form a matching whose endpoints are disjoint from the set of selected vertices. This observation immediately gives an $O^*(3^n)$ algorithm for the problem, which was recently improved to $O^*(2^n)$ by Madathil et al. [18] by using a dynamic programming approach, which requires $O^*(2^n)$ space.

Our results

The state of the art summarized above motivates two basic questions: first, can the gap in the complexity of the problem for treewidth and pathwidth and the gap between the lower and upper bound for these parameters be closed, as explicitly asked in [12]; second, can we solve this problem faster than the natural $O^*(2^n)$ barrier? We answer these questions and along the way obtain an improved FPT algorithm for parameter k. Specifically we show:

(i) Mixed Dominating Set can be solved in $O^*(5^{tw})$ time. Somewhat surprisingly, this result is obtained by combining observations that exist in the literature: the equivalence of Mixed Dominating Set to Distance-2-Dominating Set [18]; and the algorithm of Borradaile and Le for this problem [3].

(ii) Mixed Dominating Set cannot be solved in time $O^*((5-\varepsilon)^{pw})$, under the SETH. This is our main result on this front, and shows that our algorithm for treewidth and the algorithm of [12] for pathwidth are optimal.

(iii) Mixed Dominating Set can be solved in time $O^*(1.912^n)$ and $O^*(3.510^k)$, in both cases using polynomial space. In order to obtain these algorithms we refine the notion of *nice mixed dominating set* which was used in previous algorithms. In particular, we show that a mixed dominating set with the minimum number of vertices has the property that any selected vertex has at least two *private* neighbors. This allows us to speed up branching on low-degree vertices.

Other related work

The notion of Mixed Dominating Set was first introduced in 1977 by Alavi. et al [1], and has been studied extensively in graph theory [2, 6, 21, 23]. See the chapter in [10] for a survey on the Mixed Dominating Set problem. The computational complexity of Mixed Dominating Set was first studied in 1993 by Majumbar [19], where he showed that the problem is NP-complete. The problem remains NP-complete on split graphs [27] and on planar bipartite graphs of maximum degree 4 [20]. Majumbar [19], Lan and Chang [16], Rajaati et al. [25] and Madathil et al. [18] showed that the problem is polynomial-time solvable on trees, cacti, generalized series-parallel graphs and proper interval graphs, respectively.

[1] O^* notation suppresses polynomial factors in the input size.

2 Preliminaries

We assume familiarity with the basics of parameterized complexity (e.g. treewidth, pathwidth, and the SETH), as given in [4]. Let $G = (V, E)$ be a graph with $|V| = n$ vertices and $|E| = m$ edges. For $u \in V$, $N(u)$ denotes the set of neighbors of u, $d(u) = |N(u)|$ and $N[u] = N(u) \cup \{u\}$. For $U \subseteq V$ and $u \in V$, we note $N_U(u) = N(u) \cap U$ and use $d_U(u)$ to denote $|N_U(u)|$. Furthermore, for $U \subseteq V$ we denote $N(U) = \cup_{u \in U} N(u)$. For an edge set E', we use $V(E')$ to denote the set of endpoints of E'. For $V' \subseteq V$, we use $G[V']$ to denote the subgraph of G induced by V'. A *mixed dominating set* of a graph $G = (V, E)$ is a set of vertices $D \subseteq V$ and edges $M \subseteq E$ such that (i) all vertices of $V \setminus (D \cup V(M))$ have a neighbor in D (ii) all edges of $E \setminus M$ have an endpoint in $D \cup V(M)$.

We note that the minimization problem Mixed Dominating Set is harder than the more well-studied Edge Dominating Set (EDS) problem, in a way that preserves most parameters and the size of the optimal solution. Hence, essentially all hardness results for the latter problem, such as its inapproximability [5] or its W[1]-hardness for clique-width [7], carry over to Mixed Dominating Set.

▶ **Proposition 1.** *There is an approximation and parameter-preserving reduction from Edge Dominating Set to Mixed Dominating Set.*

Proof. Given an instance $G = (V, E)$ of EDS we seek a set M of k edges such that all edges have an endpoint in $V(M)$. We add a new vertex u connected to all of V and attach to u $|V| + 2$ leaves. The new graph has a mixed dominating set of size $k + 1$ if and only if G has an edge dominating set of size k. ◀

We now define a restricted notion of mixed dominating set.

▶ **Definition 2.** *A* nice *mixed dominating set of a graph $G = (V, E)$ is a mixed dominating set $D \cup M$ which satisfies the following: (i) $D \cap V(M) = \emptyset$; (ii) M is a matching; (iii) for all $u \in D$ there exist at least two private neighbors of u, that is, two vertices $v_1, v_2 \in V \setminus (D \cup V(M))$ with $N(v_1) \cap D = N(v_2) \cap D = \{u\}$.*

We note that a mixed dominating set that satisfies the first two properties of Definition 2 was called *special mds* in [18]. The notion of nice mds was implicit also in the algorithms of [12, 26], with the key difference that these algorithms do not use the fact that every vertex of D must have at least two *private* neighbors, that is, two neighbors which are dominated only by this vertex.

Let us now prove that restricting ourselves to nice solutions does not change the value of the optimal. The idea behind the proof is to reuse the arguments of [18] to obtain an optimal solution satisfying the first two properties; and then while there exists $u \in D$ with at most one private neighbor, we replace it by an edge while maintaining a valid solution satisfying the first two properties.

▶ **Lemma 3.** *For any graph $G = (V, E)$ without isolated vertices, G has a mixed dominating set $D \cup M$ of size at most k if and only if G has a nice mixed dominating set $D' \cup M'$ of size at most k.*

Proof. One direction is trivial, since any nice mixed dominating set is also by definition a mixed dominating set. For the other direction, we first recall that it was shown in [18] that if a graph has a mixed dominating set of size k, then it also has such a set that satisfies the first two conditions of Definition 2. Suppose then that $D \cup M$ is such that $D \cap V(M) = \emptyset$ and M is a matching.

We will now edit this solution so that we obtain the missing desired properties, namely the fact that all vertices of D have two private neighbors. Our transformations will be applicable as long as there exists a vertex $u \in D$ without two private neighbors, and will either decrease the size of the solution, or decrease the size of D, while maintaining a valid solution satisfying the first two properties of Definition 2. As a result, applying these transformations at most n times yields a nice mixed dominating set.

Let $I = V \setminus (D \cup V(M))$. If there exists $u \in D$ with exactly one private neighbor, let v be this private neighbor. We set $D' := D \setminus \{u\}$ and $M' := M \cup \{(u, v)\}$ to obtain another solution. This solution is valid because $N(u) \setminus \{v\}$ is dominated by $(D \cup M) \setminus \{u\}$, otherwise u would have more than one private neighbor.

Let us now consider a vertex $u \in D$ with no private neighbor. Note that for such a vertex u, its neighborhood (which is non-empty, since G has no isolated vertices) is dominated by $(D \cup M) \setminus \{u\}$, because otherwise u would have at least one private neighbor. If there exists $v \in N(u) \cap I$, set $D' := D \setminus \{u\}$ and $M' := M \cup \{(u, v)\}$ to obtain another feasible solution.

Now consider a vertex $u \in D$ with no private neighbor for which $N(u) \cap I = \emptyset$. We have $N(u) \subseteq D \cup V(M)$, which implies that the neighborhood of u and all the edges incident on u are dominated by $(D \cup M) \setminus \{u\}$. If there exists $v \in N(u) \cap D$, remove u from the solution to get a better solution. Now consider a vertex $u \in D$ with no private neighbor for which $N(u) \subseteq V(M)$. If there exists $v \in N(u)$ with $(v, w) \in M$ such that there exists $z \in N(w) \cap I$, set $D' := D \setminus \{u\}$ and $M' := (M \setminus \{(v, w)\}) \cup \{(u, v), (w, z)\}$ to obtain another feasible solution. If for all $v \in N(u)$ with $(v, w) \in M$, we have $N(w) \subseteq D \cap V(M)$, pick such a vertex v and set $D' := (D \setminus \{u\}) \cup \{v\}$ and $M' := M \setminus \{(v, w)\}$ to get a better solution. We repeat these modifications until we obtain the claimed solution. ◀

In the remainder, when considering a mixed dominating set $D \cup M$ of a graph $G = (V, E)$, we will associate with it the partition $V = D \cup P \cup I$ where $P = V(M)$ and $I = V \setminus (D \cup P)$. We will call this a nice mds partition. It is not hard to see that the following properties follow from the definition: (i) $G[P]$ has a perfect matching (ii) I is an independent set (iii) D dominates I (iv) each $u \in D$ has two private neighbors $v_1, v_2 \in I$, that is, $N(v_1) \cap D = N(v_2) \cap D = \{u\}$.

We also note the following useful relation.

▶ **Lemma 4.** *For any graph $G = (V, E)$ and any nice mds partition $V = D \cup P \cup I$ of G, there exists a minimal vertex cover C of G such that $D \subseteq C \subseteq D \cup P$.*

Proof. Since I is an independent set of G, $D \cup P$ is a vertex cover of G and hence contains some minimal vertex cover. We claim that any such minimal vertex cover $C \subseteq D \cup P$ satisfies $D \subseteq C$. Indeed, for each $u \in D$ there exist two private neighbors $v_1, v_2 \notin D \cup P$. Hence, if $u \notin C$, the edge (u, v_1) is not covered, contradiction. ◀

3 Treewidth

We begin with an algorithm for Mixed Dominating Set running in time $O^*(5^{tw})$. We rely on three ingredients: (i) the fact that Mixed Dominating Set on G is equivalent to Distance-2-Dominating Set on the incidence graph of G [18] ; (ii) the standard fact that the incidence graph of G has the same treewidth as G ; (iii) and an $O^*(5^{tw})$ algorithm (from [3]) for Distance-2-Dominating Set.

▶ **Theorem 5.** *There is an $O^*(5^{tw})$-time algorithm for Mixed Dominating Set in graphs of treewidth tw.*

The main result of this section is a lower bound matching Theorem 5. We prove that, under SETH, for all $\varepsilon > 0$, there is no algorithm for Mixed Dominating Set with complexity $O^*((5-\varepsilon)^{pw})$. The starting point of our reduction is the problem q-CSP-5 [15]. In this problem we are given a Constraint Satisfaction (CSP) instance with n variables and m constraints. The variables take values in a set of size 5, say $\{0,1,2,3,4\}$. Each constraint involves at most q variables and is given as a list of acceptable assignments for these variables. The following result was shown in [15] to be a natural consequence of the SETH.

▶ **Lemma 6** (Theorem 2 of [15]). *If the SETH is true, then for all $\varepsilon > 0$, there exists a q such that n-variable q-CSP-5 cannot be solved in time $O^*((5-\varepsilon)^n)$.*

Note that in [15] it was shown that for any alphabet size B, q-CSP-B cannot be solved in time $O^*\left((B-\varepsilon)^n\right)$ under the SETH, but for our purposes only the case $B = 5$ is relevant for two reasons: because this corresponds to the base of our target lower bound ; and because in our construction we will represent the $B = 5$ possible values for a variable with a path of five vertices in which there exists exactly five different ways of selecting one vertex and one edge among these five vertices. Our plan is therefore to produce a polynomial-time reduction which, given a q-CSP-5 instance with n variables, produces an equivalent Mixed Dominating Set instance whose pathwidth is at most $n + O(1)$. Then, the existence of an algorithm for the latter problem running faster than $O^*\left((5-\varepsilon)^{pw}\right)$ would give an $O^*\left((5-\varepsilon)^n\right)$ algorithm for q-CSP-5, contradicting the SETH.

Before giving the details of our reduction let us sketch the basic ideas, which follow the pattern of other SETH-based lower bounds which have appeared in the literature [8, 11, 13, 14, 17]. The constructed graph consists of a main selection part of n paths of length $5m$, divided into m sections. Each path corresponds to a variable and each section to a constraint. The idea is that the optimal solution will follow for each path a basic pattern of selecting one vertex and one edge among the first five vertices and then repeat this pattern throughout the path (see Figure 1). There are 5 natural ways to do this, so this can represent all assignments to the q-CSP-5 instance. We will then add verification gadgets to each section, connected only to the vertices of that section that represent variables appearing in the corresponding constraint (thus keeping the pathwidth under control), in order to check that the selected assignment satisfies the constraint.

The main difficulty in completing the proof is showing that the optimal solution has the desired form, and in particular, that the pattern that is selected for a variable is kept constant throughout the construction. This is in general not possible to prove, but using a technique introduced in [17], we work around this difficulty by making polynomially many copies of our construction, gluing them together, and arguing that a large enough consistent copy must exist.

Construction

We are given a q-CSP-5 instance φ with n variables $x_1, \ldots, x_n$ taking values over the set $\{0,1,2,3,4\}$, and m constraints $c_0, \ldots, c_{m-1}$. For each constraint we are given a set of at most q variables which are involved in this constraint and a list of satisfying assignments for these variables. Without loss of generality, we make the following assumptions: (i) each constraint involves exactly q variables, because if it has fewer variables, we can add to it new variables and augment the list of satisfying assignments so that the value of the new variables is irrelevant (ii) all constraints have lists of satisfying assignments of size $C = 5^q - 1$; note that this is an upper bound on the size of the list of satisfying assignments, and for each constraint which has fewer we add several copies of one of its satisfying assignments to its list (so the list may repeat an assignment). We define two "large" numbers $F = (3n+1)(2n+1)$ and $A = 20$ and we set our budget to be $k = 8AFmn + 2Fmn + 2Fmq(C-1) + n + 1$.

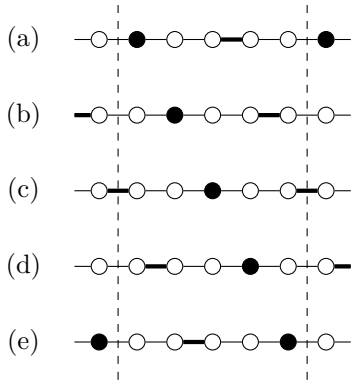


Figure 1 Main part of the construction with the five possible configurations. Filled vertices are in D, thick edges are in M.

We now construct our graph as follows:

1. We construct a vertex s and attach to it two leaves s_1, s_2.
2. For $i \in \{1, \ldots, n\}$ we construct a path on $5Fm$ vertices: the vertices are labeled $u_{i,j}$, for $j \in \{0, 1, \ldots, 5Fm - 1\}$ and for each i, j the vertex $u_{i,j}$ is connected to $u_{i,j+1}$. We call these paths the *main* part of our construction.
3. For each $j \in \{0, 1, \ldots, Fm - 1\}$, let $j' = j \bmod m$. We construct a checker gadget H_j as follows (see Figure 2):

   a. For each satisfying assignment σ in the list of the constraint $c_{j'}$, we construct an independent set $Z_{\sigma,j}$ of size $2q$ (therefore, C such independent sets). The $2q$ vertices are partitioned so that for each of the q variables involved in $c_{j'}$ we reserve two vertices. In particular, if x_i is involved in $c_{j'}$ we denote by $z^1_{\sigma,j,i}, z^2_{\sigma,j,i}$ its two reserved vertices in $Z_{\sigma,j}$.
   b. For each $i \in \{1, \ldots, n\}$ such that x_i is involved in $c_{j'}$, for each satisfying assignment σ in the list of $c_{j'}$, if σ sets x_i to value $\alpha \in \{0, 1, 2, 3, 4\}$ we add the following edges:
      i. $(u_{i,5j+\alpha}, z^1_{\sigma,j,i})$ and $(u_{i,5j+\alpha}, z^2_{\sigma,j,i})$.
      ii. Let $\beta = (\alpha + 2) \bmod 5$ and $\gamma = (\alpha + 3) \bmod 5$. We add the edges $(u_{i,5j+\beta}, z^1_{\sigma,j,i})$ and $(u_{i,5j+\gamma}, z^2_{\sigma,j,i})$.
   c. For all assignments $\sigma \neq \sigma'$ of $c_{j'}$, add all edges between $Z_{\sigma,j}$ and $Z_{\sigma',j}$.
   d. We construct an independent set W_j of size $2q(C - 1)$
   e. Add all edges between W_j and $Z_{\sigma,j}$, for all assignments σ of $c_{j'}$.
   f. For each $w \in W_j$, we construct an independent set of size $2k + 1$ whose vertices are all connected to w and to s.

4. We define the consistency gadget $Q_{i,j}$, for $i \in \{1, \ldots, n\}$ and $j \in \{0, \ldots, Fm - 1\}$ which consists of (see Figure 2):

   a. An independent set of size 8 denoted $A_{i,j}$.
   b. Five independent sets of size 2 each, denoted $B_{i,j,0}, B_{i,j,1}, \ldots, B_{i,j,4}$.
   c. For each $\ell, \ell' \in \{0, \ldots, 4\}$ with $\ell \neq \ell'$ all edges from $B_{i,j,\ell}$ to $B_{i,j,\ell'}$.
   d. For each $\ell \in \{0, \ldots, 4\}$ all possible edges from $B_{i,j,\ell}$ to $A_{i,j}$.
   e. For each $a \in A_{i,j}$, $2k + 1$ vertices connected to a and to s.

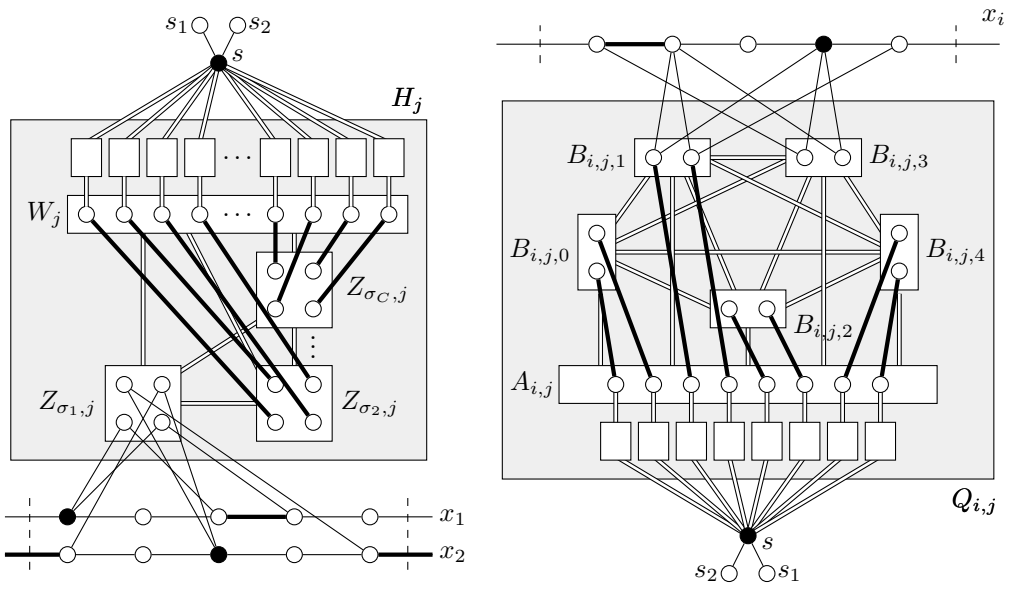


Figure 2 (Double edges between two sets of vertices represent all edges between the two sets.) Left: Checker gadget H_j connected to the main part. Here we have considered an instance where the clause $c_{j'}$ has only two variables, x_1 and x_2. Moreover, only the independent set $Z_{\sigma_1,j}$ is shown connected to the main part. The possible assignment σ_1 of $c_{j'}$ is $(x_1 = 0, x_2 = 2)$. We have supposed that this assignment is satisfiable, and we have marked the corresponding mixed dominating set: filled vertices are in D, thick edges are in M. Right: Checker gadget $Q_{i,j}$ connected to the main part, that is to the path corresponding to the variable x_i. Only the independent sets $B_{i,j,1}$ and $B_{i,j,3}$ are shown connected to the main part. We have supposed that the assignment $(x_i = 3)$ is satisfiable, and we have marked the corresponding mixed dominating set: filled vertices are in D, thick edges are in M.

f. For each $\ell \in \{0, \ldots, 4\}$ both vertices of $B_{i,j,\ell}$ are connected to $u_{i,5j+\ell}$.
g. For each $\ell \in \{0, \ldots, 4\}$ let $\ell' = (\ell + 2) \bmod 5$ and $\ell'' = (\ell + 3) \bmod 5$. One vertex of $B_{i,j,\ell}$ is connected to $u_{i,5j+\ell'}$ and the other to $u_{i,5j+\ell''}$.

5. For each $i \in \{1, \ldots, n\}$ and $j \in \{0, \ldots, Fm - 1\}$ construct A copies of the gadget $Q_{i,j}$ and connect them to the main part as described above.

This completes the construction. The target mds size is k, as defined above. We now argue that the reduction is correct and G has the desired pathwidth.

▶ **Lemma 7.** *If φ is satisfiable, then there exists an mds in G of size at most k.*

Proof. Assume that φ admits some satisfying assignment $\rho : \{x_1, \ldots, x_n\} \to \{0, 1, 2, 3, 4\}$. We construct a solution as follows:

1. For each $i \in \{1, \ldots, n\}$ let $\alpha = \rho(x_i)$. For each $j \in \{0, \ldots, Fm - 1\}$, we select in the dominating set the vertex $u_{i,5j+\alpha}$.
2. Let U' be the set of vertices $u_{i,j}$ of the main part which were not selected in the previous step and which do not have a neighbor selected in the previous step. We add to the solution all edges of a maximum matching of $G[U']$, as well as all vertices of U' left unmatched by this matching.

3. For each $j \in \{0, \ldots, Fm-1\}$, G contains a gadget H_j. Consider the constraint $c_{j'}$ for $j' = j \bmod m$. Let σ be an assignment in the list of $c_{j'}$ that agrees with ρ (such a σ must exist, since the constraint is satisfied by ρ). We add to the solution the edges of a perfect matching from W_j to $\bigcup_{\sigma' \neq \sigma} Z_{\sigma',j}$.
4. For each $j \in \{0, \ldots, Fm-1\}$ and $i \in \{1, \ldots, n\}$ we have added to the graph A copies of the consistency gadget $Q_{i,j}$. For each copy we add to the solution a perfect matching from $A_{i,j}$ to $\bigcup_{\ell \neq \rho(x_i)} B_{i,j,\ell}$.
5. We set $s \in D$.

Let us first argue why this solution has size at most k. In the first step we select Fnm vertices. In the second step we select at most $Fnm+n$ elements. To see this, note that if $u_{i,j}$ is taken in the previous step, then $u_{i,j+5}$ is also taken (assuming $j+5 < 5Fm$), which leaves two adjacent vertices $(u_{i,j+2}, u_{i,j+3})$. These vertices will be matched in $G[U']$ and in our solution. Note that, for a variable x_i, if $\rho(x_i) \neq 2$, then at most one vertex is left unmatched by the matching taken, so the cost for this variable is at most $Fm+1$. If $\rho(x_i) = 2$, then at most two vertices are left matched by the matching taken, so the cost for this variable is at most $(Fm-1)+2$. Furthermore, for each H_j we select $|W_j| = 2q(C-1)$ edges. For each copy of $Q_{i,j}$ we select 8 edges, for a total cost of $8AFmn$. Taking into account s, the total cost is at most $Fnm + n + Fnm + 2Fmq(C-1) + 8AFmn + 1 = k$.

Let us argue why the solution is feasible. First, all vertices $u_{i,j}$ and all edges connecting them to each other are clearly dominated by the first two steps of our selection. Second, for each H_j, the vertex s together with the endpoints of selected edges form a vertex cover of H_j, so all internal edges are dominated. Furthermore, s dominates all vertices which are not endpoints of our solution, except $Z_{\sigma,j}$, where σ is the selected assignment of $c_{j'}$, with $j' = j \bmod m$. We then need to argue that the vertices of $Z_{\sigma,j}$ and the edges connecting it to the main part are covered.

Recall that the $2q$ vertices of $Z_{\sigma,j}$ are partitioned into pairs, with each pair $z^1_{\sigma,j,i}, z^2_{\sigma,j,i}$ reserved for the variable x_i involved in $c_{j'}$. We now claim that $z^1_{\sigma,j,i}, z^2_{\sigma,j,i}$ are dominated by our solution, since we have selected the vertex $u_{i,5j+\alpha}$, where $\alpha = \rho(x_i)$. Furthermore, $u_{i,5j+\beta}, u_{i,5j+\gamma}$, where $\beta = (a+2) \bmod m$, $\gamma = (a+3) \bmod m$, belong in U' and therefore the edges incident to them are covered. Finally, to see that the $Q_{i,j}$ gadgets are covered, observe that for each such gadget only 2 vertices of some $B_{i,j,\ell}$ are not in P. The common neighbor of these vertices is in D, and their other neighbors in the main part are in P. ◀

The idea of the proof of the next Lemma is the following: by partitioning the graph into different parts and lower bound the cost of these parts, we prove that if a mixed dominating set in G has not the same form as in Lemma 7 in a sufficiently large copy, then it has size strictly greater than k, enabling us to produce a satisfiable assignment for φ using the mixed dominating set which has the desired form.

▶ **Lemma 8.** *If there exists a mixed dominating set in G of size at most k, then φ is satisfiable.*

Proof. Suppose that we are given, without loss of generality (Lemma 3), a nice mixed dominating set of G of minimum cost. We therefore have a partition of $V(G)$ into $V = D \cup P \cup I$, and a perfect matching M of $G[P]$. Before proceeding, let us define for a set $S \subseteq V(G)$ its *cost* as $\text{cost}(S) = |S \cap D| + \frac{|S \cap P|}{2}$. Clearly, $\text{cost}(V(G)) \leq k$ and for disjoint sets S_1, S_2 we have $\text{cost}(S_1 \cup S_2) = \text{cost}(S_1) + \text{cost}(S_2)$. Our strategy will therefore be to partition V into different parts and lower bound their cost.

First, we give some notations. Consider some $j \in \{0, \dots, Fm-1\}$ and $i \in \{1, \dots, n\}$: recall that we have constructed A copies of the gadget $Q_{i,j}$, call them $Q^1_{i,j}, \dots, Q^A_{i,j}$; also let $S_{i,j} = \{u_{i,5j}, u_{i,5j+1}, \dots, u_{i,5j+4}\}$. Now, for some $j \in \{0, \dots, Fm-1\}$, let $S_j = H_j \cup \bigcup_{i \in \{1,\dots,n\}} \left(S_{i,j} \cup \bigcup_{r \in \{1,\dots,A\}} Q^r_{i,j}\right)$.

▷ Claim 9. $\text{cost}(S_j) \geq 2q(C-1) + 2n + 8An$.

Proof. We begin with some easy observations. First, it must be the case that $s \in D$. If not, either s_1 or s_2 are in D, which contradicts the niceness of the solution.

Consider some $j \in \{0, \dots, Fm-1\}$ and $i \in \{1, \dots, n\}$. We will say that, for $1 \leq r \leq A$, $Q^r_{i,j}$ is *normal* if we have the following: $Q^r_{i,j} \cap D = \emptyset$ and there exists $\ell \in \{0, \dots, 4\}$ such that $Q^r_{i,j} \cap P = A_{i,j} \cup \bigcup_{\ell' \neq \ell} B_{i,j,\ell'}$. In other words, $Q^r_{i,j}$ is normal if locally the solution has the form described in Lemma 7.

We now observe that for all i, j, r we have $\text{cost}(Q^r_{i,j}) \geq 8$. To see this, observe that if there exists $a \in A_{i,j} \cap I$, then the $2k+1$ neighbors of a must be in $D \cup P$, so the solution cannot have cost k. Hence, $A_{i,j} \subseteq D \cup P$. Furthermore, the maximum independent set of $\bigcup_{\ell \in \{0,\dots,4\}} B_{i,j,\ell}$ is 2, so $|(\bigcup_{\ell \in \{0,\dots,4\}} B_{i,j,\ell}) \cap (D \cup P)| \geq 8$. Following this reasoning we also observe that if $Q^r_{i,j}$ is not normal, then we have $\text{cost}(Q^r_{i,j}) > 8$. In other words, 8 is a lower bound for the cost of every copy of $Q_{i,j}$, which can only be attained if a copy is normal.

Consider some $j \in \{0, \dots, Fm-1\}$ and $i \in \{1, \dots, n\}$ and suppose that none of the A copies of $Q_{i,j}$ is normal. We will then arrive at a contradiction. Indeed, we have $\text{cost}(\bigcup_r Q^r_{i,j}) \geq 8A + A/2 \geq 8A + 10$. We create another solution by doing the following: take the five vertices $u_{i,5j}, u_{i,5j+1}, \dots, u_{i,5j+4}$, and take in all $Q_{i,j}$ a matching so that $Q_{i,j}$ is normal. This has decreased the total cost, while keeping the solution valid, which should not be possible.

We can therefore assume from now on that for each i, j at least one copy of $Q_{i,j}$ is normal, hence, there exists $\ell \in \{0, \dots, 4\}$ such that $B_{i,j,\ell} \subseteq I$ in that copy.

Recall that $S_{i,j} = \{u_{i,5j}, u_{i,5j+1}, \dots, u_{i,5j+4}\}$. We claim that for all $i \in \{1, \dots, n\}, j \in \{0, \dots, Fm-1\}$, we have $\text{cost}(S_{i,j}) \geq 2$. Indeed, if we consider the normal copy of $Q_{i,j}$ which has $B_{i,j,\ell} \subseteq I$, the two vertices of $B_{i,j,\ell}$ have three neighbors in $S_{i,j}$, and at least one of them must be in D.

In addition, we claim that for all $j \in \{0, \dots, Fm-1\}$ we have $\text{cost}(H_j) \geq 2q(C-1)$. The reasoning here is similar to $Q_{i,j}$, namely, the vertices of W_j cannot belong to I (otherwise we get $2k+1$ vertices in $D \cup P$); and from the $2qC$ vertices in $\bigcup_\sigma Z_{\sigma,j}$ at most $2q$ can belong to I.

We now have the lower bounds we need: $\text{cost}(S_j) \geq 2q(C-1) + 2n + 8An$. ◁

Now, if for some j we have $\text{cost}(S_j) > 2q(C-1) + 2n + 8An$ we will say that j is *problematic*.

▷ Claim 10. There exists a contiguous interval $J \subseteq \{0, \dots, Fm-1\}$ of size at least $m(3n+1)$ in which all $j \in J$ are not problematic.

Proof. Let $L \subseteq \{0, \dots, Fm-1\}$ be the set of problematic indices. We claim that $|L| \leq 2n$. Indeed, we have $\text{cost}(V(G)) = 1 + \sum_{j \in \{0,\dots,Fm-1\}} \text{cost}(S_j) \geq 1 + Fm(2q(C-1) + 2n + 8An) + |L|/2 = k - n + |L|/2$. But since the total cost is at most k, we have $|L|/2 \leq n$.

We will now consider the longest contiguous interval $J \subseteq \{0, \dots, Fm-1\}$ such that all $j \in J$ are not problematic. We have $|J| \geq Fm/(|L|+1) \geq m(3n+1)$. ◁

Before we proceed further, we note that if j is not problematic, then for any $i \in \{1, \dots, n\}$, all edges of M which have an endpoint in $S_{i,j}$, must have their other endpoint also in the main part, that is, they must be edges of the main paths. To see this note that if j is not

problematic, all $Q_{i,j}$ are normal, so there are 8 vertices in $A_{i,j} \cap P$ which must be matched to the 8 vertices of $(\bigcup_\ell B_{i,j,\ell}) \cap P$. Similarly, in H_j the $2q(C-1)$ vertices of $W_j \cap P$ must be matched to the $2q(C-1)$ vertices of $(\bigcup_\sigma Z_{\sigma,j}) \cap P$, otherwise we would increase the cost and j would be problematic.

Consider now a non-problematic $j \in J$ and $i \in \{1, \ldots, n\}$ such that $\text{cost}(S_{i,j}) = 2$. We claim that the solution must follow one of the five configurations below (see also Figure 1):

(a) $u_{i,5j} \in D$ and $(u_{i,5j+2}, u_{i,5j+3}) \in M$.
(b) $u_{i,5j+1} \in D$ and $(u_{i,5j+3}, u_{i,5j+4}) \in M$.
(c) $u_{i,5j+2} \in D$, $(u_{i,5j+4}, u_{i,5j+5}) \in M$, and $(u_{i,5j-1}, u_{i,5j}) \in M$.
(d) $u_{i,5j+3} \in D$ and $(u_{i,5j}, u_{i,5j+1}) \in M$.
(e) $u_{i,5j+4} \in D$ and $(u_{i,5j+1}, u_{i,5j+2}) \in M$.

Indeed, it is not hard to see that these configurations cover all the cases where exactly one vertex of $S_{i,j}$ is in D and exactly two are in P. This is a condition enforced by the fact that one of the $Q_{i,j}$ copies is normal, and that $\text{cost}(S_{i,j}) = 2$.

▷ Claim 11. There exists a contiguous interval $J' \subseteq \{0, \ldots, Fm-1\}$ of size at least m in which all $j \in J'$ are not problematic and for all $j_1, j_2 \in J'$, S_{i,j_1} and S_{i,j_2} are in the same configuration.

Proof. Given the five configurations, we now make the following simple observations, where statements apply for all $i \in \{1, \ldots, n\}$ and j such that $j, j+1 \in J$:

- If $S_{i,j}$ is in configuration (a), then $S_{i,j+1}$ is also in configuration (a).
- If $S_{i,j}$ is in configuration (c), then $S_{i,j+1}$ is also in configuration (c).
- If $S_{i,j}$ is in configuration (d), then $S_{i,j+1}$ is in configuration (d) or (a).
- If $S_{i,j}$ is in configuration (b), then $S_{i,j+1}$ is in configuration (b), (d), or (a).

For the first claim, we note that in configuration (a) vertex $u_{i,5j+4}$ is not dominated, forcing the selection of $u_{i,5j+5} \in D$. The second claim is obtained by the observation that all edges of M in this area of the graph must be edges of the path and a parity argument. The third claim is based on the fact that in configuration (d) the edge $(u_{i,5j+4}, u_{i,5j+5})$ must be covered by placing $u_{i,5j+5}$ in $D \cup P$. Finally, configuration (b) cannot be followed by configuration (c), again for parity reasons, nor by configuration (e), because the vertex $u_{i,5j+5}$ would be uncovered.

We will now say for some $i \in \{1, \ldots, n\}$, $j \in J$, that j is *shifted* for variable i if $j+1 \in J$ but $S_{i,j}$ and $S_{i,j+1}$ do not have the same configuration. We observe that there cannot exist distinct $j_1, j_2, j_3, j_4 \in J$ such that all of them are shifted for variable i. Indeed, if we draw a directed graph with a vertex for each configuration, and an arc (u, v) expressing the property that the configuration represented by v can follow the one represented by u, if we take into account the observations above, the graph will be a DAG with maximum path length 3. Hence, a configuration cannot shift 4 times, as long as we stay in J (the part of the graph where the minimum local cost is attained everywhere).

By the above, the number of shifted indices $j \in J$ is at most $3n$. Hence, the longest contiguous interval without shifted indices has length at least $|J|/(3n+1) \geq m$. Let J' be this interval. ◁

We are now almost done: we have located an interval $J' \subseteq \{0, \ldots, Fm-1\}$ of length at least m where for all $i \in \{1, \ldots, n\}$ and all $j_1, j_2 \in J'$ we have the same configuration in S_{i,j_1} and S_{i,j_2}. We now extract an assignment from this in the natural way: if $u_{i,5j+\ell} \in D$,

for some $j \in J', \ell \in \{0, \ldots, 4\}$, then we set $x_i = \ell$. We claim this satisfies φ. Consider a constraint $c_{j'}$ of φ. There must exist $j \in J'$ such that $j' = j \bmod m$, because $|J'| \geq m$ and J' is contiguous. We therefore check H_j, where there exists σ such that $Z_{\sigma,j} \subseteq I$ (this is because j is not problematic, that is, H_j attains the minimum cost). But because the vertices and incident edges of $Z_{\sigma,j}$ are dominated, it must be the case that the assignment we extracted agrees with σ, hence $c_{j'}$ is satisfied. ◀

We now show that the pathwidth of G is at most $n + O(1)$.

▶ **Lemma 12.** *The pathwidth of G is at most $n + O(q5^q)$.*

Proof. We will show how to build a path decomposition. First, we can add s to all bags, so we focus on the rest of the graph. Second, after removing s from the graph, some vertices become leaves. It is a well-known fact that removing all leaves from a graph can only increase the pathwidth by at most 1. To see this, let G' be the graph obtained after deleting all leaves of G and suppose we have a path decomposition of G' of width w. We obtain a path decomposition of G by doing the following for every leaf v: find a bag of width at most w that contains the neighbor of v and insert after this bag, a copy of the bag with v added. Clearly, the width of the new decomposition is at most $w + 1$. Because of the above we will ignore all vertices of G which become leaves after the removal of s.

For all $j \in \{0, \ldots, Fm-1\}$, we will denote $S_j = H_j \cup \bigcup_{i \in \{1,\ldots,n\}} \left(S_{i,j} \cup \bigcup_{r \in \{1,\ldots,A\}} Q^r_{i,j} \right)$, where $S_{i,j} = \{u_{i,5j}, \ldots, u_{i,5j+4}\}$, and $Q^1_{i,j}, \ldots, Q^A_{i,j}$ are the A copies of the gadget $Q_{i,j}$. We will show how to build a path decomposition of $G[S_j]$ with the following properties:

- The first bag of the decomposition contains vertices $u_{i,5j}$, for all $i \in \{1, \ldots, n\}$.
- The last bag of the decomposition contains vertices $u_{i,5j+4}$, for all $i \in \{1, \ldots, n\}$.
- The width of the decomposition is $n + O(q5^q)$.

If we achieve the above then we can obtain a path decomposition of the whole graph: indeed, the sets S_j partition all remaining vertices of the graph, while the only edges not covered by the above decompositions are those between $u_{i,5j+4}$ and $u_{i,5(j+1)}$. We therefore place the decompositions of S_j in order, and then between the last bag of the decomposition of S_j and the first bag of the decomposition of S_{j+1} we have $2n$ "transition" bags, where in each transition step we add a vertex $u_{i,5(j+1)}$ in the bag, and then remove $u_{i,5j+4}$.

Let us now show how to obtain a decomposition of $G[S_j]$, having fixed the contents of the first and last bag. First, H_j has order $O(q5^q)$, so we place all its vertices to all bags. The remaining graph is a union of paths of length 4 with the $Q_{i,j}$ gadgets attached. We therefore have a sequence of $O(n)$ bags, where for each $i \in \{1, \ldots, n\}$ we add to the current bag the vertices of $S_{i,j}$, then add and remove one after another whole copies of $Q_{i,j}$, then remove $S_{i,j}$ except for $u_{i,5j+4}$. ◀

We are now ready to present the main result of this section. By putting together Lemmas 7, 8, 12 and the negative result for q-CSP-5 (Lemma 6), we get the following Theorem:

▶ **Theorem 13.** *Under SETH, for all $\varepsilon > 0$, no algorithm solves Mixed Dominating Set in time $O^*((5-\varepsilon)^{pw})$, where pw is the input graph's pathwidth.*

Proof. Fix $\varepsilon > 0$ and let q be sufficiently large so that Lemma 6 is true. Consider an instance φ of q-CSP-5. Using our reduction, create an instance (G, k) of Mixed Dominating Set. Thanks to Lemma 7 and Lemma 8, we know that φ is satisfiable if and only if there exists a mixed dominating set of size at most k in G.

Suppose there exists an algorithm which solves Mixed Dominating Set in time $O^*((5-\varepsilon)^{pw})$. With this algorithm and our reduction, we can determine if φ is satisfiable in time $O^*((5-\varepsilon)^{pw})$, where $pw = n + O(q5^q) = n + O(1)$, so the total running time of this procedure is $O^*((5-\varepsilon)^n)$, contradicting the SETH. ◀

4 Exact Algorithm

In this section, we describe an algorithm for the Mixed Dominating Set problem running in time $O^*(1.912^n)$. Let us first give an overview of our algorithm. Consider an instance $G = (V, E)$ of the Mixed Dominating Set problem and fix, for the sake of the analysis, an optimal solution which is a nice mixed dominating set. Such an optimal solution must exist by Lemma 3, so suppose it gives the nice mds partition $V = D \cup P \cup I$.

By Lemma 4, there exists a minimal vertex cover C of G for which $D \subseteq C \subseteq D \cup P$. Our first step is to "guess" C, by enumerating all minimal vertex covers of G. This decreases our search space, since we can now assume that vertices of C only belong in $D \cup P$, and vertices of $V \setminus C$ only belong in $P \cup I$.

For our second step, we branch on the vertices of V, placing them in D, P, or I. The goal of this branching is to arrive at a situation where our partial solution dominates $V \setminus C$. The key idea is that any vertex of C that may belong in D must have at least two private neighbors, hence this allows us to significantly speed up the branching for low-degree vertices of D. Finally, once we have a partial solution that dominates all of $V \setminus C$, we show how to complete this optimally in polynomial time using a maximum matching computation.

We now describe the three steps of our algorithm in order and give the properties we are using step by step. In the remainder we assume that G has no isolated vertices (since these are trivially handled). Therefore, by Lemma 3 there exists an optimal nice mds. Denote the corresponding partition as $V = D \cup P \cup I$.

Step 1. Enumerate all minimal vertex covers of G. For each such vertex cover C we execute the rest of the algorithm. In the end output the best solution found.

Thanks to Lemma 4, there exists a minimal vertex cover C with $D \subseteq C \subseteq D \cup P$. Since we will consider all minimal vertex covers, in the remainder we focus on the case where the set C considered satisfies this property. Let $Z = V \setminus C$. Then Z is an independent set of G. We now get two properties we will use in the branching step of our algorithm:

1. For all $u \in C$, u can be either in D or in P, because $C \subseteq D \cup P$.
2. For all $v \in Z$, v can be either in P or in I, because $D \subseteq C$.

Step 2. Branch on the vertices of V as described below.

The branching step of our algorithm will be a set of Reduction and Branching Rules over the vertices of C or Z. In order to describe a recursive algorithm, it will be convenient to consider a slightly more general version of the problem: in addition to G, we are given three disjoint sets $D_f, P_f, P'_f \subseteq V$, and the question is to build a nice mds partition $V = D \cup P \cup I$ of minimum cost which satisfies the following properties: $D_f \subseteq D \subseteq C$, $P_f \subseteq P \cap C$, and $P'_f \subseteq P \cap Z$. Clearly, if $D_f = P_f = P'_f = \emptyset$ we have the original problem and all properties are satisfied. We will say that a branch where all properties are satisfied is *good*, and our proof of correctness will rely on the fact that when we branch on a good instance, at least one of the produced branches is good. The intuitive meaning of these sets is that when we decide in a branch that a vertex belongs in D or in P in the optimal partition we place it respectively in D_f, P_f or P'_f (depending on whether the vertex belongs in C or Z).

We now describe a series of Rules which, given an instance of Mixed Dominating Set and three sets D_f, P_f, P'_f, will recursively produce subinstances where vertices are gradually placed into these sets. Our algorithm will consider the Reduction and Branching Rules in order and apply the first Rule that can be applied. Note that we say that a vertex u is *decided* if it is in one of the sets $D_f \subseteq D$, $P_f \subseteq P$, or $P'_f \subseteq P$. All the other vertices are considered *undecided*.

Throughout the description that follows, we will use U to denote the set of undecided vertices which are not dominated by D_f, that is, $U := V \setminus (D_f \cup P_f \cup P'_f \cup (N(D_f) \cap Z))$. We will show that when no rule can be applied, U is empty, that is, all vertices are decided or dominated by D_f. In the third step of our algorithm we will show how to complete the solution in polynomial time when U is empty. Since our Rules do not modify the graph, we will describe the subinstances we branch on by specifying the tuple (D_f, P_f, P'_f).

To ease notation, let $U_C = U \cap C$ and $U_Z = U \cap Z$. Recall that for $u \in V$, we use $d_{U_C}(u)$ and $d_{U_Z}(u)$ to denote the size of the sets $N(u) \cap U_C = N_{U_C}(u)$ and $N(u) \cap U_Z = N_{U_Z}(u)$, respectively.

Reduction Rule (R1): If there exists $u \in U_C$ such that $d_{U_Z}(u) \leq 1$, then put u in P_f, that is, recurse on the instance $(D_f, P_f \cup \{u\}, P'_f)$.

Reduction Rule (R2): If there exists $v \in U_Z$ such that $d_{U_C}(v) = 0$, then put u in P'_f, that is, recurse on the instance $(D_f, P_f, P'_f \cup \{v\})$.

Branching Rule (B1): If there exists $u \in U_C$ such that $d_{U_Z}(u) \geq 4$, then branch on the following two subinstances: $(D_f \cup \{u\}, P_f, P'_f)$ and $(D_f, P_f \cup \{u\}, P'_f)$.

Note that we may now assume that all vertices of U_C have $d_{U_Z} \in \{2, 3\}$. The following two rules eliminate vertices $u \in U_C$ with $d_{U_Z}(u) = 2$.

Branching Rule (B2.1): If there exists $u_1, u_2 \in U_C$ such that $d_{U_Z}(u_1) = 3$, $d_{U_Z}(u_2) = 2$, and $N_{U_Z}(u_1) \cap N_{U_Z}(u_2) \neq \emptyset$ then branch on the following instances: $(D_f \cup \{u_1\}, P_f \cup \{u_2\}, P'_f)$ and $(D_f, P_f \cup \{u_1\}, P'_f)$.

Branching Rule (B2.2): If there exists $u \in U_C$ with $d_{U_Z}(u) = 2$ we branch on the instances $(D_f \cup \{u\}, P_f, P'_f)$ and $(D_f, P_f \cup \{u\}, P'_f)$.

We now have that all vertices $u \in U_C$ have $d_{U_Z}(u) = 3$. Let us now branch on vertices of U_Z to ensure that these also do not have too low degree.

Branching Rule (B3.1): If there exists $v \in U_Z$ with $d_{U_C}(v) = 1$ let $N_{U_C}(v) = \{u\}$. We branch on the instances $(D_f \cup \{u\}, P_f, P'_f)$ and $(D_f, P_f \cup \{u\}, P'_f)$.

Branching Rule (B3.2): If there exists $v \in U_Z$ with $d_{U_C}(v) = 2$ let $N_{U_C}(v) = \{u_1, u_2\}$. We branch on the instances $(D_f \cup \{u_1\}, P_f, P'_f)$, $(D_f \cup \{u_2\}, P_f \cup \{u_1\}, P'_f)$, and $(D_f, P_f \cup \{u_1, u_2\}, P'_f)$.

If we cannot apply any of the above Rules, for all $u \in U_C$ we have $d_{U_Z}(u) = 3$ and for all $v \in U_Z$ we have $d_{U_C}(v) \geq 3$. We now consider three remaining cases: (i) there exists a C_4 made up of two vertices of U_C and two vertices of U_Z (ii) there exists a vertex $v \in U_Z$ with $d_{U_C}(v) = 3$ (iii) everything else.

Branching Rule (B4): If there exist $u_1, u_2 \in U_C$ and $v_1, v_2 \in U_Z$ with $(u_i, v_j) \in E$ for all $i, j \in \{1, 2\}$, then we branch on the instances $(D_f \cup \{u_1\}, P_f \cup \{u_2\}, P'_f)$ and $(D_f, P_f \cup \{u_1\}, P'_f)$.

Branching Rule (B5): If there exists $v \in U_Z$ with $d_{U_C}(v) = 3$, let $N_{U_C}(v) = \{u_1, u_2, u_3\}$ and for $i \in \{1, 2, 3\}$ let $X_i = \{w \in U_C \setminus \{u_1, u_2, u_3\} \mid N(w) \cap N(u_i) \cap (U_Z \setminus \{v\}) \neq \emptyset\}$, that is, X_i is the set of vertices of U_C that share a neighbor with u_i in U_Z other than v. Then we branch on the following 8 instances: (i) the instance $(D_f, P_f \cup \{u_1, u_2, u_3\}, P'_f \cup \{v\}\}$

(ii) for $i \in \{1,2,3\}$, we produce the instances $(D_f \cup \{u_i\}, P_f \cup (\{u_1,u_2,u_3\} \setminus \{u_i\}), P'_f)$ (iii) for $i,j \in \{1,2,3\}$, with $i < j$ we produce the instances $(D_f \cup \{u_i,u_j\}, P_f \cup (\{u_1,u_2,u_3\} \setminus \{u_i,u_j\}) \cup X_i \cup X_j, P'_f)$ (iv) we produce the instance $(D_f \cup \{u_1,u_2,u_3\}, P_f \cup X_1 \cup X_2 \cup X_3, P'_f)$.

Branching Rule (B6): Consider $u \in U_C$ and let $N_{U_Z}(u) = \{v_1,v_2,v_3\}$. We branch on the following instances: $(D_f, P_f \cup \{u\}, P'_f)$, $(D_f \cup \{u\}, P_f \cup N_{U_1}(v_1) \setminus \{u\}, P'_f)$, and $(D_f \cup \{u\}, P_f \cup (N_{U_1}(v_2) \cup N_{U_1}(v_3)) \setminus \{u\}, P'_f)$.

Our algorithm applies the above Rules in order as long as possible. Before proceeding to explain what happens when no Rule is applicable, let us first establish two useful correctness properties. We will say that a tuple (D_f, P_f, P'_f) is *good* if $D_f \subseteq D$, $P_f \subseteq P \cap C$, and $P'_f \subseteq P \setminus C$.

▶ **Lemma 14.** *If we apply the first Rule that can be applied on an instance characterized by a good tuple, then we produce at least one instance characterized by a good tuple.*

Proof. We consider the Rules in order. For Reduction Rule 1, observe that all neighbors of u in U_1 cannot be private neighbors of u since $U_C \subseteq C \subseteq D \cup P$, and because $d_{U_Z}(u) \leq 1$, the vertex u can have at most one private neighbor, so it must be the case that $u \in P$. For Reduction Rule 2, v must be dominated, but it has no neighbor in U_C, so it must be the case that $v \in P$. Branching Rule B1 is trivially correct from $C \subseteq D \cup P$.

Branching Rule B2.1 is correct because if $u_1 \in D$, then u_2 cannot have two private neighbors and it is forced to be in P. Branching Rule B2.2 is trivially correct again from $C \subseteq D \cup P$.

Again, from $C \subseteq D \cup P$, Branching Rule B3.1 is trivially correct. Branching rule B3.2 is correct since we have the three following cases: $u_1 \in D$; or $u_1 \in P$ and $u_2 \in D$; or u_1 and $u_2 \in P$.

Branching Rule B4 is correct because if $u_1 \in D$, then u_2 cannot have two private neighbors since $d_{U_Z}(v) = 3$.

Branching Rule B5 is correct since we have the following cases: all vertices u_1, u_2 and u_3 are in P ; or exactly one of them is in D ; or exactly two of them are in D ; or all of them are in D. Note first that u_1, u_2 and u_3 only share v as neighbor in U_Z since Branching Rule B4 is not triggered. In the first case, v has to be dominated so it must be the case that $v \in P$. In the second case, the two vertices not in D necessarily are in P. In the third case, since u_i and u_j share v as common neighbor and both have exactly three neighbors in U_Z, the vertices of X_i and X_j have to be in P because otherwise u_i and u_j do not have two private neighbors. In the last case, and for the same reason, the vertices of X_1, X_2, and X_3 have to be in P.

Finally, Branching Rule B6 is correct because if $u \in D$, then either v_1 is one of its private neighbors, or both v_2 and v_3 are its private neighbors. ◀

▶ **Lemma 15.** *If none of the Rules can be applied then $U = \emptyset$.*

Proof. Observe that by applying rules R1, B1, B2.2, B6, we eventually eliminate all vertices of U_C, since these rules alone cover all the cases for $d_{U_Z}(u)$ for any $u \in U_C$. So, if none of these rules applies, U_C is empty. But then applying R2 will also eliminate U_Z, which makes all of U empty. ◀

Step 3. When U is empty, reduce the problem to Maximum Matching.

We now show how to complete the solution in polynomial time.

▶ **Lemma 16.** *Let (D_f, P_f, P'_f) be a good tuple such that no Rule can be applied. Then it is possible to construct in polynomial time a mixed dominating set of size $|D| + \frac{|P|}{2}$.*

Proof. Because no Rule can be applied, by Lemma 15 we have $U = V \setminus (D_f \cup P_f \cup P'_f \cup (N(D_f) \setminus C)) = \emptyset$.

Let M be a maximum matching of $G[P_f \cup P'_f]$. Then, we claim that $|D| + |P|/2 \geq |D_f| + |P_f \cup P'_f| - |M|$. First, $|D| \geq |D_f|$ because $D_f \subseteq D$. We now claim that $P \setminus (P_f \cup P'_f)$ is an independent set. Indeed, $P \setminus (P_f \cup P'_f)$ is a set of undecided vertices, and because U is empty, the only undecided vertices are those in $Z \cap N(D_f)$, which is an independent set. Consider now a perfect matching M' of $G[P]$, and let M'' be the set of edges of that matching that have both endpoints in $P_f \cup P'_f$. Clearly, $|M''| \leq |M|$. Let $P' = (P_f \cup P'_f) \setminus V(M'')$. By the definitions of M', M'', P', and the fact that $P \setminus (P_f \cup P'_f)$ is an independent set, we have: $|P|/2 = |M'| = |M''| + |P'|$. By this, we get: $|P|/2 = |M''| + |P_f \cup P'_f| - |V(M'')| = |P_f \cup P'_f| - |M''| \geq |P_f \cup P'_f| - |M|$. By summing this last inequality with $|D| \geq |D_f|$, we get: $|D| + |P|/2 \geq |D_f| + |P_f \cup P'_f| - |M|$.

We will now show how to construct a valid mixed dominating set of size $|D_f| + |P_f \cup P'_f| - |M|$ in polynomial time, where again M is a maximum matching of $G[P_f \cup P'_f]$. Specifically, we select all vertices of D_f, all edges of M, and an edge incident on each unmatched vertex of $P_f \cup P'_f$. The size of such a solution is $|D_f| + |M| + (|P_f \cup P'_f| - 2|M|)$, which is equal to the bound we promised.

To conclude, let us explain why the solution we have produced is a valid mixed dominating set (even though it is not necessarily a nice mds). First, the solution we produced puts all vertices of C in $D \cup P$, therefore, since C is a vertex cover, all edges are covered. Second, since all Rules were exhaustively applied, our tuple gives $U = \emptyset$, which implies that all vertices of Z are either in $N(D_f)$ or in P'_f, therefore dominated. ◀

We give a small overview of the analysis of the running time of our exact algorithm. First, enumerating all minimal vertex covers takes times at most $O^*(3^{n/3})$, which is also an upper bound on the number of such covers [22]. Moreover, we observe that we can decide if a Rule applies in polynomial time, and the algorithm of Lemma 16 runs in polynomial time. We therefore only need to bound the number of subinstances the branching step will produce, as a function of n.

We define our measure of progress as the size of the set $\{u \in U_C \mid d_{U_Z}(u) \geq 2\} \cup \{v \in U_Z \mid d_{U_C}(v) \geq 1\}$. In other words, we count the undecided vertices of U_C that have at least two undecided, non-dominated vertices in Z, and the undecided, non-dominated vertices of Z that have at least one undecided neighbor in C. This is motivated by the fact that undecided vertices that do not respect these degree bounds are eliminated by the Reduction Rules and hence do no affect the running time. Let l denote the number of vertices that we counted according to this measure. Clearly, we have $l \leq n$.

Of all the above rules, the worst case is given by Branching Rule B5, which leads to a complexity of 1.3252^l. Taking into account the cost of enumerating all minimal vertex covers and the fact that $l \leq n$, the running time of our algorithm is $O^*(3^{n/3} \cdot 1.3252^n) = O^*(1.912^n)$.

▶ **Theorem 17.** *Mixed Dominating Set can be solved in time $O^*(1.912^n)$ and polynomial space.*

5 FPT Algorithm

In this section we describe an algorithm for k-Mixed Dominating Set running in time $O^*(3.510^k)$, where k is the value of the optimal solution. Our algorithm is based on a branching procedure very similar to the one used in [26], which runs in time $O^*(4.172^k)$. We only sketch the different branching rules and explain, on a high level, how the notion of nice dominating sets allows us to obtain the improved running time.

We fix for the analysis an optimal nice mds and its partition $V = D \cup P \cup I$. The branching algorithm will gradually build two sets D_f, P_f which are the vertices decided to be in D, P respectively. Let $U = V \setminus (D_f \cup P_f)$ be the set of undecided vertices. As noted in [26], a basic branching algorithm considers for each $u \in U$ the cases $u \in D_f$, $u \in P_f$, and all partitions of $N_U(u)$ into D_f, P_f. This leads to a performance similar to that of [12] ($O^*(7.465^k)$). The key idea of [26] is to identify the importance of the set $U^* = U \setminus N(D_f)$ of undecided, undominated vertices. Branching on U^* is faster because we no longer need to consider the case $N(u) \subseteq P_f$. Once $U^* = \emptyset$, the problem becomes much easier.

Our improvement is based on the fact that (by Lemma 3) each $u \in D$ has two private neighbors. This speeds up branching on U^*, as we have: (i) if $d_{U^*}(u) < 2$, then the branch $u \in D_f$ need not be considered (ii) if $d_{U^*}(u) = 2$ then in the branch where $u \in D_f$ we may assume that the two vertices $v_1, v_2 \in N(u) \cap U^*$ are private neighbors of u, so their undecided neighbors are automatically placed in P_f (iii) if $d_{U^*}(u) > 3$, for the branch where $u \in D_f$ we can consider sub-branches where we decide which are the private neighbors of u, placing the neighbors of these vertices in P_f. A key element of our analysis is that, because we have sped up the branching on low-degree ($d_{U^*}(u) \leq 2$) vertices, in subsequent branches we are allowed to assume that all neighbors of u have several undecided neighbors, increasing the profit of guessing that v is a private neighbor of u. Using these ideas we speed up the branching on U^* and the remainder of the algorithm follows along similar lines to [26].

▶ **Theorem 18.** *k-Mixed Dominating Set can be solved in time $O^*(3.510^k)$.*

References

1 Yousef Alavi, M. Behzad, Linda M. Lesniak-Foster, and E. A. Nordhaus. Total matchings and total coverings of graphs. *Journal of Graph Theory*, 1(2):135–140, 1977.

2 Yousef Alavi, Jiuqiang Liu, Jianfang Wang, and Zhongfu Zhang. On total covers of graphs. *Discrete Mathematics*, 100(1-3):229–233, 1992.

3 Glencora Borradaile and Hung Le. Optimal dynamic program for r-domination problems over tree decompositions. In *IPEC*, volume 63 of *LIPIcs*, pages 8:1–8:23, 2016.

4 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

5 Szymon Dudycz, Mateusz Lewandowski, and Jan Marcinkowski. Tight approximation ratio for minimum maximal matching. In *IPCO*, volume 11480 of *LNCS*. Springer, 2019.

6 Paul Erdös and Amram Meir. On total matching numbers and total covering numbers of complementary graphs. *Discrete Mathematics*, 19(3):229–233, 1977.

7 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010.

8 Tesshu Hanaka, Ioannis Katsikarelis, Michael Lampis, Yota Otachi, and Florian Sikora. Parameterized orientable deletion. In *SWAT*, volume 101 of *LIPIcs*, pages 24:1–24:13, 2018.

9 Pooya Hatami. An approximation algorithm for the total covering problem. *Discussiones Mathematicae Graph Theory*, 27(3):553–558, 2007.

10 Teresa W. Haynes, Stephen T. Hedetniemi, and Peter J. Slater. *Fundamentals of domination in graphs*, volume 208 of *Pure and applied mathematics*. Dekker, 1998.

11 Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In *CIAC*, volume 10236 of *LNCS*, pages 345–356, 2017.

12 Pallavi Jain, M. Jayakrishnan, Fahad Panolan, and Abhishek Sahu. Mixed dominating set: A parameterized perspective. In *WG*, volume 10520 of *LNCS*, pages 330–343. Springer, 2017.

13 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structurally parameterized d-scattered set. In *WG*, volume 11159 of *LNCS*, pages 292–305. Springer, 2018.

14 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discr. Applied Math.*, 264:90–117, 2019.

15 Michael Lampis. Finer tight bounds for coloring on clique-width. In *ICALP*, volume 107 of *LIPIcs*, pages 86:1–86:14, 2018.

16 James K. Lan and Gerard Jennhwa Chang. On the mixed domination problem in graphs. *TCS*, 476:84–93, 2013.

17 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018.

18 Jayakrishnan Madathil, Fahad Panolan, Abhishek Sahu, and Saket Saurabh. On the complexity of mixed dominating set. In *CSR*, volume 11532 of *LNCS*, pages 262–274. Springer, 2019.

19 Aniket Majumdar. Neighborhood hypergraphs: A framework for covering and packing parameters in graphs. *PhD thesis, Clemson University*, 1993.

20 David Manlove. On the algorithmic complexity of twelve covering and independence parameters of graphs. *Discrete Applied Mathematics*, 91(1-3):155–175, 1999.

21 Amram Meir. On total covering and matching of graphs. *JCTS B*, 24(2):164–168, 1978.

22 John W Moon and Leo Moser. On cliques in graphs. *Israel j. of Math.*, 3(1):23–28, 1965.

23 Uri N. Peled and Feng Sun. Total matchings and total coverings of threshold graphs. *Discrete Applied Mathematics*, 49(1-3):325–330, 1994.

24 M. Rajaati, Mohammad Reza Hooshmandasl, Michael J. Dinneen, and Ali Shakiba. On fixed-parameter tractability of the mixed domination problem for graphs with bounded tree-width. *Discrete Mathematics & Theoretical Computer Science*, 20(2), 2018.

25 M. Rajaati, P. Sharifani, Ali Shakiba, Mohammad Reza Hooshmandasl, and Michael J. Dinneen. An efficient algorithm for mixed domination on generalized series-parallel graphs. *CoRR*, abs/1708.00240, 2017. `arXiv:1708.00240`.

26 Mingyu Xiao and Zimo Sheng. Improved parameterized algorithms for mixed domination. In *AAIM*, volume 11640 of *LNCS*, pages 304–315. Springer, 2019.

27 Yancai Zhao, Liying Kang, and Moo Young Sohn. The algorithmic complexity of mixed domination in graphs. *Theor. Comput. Sci.*, 412(22):2387–2392, 2011.

A Polynomial Kernel for Paw-Free Editing

Eduard Eiben
Department of Computer Science, Royal Holloway, University of London, Egham, UK
eduard.eiben@rhul.ac.uk

William Lochet
Department of Informatics, University of Bergen, Norway
william.lochet@uib.no

Saket Saurabh
Institute of Mathematical Sciences, Chennai, India
Department of Informatics, University of Bergen, Norway
saket@imsc.res.in

Abstract

For a fixed graph H, the H-free Edge Editing problem asks whether we can modify a given graph G by adding or deleting at most k edges such that the resulting graph does not contain H as an induced subgraph. The problem is known to be NP-complete for all fixed H with at least 3 vertices and it admits a $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ algorithm. Cai and Cai [Algorithmica (2015) 71:731–757] showed that, assuming coNP $\not\subseteq$ NP/poly, H-free Edge Editing does not admit a polynomial kernel whenever H or its complement is a path or a cycle with at least 4 edges or a 3-connected graph with at least one edge missing. Based on their result, very recently Marx and Sandeep [ESA 2020] conjectured that if H is a graph with at least 5 vertices, then H-free Edge Editing has a polynomial kernel if and only if H is a complete or empty graph, unless coNP $\subseteq$ NP/poly. Furthermore they gave a list of 9 graphs, each with five vertices, such that if H-free Edge Editing for these graphs does not admit a polynomial kernel, then the conjecture is true. Therefore, resolving the kernelization of H-free Edge Editing for graphs H with 4 and 5 vertices plays a crucial role in obtaining a complete dichotomy for this problem. In this paper, we positively answer the question of compressibility for one of the last two unresolved graphs H on 4 vertices. Namely, we give the first polynomial kernel for Paw-free Edge Editing with $\mathcal{O}\left(k^6\right)$ vertices.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Kernelization, Paw-free graph, H-free editing, graph modification problem

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.10

Funding *William Lochet*: Received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 819416).
Saket Saurabh: Received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 819416), and Swarnajayanti Fellowship (No DST/SJF/MSA01/2017-18).

Acknowledgements The authors wish to thank the anonymous referees for helping in the presentation of the paper and pointing out some missing argument in Section 5.

1 Introduction

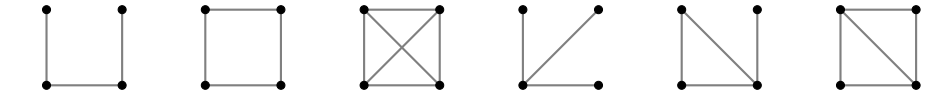
For a family of graphs $\mathcal{G}$, the general $\mathcal{G}$-Graph Modification problem asks whether we can modify a graph G into a graph in $\mathcal{G}$ by performing at most k simple operations. Typical examples of simple operations that are well-studied in the literature include vertex deletion, edge deletion, edge addition, or combination of edge deletion and addition. We call these problems $\mathcal{G}$-Vertex Deletion, $\mathcal{G}$-Edge Deletion, $\mathcal{G}$-Edge Addition, and

$\mathcal{G}$-Edge Editing, respectively. While a classical result by Lewis and Yannakakis [15] shows that $\mathcal{G}$-Vertex Deletion is NP-complete for all non-trivial hereditary graph classes, the problem seems more difficult for the Edge Modification version and to this day, no simple classification exists.

$\mathcal{G}$-Graph Modification problems have been extensively investigated for graph classes $\mathcal{G}$ that can be characterized by a finite set of forbidden induced subgraphs. We say that a graph is $\mathcal{H}$*-free*, if it does not contain any graph in $\mathcal{H}$ as an induced subgraph. For this special case, the $\mathcal{H}$-free Vertex Deletion problem is well understood. If $\mathcal{H}$ contains a graph on at least two vertices and the class of $\mathcal{H}$-free graphs is non-trivial, then all of these problems are NP-complete, but admit $c^k n^{\mathcal{O}(1)}$ algorithm [3], where c is the size of the largest graph in $\mathcal{H}$ (the algorithms with running time $f(k)n^{\mathcal{O}(1)}$ are called fixed-parameter tractable (FPT) algorithms [9, 11]). Finally, Flum and Grohe [12] showed the existence of a *kernel* with $\mathcal{O}(k^c)$ vertices, where c is again the size of the largest graph in $\mathcal{H}$. A kernel is a polynomial time preprocessing algorithm which outputs an equivalent instance of the same problem such that the size of the reduced instance is bounded by some function $f(k)$ that depends only on k. We call the function $f(k)$ the size of the kernel. It is well-known that any problem that admits an FPT algorithm admits a kernel. Therefore, for problems with FPT algorithms one is interested in polynomial kernels, i.e., kernels with the size upper bounded by a polynomial function of the parameter.

For the edge modification problems, the situation is more complicated. While all of these problems also admit $c^{2k}n^{\mathcal{O}(1)}$ time algorithm, where c is the maximum number of vertices in a graph in $\mathcal{H}$ [3], the P vs NP dichotomy is still not known. Only recently Aravind et al. [1] gave the dichotomy for the special case when $\mathcal{H}$ contains precisely one graph H [1]. From the kernelization point of view, the situation is even more difficult. The reason is that deleting or adding an edge to a graph can introduce a new copy of H and this might further propagate. Hence, we cannot use the sunflower lemma to reduce the size of the instance. Cai asked the question whether H-free Edge Deletion admits a polynomial kernel for all graphs H [2]. Kratsch and Wahlström [14] showed that this is probably not the case and gave a graph H on 7 vertices such that H-free Edge Deletion and H-free Edge Editing do not admit a polynomial kernel unless coNP $\subseteq$ NP/poly. Consequently, it was shown that this is not an exception, but rather a rule [4, 13]. Indeed the result by Cai and Cai [4] shows that H-free Edge Deletion, H-free Edge Addition, and H-free Edge Editing do not admit a polynomial kernel whenever H or its complement is a path or a cycle with at least 4 edges or a 3-connected graph with at least 2 edges missing (resp. at least 1 edge missing in the case of H-free Edge Editing). This suggests that actually the H-free edge modification problems with polynomial kernels are rather rare and only for small graphs H. Based on these observations, very recently Marx and Sandeep [16] conjectured that if H is a graph with at least 5 vertices, then H-free Edge Editing has a polynomial kernel if and only if H is a complete or empty graph, unless coNP $\subseteq$ NP/poly. Furthermore they gave a list of 9 graphs, each with 5 vertices, such that if H-free Edge Editing for all of these graphs does not admit a polynomial kernel, then the conjecture is true. For the graphs on 4 vertices the kernelization of H-free edge modification problems was open for last two graphs and their complements (see Table 1), namely paw and claw, and Cao et al. [7] conjectured that all of these problems admit polynomial kernels. In this paper, we give kernels for the first of the two remaining graphs, namely the paw[1].

[1] Independent of our work Cao et al. [6] obtained polynomial kernels for Paw-free Edge Deletion and Paw-free Edge Addition.



(a) P_4 **(b)** C_4 **(c)** K_4 **(d)** claw **(e)** paw **(f)** diamond

Figure 1 List of graphs on 4 vertices, excluding their complements.

Table 1 The kernelization results of H-free edge modification problems for H being 4-vertex graphs. Note that for a complement of H, the rows with deletion and addition are swapped, but otherwise the same results hold.

H	deletion	addition	editing
K_4	$\mathcal{O}\left(k^4\right)$ [5]	trivial	$\mathcal{O}\left(k^4\right)$ [5]
P_4	$\mathcal{O}\left(k^3\right)$ [13]	$\mathcal{O}\left(k^3\right)$ [13]	$\mathcal{O}\left(k^3\right)$ [13]
diamond	$\mathcal{O}\left(k^3\right)$ [18]	trivial	$\mathcal{O}\left(k^8\right)$ [7]
paw	$\mathcal{O}\left(k^4\right)$ [this paper]	$\mathcal{O}\left(k^3\right)$ [this paper]	$\mathcal{O}\left(k^6\right)$ [this paper]
claw	open	open	open
C_4	no [13]	no [13]	no [13]

1.1 Brief Overview of the Algorithm

Our main result is a polynomial kernel for Paw-free Edge Editing. The key to obtain the kernel is a structural theorem by Olariu [17] that states that every connected paw-free graph is either triangle-free or complete multipartite. We start our kernelization algorithm by finding greedily a maximal set of paws $P_1, \dots P_\ell$ such that for any $1 \leq i < j \leq \ell$, P_i and P_j share at most one vertex. This clearly contains at most k paws and hence at most $4k$ vertices. Let us denote the set of these vertices by S. The goal now is to bound the number of vertices in $G - S$. Bounding the number of vertices belonging to the complete multipartite components of $G - S$ is rather simple. We show that every vertex in S is adjacent to at most 1 complete multipartite component and for each multipartite component, we can reduce the size of each part as well as the number of these parts to $\mathcal{O}(k)$. The triangle-free part is trickier. The difficulty comes from the fact that instead of keeping this part of the graph triangle-free, the optimal solution might want to add some edges to make it complete multipartite. However, we argue that there is always an optimal solution that keeps the vertices at distance at least 5 from S in a triangle-free component. This structural claim allows us to look only for solutions which are not too far away from S "in some sense". Moreover, after some preprocessing of the instance, we can also show that the vertices with more than $4k + 6$ neighbors inside the triangle-free components of $G - S$ cannot end up inside a complete multipartite component. It means that we can mark the relevant vertices in triangle-free components as follows. Set $S_0 := S$ and for every $i < 5$, let S_{i+1} be the set obtained by marking for each vertex of S_i, $4k + 6$ neighbors at distance $i + 1$ from S. The size of the set of the marked vertices is then $\mathcal{O}\left(k^6\right)$. Finally, we can remove the vertices of triangle-free components which have not been marked. This is safe because these vertices are either too far from S to belong to a complete multipartite component, or every way to connect these vertices to S uses vertices with more than $4k + 6$ neighbors inside the triangle-free components of $G - S$ that cannot end up in a complete multipartite component of the reduced instance. This gives us the desired kernel.

2 Preliminaries

We assume familiarity with the basic notations and terminologies in graph theory. We refer the reader to the standard book by Diestel [10] for more information. Let us fix a graph G for the sake of this paragraph. We let $|G| = |V(G)|$ and $||G|| = |E(G)|$. For a set of pairs of vertices $A \subseteq \binom{V(G)}{2}$, we denote by $G\Delta A$ the graph whose set of vertices is $V(G)$ and set of edges is the symmetric difference of $E(G)$ and A. For a set of vertices $S \subseteq V(G)$, we denote by $G[S]$ the graph induced by G on S. We let $N_G(v)$ denote the neighborhood of a vertex $v \in V(G)$ and we omit the subscript G if the graph is clear from the context. For a set of vertices S and a vertex $v \in S$, we often refer to $N_G(v) \setminus S$ as the neighborhood of v in $G - S$. A connected component of G is a maximal, w.r.t. inclusion, set of vertices such that $G[C]$ is connected. For the sake of exposition, when speaking about a connected component C we, depended on the context, mean its set of vertices C or the graph $G[C]$ induced on these vertices. Finally, for sets $A, B \subseteq V(G)$, let $E_G(A, B) = \{ab \mid a \in A, b \in B, ab \in E(G)\}$, *i.e.*, the set of edges with one endpoint in A and the other in B. We again omit the subscript G, if the graph is clear from the context.

Parameterized Algorithms and Kernelization. For a detailed illustration of the following facts the reader is referred to [9, 11]. A *parameterized problem* is a language $\Pi \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet; the second component k of instances $(I, k) \in \Sigma^* \times \mathbb{N}$ is called the *parameter*. A parameterized problem Π is *fixed-parameter tractable* if it admits a *fixed-parameter algorithm*, which decides instances (I, k) of Π in time $f(k) \cdot |I|^{\mathcal{O}(1)}$ for some computable function f.

A *kernelization* for a parameterized problem Π is a polynomial-time algorithm that given any instance (I, k) returns an instance (I', k') such that $(I, k) \in \Pi$ if and only if $(I', k') \in \Pi$ and such that $|I'| + k' \leq f(k)$ for some computable function f. The function f is called the *size* of the kernelization, and we have a polynomial kernelization if $f(k)$ is polynomially bounded in k. It is known that a parameterized problem is fixed-parameter tractable if and only if it is decidable and has a kernelization. However, the kernels implied by this fact are usually of superpolynomial size.

A *reduction rule* is an algorithm that takes as input an instance (I, k) of a parameterized problem Π and outputs an instance (I', k') of the same problem. We say that the reduction rule is *safe* if (I, k) is a *yes*-instance if and only if (I', k') is a *yes*-instance. In order to describe our kernelization algorithm, we present a series of reduction rules.

We will need the following result describing the structure of paw-free graphs [17].

▶ **Theorem 1.** *G is a paw-free graph if and only if each connected component of G is triangle-free or complete multipartite.*

To make a clear distinction between these two cases, we will say that a graph is a complete multipartite graph if it contains at least three parts. In particular, it contains a triangle. For an instance (G, k) of Paw-free Edge Editing, we say that A is a *solution* to (G, k) if $|A| \leq k$ and $G\Delta A$ is paw-free.

3 Reduction Rules

From now on (G, k) will be an instance of Paw-free Edge Editing and we assume $k > 0$. Let us first describe two rules which can be safely applied.

▶ **Reduction Rule 1.** *If X is an independent set of $k+3$ vertices with the same neighborhood, remove a vertex $x \in X$ from the graph.*

Proof of Safeness. Suppose (G, k) is an instance of Paw-free Edge Editing and X is an independent set of $k + 3$ vertices with the same neighborhood. Let G' be the graph obtained by removing a vertex of X. We need to show that (G', k) has a solution if and only if (G, k) has one. Since G' is an induced subgraph of G, it is clear that if (G, k) has a solution, then so does (G', k). Let A be a solution to (G', k) and assume $G\Delta A$ contains a paw x_1, x_2, x_3, x_4 with x_1, x_2, x_3 being a triangle and x_4 being adjacent to x_3. Because A is a solution to (G', k), it means that one of the x_i must be the vertex x that we removed from G. Moreover, at most two of the other vertices of X belong to the paw, as x is adjacent to at least one vertex in the paw and X is an independent set. If only one other vertex of X belongs to it, consider the other $k + 1$ vertices of X which are not in the paw. They all have the same neighborhood in the paw as x, so A must contain for each of them at least one edge with the paw, or we could replace x with this vertex in the paw, which contradicts the fact that A is a solution of (G', k). However, since A is smaller than $k + 1$ we reach a contradiction. If two other vertices of X belong to the paw, then it means that $x = x_4$ and these vertices are x_1 and x_2. Moreover it means that the edge x_1x_2 must be edited as X is an independent set. In that case, consider the other k vertices of X which are not in the paw. For every $y \in X \setminus \{x, x_1, x_2\}$, the solution must contain either the edge yx_3 or at least one of the nonedges in $\{yx_1, yx_2\}$, but since $|A \setminus \{x_1x_3\}| < k$, we reach a contradiction. ◀

If Reduction Rule 1 is applicable, then we can easily find an independent set X with at least $k + 3$ vertices and the pairwise same neighborhood. This is because there are at most $|V(G)|$ different open neighborhoods of a vertex in G and for each neighborhood, we can simply pass through all vertices $v \in V(G)$ to find all vertices with the given neighborhood. Therefore, we assume from now on that (G, k) is an instance where Reduction Rule 1 cannot be applied.

Following analogous arguments for the case when X induces a complete multipartite graph with at least $k + 5$ parts, we also obtain safeness of the following rule. We note that whenever we apply Reduction Rule 2 we will always provide a suitable X and we will not require that G is irreducible w.r.t. this rule. Hence, in particular it is not required to be able to decide the existence of a suitable set X in polynomial time.

▶ **Reduction Rule 2.** *If X is a complete multipartite subgraph with $k + 5$ parts having the same neighborhood outside of X, then remove one part of X from the graph.*

Proof of Safeness. Suppose (G, k) is an instance of Paw-free Edge Editing and X is a complete multipartite subgraph with $k + 5$ parts having the same neighborhood outside of X. Let P be an arbitrary part of X and let G' be the graph obtained by removing the part P of X. We need to show that (G', k) has a solution if and only if (G, k) has one. Let A be a solution to (G', k) and assume $G\Delta A$ contains a paw x_1, x_2, x_3, x_4 with x_1, x_2, x_3 being a triangle and x_4 being adjacent to x_3. Because A is a solution to (G', k), it means that one of the x_i must belong to P. Moreover, since the vertices in P have exactly the same neighborhood in G and they form an independent set, this paw can contain at most one vertex from P. Let us call x this vertex. Since X consists of $k + 5$ parts, it means that there exists $k + 1$ parts different from P and without a vertex in this paw. However we know that every vertex in these parts has exactly the same neighborhood as x inside the paw. This means that for every vertex y in these $k + 1$ parts, the solution A contains an edge between y and a vertex in $\{x_1, x_2, x_3, x_4\} \setminus \{x\}$ or $G'[\{y, x_1, x_2, x_3, x_4\} \setminus \{x\}]$ is a paw in $G'\Delta A$. Because there are at least $k + 1$ parts of X without a vertex in the paw, it follows that either $|A| > k$ or $G'\Delta A$ is not paw-free, a contradiction with A being a solution to (G, k). ◀

Let $\mathcal{H}$ be a maximal set of paws such that any pair share at most one vertex, i.e the paws in $\mathcal{H}$ are edge and non-edge disjoint, and S the set of vertices appearing in $\mathcal{H}$. From now on we will fix the set S. The following observation is immediate from the maximality of $\mathcal{H}$.

▶ **Observation 2.** *For every vertex $v \in S$, the graph $G - (S \setminus \{v\})$ is paw-free.*

We will now introduce two new rules.

▶ **Reduction Rule 3.** *If there is a pair of adjacent vertices $s_1, s_2 \in V(G)$ with $4k + 6$ common neighbors in the triangle-free components of $G - S$, then remove the edge s_1s_2 and set $k := k - 1$.*

We remark here that while for our proof we only need to apply the above reduction rule for all the pairs s_1, s_2 in S, we can safely apply Reduction Rule 3 to all pairs of adjacent vertices. The safeness of Reduction Rule 3 is implied by the following Lemma:

▶ **Lemma 3.** *Suppose Reduction Rule 1 cannot be applied anymore and let s_1, s_2 be two adjacent vertices in G. If there are at least $4k + 6$ vertices belonging to the triangle-free components of $G - S$ adjacent to both s_1 and s_2, then either (G, k) is a* no*-instance, or every solution uses the edge s_1s_2.*

Proof. Suppose there is a solution A not using the edge s_1s_2 and let T be the set of the common neighbors of s_1 and s_2 that are not incident to any edge in A. Because $|A| \leq k$, we know that $|T| \geq 2k + 6$. Since for all $t \in T$, the vertices t, s_1, s_2 induce a triangle in $G\Delta A$, all vertices in T belong to the same complete multipartite component in $G\Delta A$. Moreover, they can only be in two different parts of this component as they belong to the triangle-free components of $G - S$. This means that $k + 3$ of vertices in T belong to the same part of a complete multipartite component of $G\Delta A$. Since vertices in T are not incident to any edge in A, they have the same neighborhood in G. Therefore, Reduction Rule 1 can be applied, which contradicts the assumptions of the lemma. ◀

▶ **Reduction Rule 4.** *If C is a complete multipartite component of $G - S$ and C_1 is a part of C with at least $3k + 3$ vertices, then remove all the edges between the other parts of C and decrease k by the number of edges removed. If this amount is greater than k, answer* no*.*

The fact that Reduction Rule 4 is safe is implied by the following Lemma:

▶ **Lemma 4.** *Suppose Reduction Rule 1 cannot be applied anymore and assume C is a complete multipartite component of $G - S$. If one part of C has at least $3k + 3$ vertices, then either (G, k) is a* no*-instance, or any solution will remove all the edges between the other parts of C.*

Proof. Let C_1 be a part of C of size at least $3k + 3$. Recall that we consider a graph to be a complete multipartite graph only if it contains at least three parts and let s_1, s_2 be two adjacent vertices of $C - C_1$. Let A be a solution to (G, k) which does not use the edge s_1s_2. A is incident to at most $2k$ vertices, so it means that at least $k + 3$ vertices of C_1 are not incident to any edge of A. Moreover, since s_1s_2 is not in A, these $k + 3$ vertices belong to the same part of a complete multipartite component of $G\Delta A$ and thus have the same neighborhood in G. This is a contradiction, as Reduction Rule 1 cannot be applied anymore. ◀

Note also that if Reduction Rules 3 and 4 can be applied, then it is possible to do it in polynomial time. From now on assume that Reduction Rules 1, 3 and 4 can not be applied.

4 Bounding the Complete Multipartite Components

The next two lemmas allow us to bound the number of vertices belonging to complete multipartite components of $G - S$.

▶ **Lemma 5.** *Let C denote a complete multipartite component of $G - S$. If $|C| \geq (3k + 3)(5k + 5)$, then Reduction Rule 2 can be applied in polynomial time.*

Proof. Because Reduction Rule 4 cannot be applied, we have that every part of C contains at most $(3k + 2)$ vertices. Suppose now that C consists of at least $5k + 5$ parts and recall that, by Observation 2, for every vertex $x \in S$ adjacent to C, $G[C \cup \{x\}]$ is paw-free and hence a complete multipartite graph. Therefore, any such vertex x is adjacent to all but at most one part of C. It follows that all but $|S| \leq 4k$ parts of C are adjacent to all vertices in $N(C) \cap S$ and thus at least $5k + 5 - |S|$ parts of C are adjacent to all the vertices of $N(C) \cap S$ and we can find the complete multipartite subgraph X of C induced on these parts in polynomial time by checking the neighborhoods of all vertices in S. Reduction Rule 2 then applies to X and since it simply remove arbitrary part of X, it can be also executed in polynomial time. ◀

▶ **Lemma 6.** *For every $s \in S$, s is adjacent to at most one complete multipartite component of $G - S$.*

Proof. Suppose $s \in S$ is adjacent to two complete multipartite components C and D. Let x be a vertex of C adjacent to s. Since C is a complete multipartite component, it contains at least 3 parts and, in particular, there exist vertices y and z in C such that x, y, z is a triangle. This implies that one of y and z has to be adjacent to s or it would yield a paw without any edge in S which is not possible by definition of $\mathcal{H}$.

Suppose now that y is adjacent to s (the case z is adjacent to s is identical). Now let d be a vertex of D adjacent to s. Because C and D are two different components of $G - S$, d cannot be adjacent to either x or y, which means that s, x, d and y form a paw without any edge or non-edge in S, a contradiction. ◀

The next section is devoted to proving that, if there exists a solution A, then we can assume that any complete multipartite component of $G\Delta A$ only contains vertices at distance 5 from S.

5 Bounding the Diameter of Relevant Vertices

Let A denote an optimal solution and suppose that, among all the optimal solutions, A is chosen so that the sum of the sizes of the multipartite components in $G\Delta A$ is minimized. In this section, C will denote a complete multipartite component of $G\Delta A$, and $C_1, C_2, \ldots, C_r$ the parts of C (see also Figure 2). Furthermore, we will split the vertices of C into *levels* depending on their distance to S. That is we say that a vertex is in the i-th level, if it is at distance i from S in G and we let L_i denote the set of all vertices in the i-th level, i.e., $L_0 = C \cap S$, $L_1 = C \cap N_G(S)$, $L_2 = C \cap N_G(N_G(S)) \setminus L_0$, and so on. For the part C_i and the level L_j, we let $C_{i,j}$ denote the subset of C_i in j-th level. That is for every $i \in [r]$ and every j such that L_j is not empty we let $C_{i,j} = C_i \cap L_j$. Finally, throughout the section for $i \in [r]$ and some level j, we will need to consider the set of all vertices in the j-th level that are not in C_i, we will denote this set $\overline{C_{i,j}}$, i.e., $\overline{C_{i,j}} = \bigcup_{t \neq i} C_{t,j} = L_j \setminus C_i$.

The goal of this section is to show that, because we chose an optimal solution that minimizes the sum of the sizes of the multipartite components in $G\Delta A$, there is no vertex in the j-th level for $j \geq 5$. Let us first show that the result follows easily when L_0 is empty.

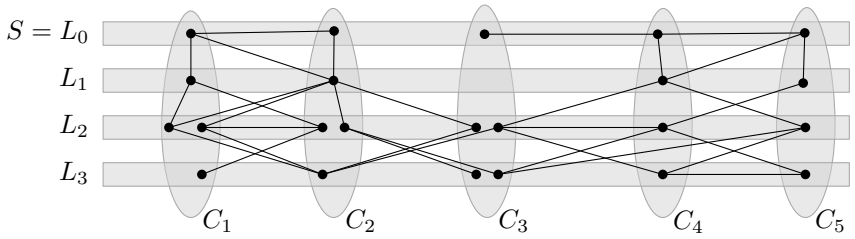


Figure 2 An example of a complete multipartite component C in $G\Delta A$ for some solution A whose vertices were in a triangle-free component of G. The edges drawn are the edges in G. $C_1, \ldots, C_5$ are parts of C, that is, in $G\Delta A$, each C_i is an independent set that is complete to $\bigcup_{j\in[5]\setminus\{i\}} C_j$. L_0, L_1, L_2, and L_3 are the levels in C. That is vertices in L_i are at the distance i from S in G.

▶ **Observation 7.** *If L_0 is empty, then C contains only vertices at distance at most 3 from S.*

Proof. Indeed, if L_0 is empty, then C contains only vertices of $G - S$. In that case, since $G - S$ is paw-free and A is an optimal solution, it follows that A does not contain any pair of vertices of C and thus that C is a complete multipartite component of $G - S$. This ends the proof as the diameter of a complete multipartite graph is 2. ◀

Therefore, from now on we assume that L_0 is not empty. In that case, we observe that it suffices to show that L_5 is empty. Indeed, if some level L_i is empty, then all the edges between the first $i - 1$ levels and the remaining levels in $G\Delta A$ are not in G and hence removing them from A gives a smaller solution that splits C into multiple paw-free components. This however contradicts the optimality of A and we get the following observation.

▶ **Observation 8.** *If for some $i \in \mathbb{N}$ is $L_i = \emptyset$, then for all $j > i$ it holds that $L_j = \emptyset$.*

The first step of our proof is the following lemma that basically says that for the set of vertices $C_{i,j}$, the number of edges between $C_{i,j}$ and the rest of C that is added by A has to be smaller than the number of such edges that already exists in G, otherwise we can isolate $C_{i,j}$ from C instead of including it in C and obtain a solution that contradicts our choice of A, because $C_{i,j}$ will not be anymore in a complete multipartite component of the solution.

▶ **Lemma 9.** *For every $j \geq 2$ and every $i \in [r]$ such that $C_{i,j}$ is not empty, if $P \subseteq A$ denotes the set of pairs of A of type xy where $x \in C_{i,j}$ and $y \in \overline{C_{i,j'}}$ with $j' \in \mathbb{N}$, then $|P| < |E_G(C_{i,j}, \overline{C_{i,j-1}} \cup \overline{C_{i,j}} \cup \overline{C_{i,j+1}})|$.*

Proof. See Figure 3 for an illustration. In order to reach a contradiction, suppose this is not the case and consider the set of pairs A' obtained from A by:

- removing all the pairs in P and
- adding $E_G(C_{i,j}, \overline{C_{i,j-1}} \cup \overline{C_{i,j}} \cup \overline{C_{i,j+1}})$.

Because $|P| \geq |E_G(C_{i,j}, \overline{C_{i,j-1}} \cup \overline{C_{i,j}} \cup \overline{C_{i,j+1}})|$, $|A'| \leq |A|$. Moreover, since $A'\Delta A$ are pairs of vertices of C, it means that $(G - C)\Delta A'$ is identical to $(G - C)\Delta A$. We will show now that $G[C]\Delta A'$ consists of one multipartite component $C - C_{i,j}$ and an independent set $C_{i,j}$. Indeed, suppose $x \in C_{i,j}$ and $y \in C \setminus C_{i,j}$, then we can show that yx is not an edge of $G[C]\Delta A'$. The proof can be done by checking the different cases:

- If $y \in C_i$, then the set of pairs of A' containing y is the same as the one in A, and we can conclude since $xy \notin G[C]\Delta A'$.
- If $y \in \overline{C_{i,j'}}$ for some j' such that $|j' - j| > 1$, then because x can only be adjacent to vertices at distance $j, j - 1$ and $j + 1$, we know that xy is not in $E(G)$ and it does not belong to A', because it is in P.

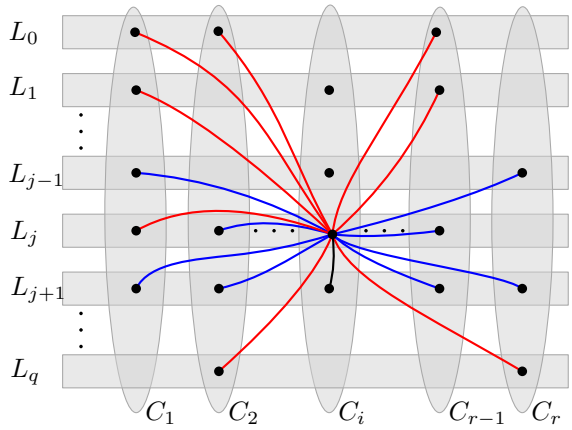


Figure 3 Illustration of Lemma 9. A complete multipartite component C of $G\Delta A$. For simplicity, every nonempty set $C_{s,t}$, $s \in [r]$ and $t \in [q]$, contains only one vertex, but this is not the case in general. The red edges are the set P, *i.e.*, the edges between $C_{i,j}$ and all the parts of C other than C_i added by A. The blue edges are $E_G(C_{i,j}, \overline{C_{i,j-1}} \cup \overline{C_{i,j}} \cup \overline{C_{i,j+1}})$, *i.e.*, all the edges between $C_{i,j}$ and all the parts of C other than C_i that are already in $E(G)$. The black edge is incident to $C_{i,j}$ in G, but its other endpoint is also in C_i, so it is in both A and the solution A' obtained from A by replacing P by $E_G(C_{i,j}, \overline{C_{i,j-1}} \cup \overline{C_{i,j}} \cup \overline{C_{i,j+1}})$ in A.

- If $y \in (\overline{C_{i,j-1}} \cup \overline{C_{i,j}} \cup \overline{C_{i,j+1}})$, then either $xy \in E(G)$ but then the pair belongs to A', or $xy \notin E(G)$ and the pair belongs to A but has been removed in A'.

Overall, A' is a solution to the problem and the complete multipartite components of $G\Delta A'$ are exactly the same as those of $G\Delta A$, except for C which is strictly smaller. This contradicts the choice of A, as $|A'| \leq |A|$. ◀

▶ **Lemma 10.** *If for some $i \in [r]$ the set $C_{i,0} \cup C_{i,1}$ is not empty, then $C_{i,j} = \emptyset$ for every $j \geq 4$.*

Proof. Suppose $C_{i,0} \cup C_{i,1}$ and $C_{i,j}$ are not empty. Because $j \geq 4$, we know that $E_G(C_{i,j}, \overline{C_{i,0}} \cup \overline{C_{i,1}} \cup \overline{C_{i,2}})$ is empty. This implies that A contains all the pairs in $C_{i,j} \times (\overline{C_{i,0}} \cup \overline{C_{i,1}} \cup \overline{C_{i,2}})$. By applying Lemma 9 to $C_{i,j}$, we deduce that

$$|C_{i,j}| \times |\overline{C_{i,0}} \cup \overline{C_{i,1}} \cup \overline{C_{i,2}}| < |E_G(C_{i,j}, \overline{C_{i,j-1}} \cup \overline{C_{i,j}} \cup \overline{C_{i,j+1}})|.$$

However,

$$|E_G(C_{i,j}, \overline{C_{i,j-1}} \cup \overline{C_{i,j}} \cup \overline{C_{i,j+1}})| \leq |C_{i,j}| \times |\overline{C_{i,j-1}} \cup \overline{C_{i,j}} \cup \overline{C_{i,j+1}}|$$

and by combining the two inequalities we obtain that

$$|\overline{C_{i,j-1}} \cup \overline{C_{i,j}} \cup \overline{C_{i,j+1}}| > |\overline{C_{i,0}} \cup \overline{C_{i,1}} \cup \overline{C_{i,2}}|. \tag{1}$$

Consider the set of pairs A', obtained from A by:
- adding $E_G(C_{i,0} \cup C_{i,1}, \overline{C_{i,0}} \cup \overline{C_{i,1}} \cup \overline{C_{i,2}})$ and
- removing all the pairs of the form xy with $x \in C_{i,0} \cup C_{i,1}$ and $y \in C_s$ for $s \neq i$.

By a very similar argument to the one of Lemma 9, we can show that A' is a solution such that $G\Delta A'$ differs from $G\Delta A$ only in the fact that $C_{i,0} \cup C_{i,1}$ has been disconnected from C. Moreover, we know that the set of pairs of the form xy with $x \in C_{i,0} \cup C_{i,1}$ and $y \in C_s$ for $s \neq i$ contains $\big((C_{i,0} \cup C_{i,1}) \times (\overline{C_{i,j-1}} \cup \overline{C_{i,j}} \cup \overline{C_{i,j+1}})\big)$. However, from (1), we can deduce that $|E_G(C_{i,0} \cup C_{i,1}, \overline{C_{i,0}} \cup \overline{C_{i,1}} \cup \overline{C_{i,2}})| < |\big((C_{i,0} \cup C_{i,1}) \times (\overline{C_{i,j-1}} \cup \overline{C_{i,j}} \cup \overline{C_{i,j+1}})\big)|$ and thus $|A'| < |A|$, which gives us a contradiction. ◀

The main implication of Lemma 10 is that, if L_j is not empty for $j \geq 4$, then A contains all the pairs $L_j \times (L_0 \cup L_1)$. Indeed, it shows that vertices in L_j and $L_0 \cup L_1$ belong to different parts and thus must be adjacent in $G\Delta A$. However, just by considering the distance to S in G, these vertices cannot be adjacent in G, and thus these pairs must be in A. This allows us to prove the following lemma.

▶ **Lemma 11.** *For every $j \geq 5$, L_j is empty.*

Proof. First note that by Observation 8, it is enough to show that, for some $j \leq 5$, the set L_j empty. Hence, for the sake of a contradiction, suppose none of $L_0, L_1, \ldots, L_5$ is empty. Now by Lemma 10, we know that the vertices in L_5 and $L_0 \cup L_1$ belong to different parts of the complete multipartite component. This implies that A contains $L_5 \times (L_0 \cup L_1)$. Consider A' the set of pairs obtained from A by:

- removing all the pairs $xy \in A$ where $xy \notin E(G)$, $x \in L_5$ and $y \in L_4$,
- adding all the edges of $E_G(L_5, L_4)$ which are not already in A, and
- removing all pairs xy with $x \in L_s$ and $y \in L_f$ for $s \leq 3$ and $f \geq 5$.

By doing so, we only disconnect $\bigcup_{s \leq 4} L_s$ from $\bigcup_{f \geq 5} L_f$ in $G\Delta A'$ compared to $G\Delta A$. This means that A' is also a solution, and by minimality of A, we have that $|A'| \geq |A|$. We can then deduce that $|L_4| \cdot |L_5| \geq E_G(L_5, L_4) \geq |L_5| \cdot |L_0 \cup L_1|$ and thus $|L_4| \geq |L_0 \cup L_1|$.

Now again by Lemma 10, we have that A contains $L_4 \times (L_0 \cup L_1)$. However, $|L_4| \geq |L_0 \cup L_1|$ so it means that $|L_0 \cup L_1|^2 \leq |L_4| \cdot |L_0 \cup L_1|$. Let A'' be the solution obtained from A by:

- removing all the pairs $xy \in A$ where $xy \notin E(G)$, $x \in L_0$ and $y \in L_1$,
- adding all the edges of $E_G(L_0, L_1)$ which are not already in A,
- removing all the pairs $xy \in A$ where $x \in L_0$ and $y \in L_i$ with $i \geq 2$, and
- removing all the pairs $xy \in A$ where $x, y \in C \setminus L_0$.

Note first that again $(G - C)\Delta A$ and $(G - C)\Delta A''$ are the same. Now consider a component D of $G[C]\Delta A''$. Since we removed all the edges between L_0 and the rest of C, D is either subset of S or a subset of $V(G) \setminus S$. In the case that D is a subset of S, then $G[D]\Delta A'' = G[D]\Delta A$ and $G[D]\Delta A''$ is an induced subgraph of the complete multipartite graph $G[C]\Delta A$ and hence either complete multipartite or triangle-free. In the case that D is a subset of $V(G) \setminus S$, then we removed from A all the pair that have one endpoint in D and the other anywhere in C (including D), so $G[D]\Delta A''$ is an induced subgraph of exactly one connected component of $G - S$. Since all components of $G - S$ are paw-free and being paw-free is a property closed under taking induced subgraphs, it follows that $G[D]\Delta A''$ is also paw-free. Therefore, we conclude that $G\Delta A''$ is paw-free and A''. It remains to show that A'' contradicts the choice of A.

The set $A'' \setminus A$ contains only edges in $E_G(L_0, L_1)$, so $|A'' \setminus A| \leq |L_0 \cup L_1|^2 \leq |L_4| \cdot |L_0 \cup L_1|$. On the other hand, by Lemma 10, the fact that G does not contain edges between L_i and L_j for $|i - j| > 1$, and by the construction of A'', we have that $A \setminus A''$ contains all the pairs xy such that $x \in L_0 \cup L_1$ and $y \in L_4 \cup L_5$. In particular, $|A \setminus A''| \geq |L_0 \cup L_1| \cdot |L_4 \cup L_5|$, but since L_5 is not empty, we have

$$|A \setminus A''| \geq |L_0 \cup L_1| \cdot |L_4 \cup L_5| > |L_0 \cup L_1| \cdot |L_4| \geq |A'' \setminus A|,$$

and it follows that $|A''| < |A|$, which contradicts the optimality of A. ◀

6 Triangle-Free Components

Before proving our main result let us prove the following lemma, which will be useful in bounding the number of vertices outside of S.

▶ **Lemma 12.** *If $x \in G$ has at least $4k+6$ neighbors belonging to triangle-free components of $G - S$, then there is no solution A such that x belongs to a complete multipartite component of $G\Delta A$.*

Proof. Let T denote the set of neighbors of x belonging to triangle-free components of $G - S$. Suppose x belongs to a complete multipartite component C of $G\Delta A$. First note that at least $2k + 6$ of the vertices of T will not be adjacent to any edge of A, which means that their neighborhood in G and $G\Delta A$ are the same and they belong to C in $G\Delta A$. Now because the vertices of T belong to triangle-free components, it means that these $2k + 6$ vertices can only belong to two different parts of this multipartite component. In particular, at least $k + 3$ of those belong to the same part and thus have the exact same neighborhood in $G\Delta A$ and thus in G. This means that Reduction Rule 1 can be applied, which is a contradiction. ◀

▶ **Lemma 13.** *Suppose (G, k) is a* yes*-instance. Then there exists a set S' of at most $(4k + 6)4k$ vertices such that if $x \notin S'$ belongs to a triangle-free component of $G - S$, then x does not belong to any triangle in G using only one vertex of $S \cup S'$. Moreover, there is a polynomial time algorithm that either finds this set or concludes that (G, k) is a* no*-instance.*

Proof. Let x be a vertex belonging to a triangle-free component C of $G - S$. Suppose that x belongs to a triangle using only one vertex s of S and another vertex y of C. Note first that C is the only component of $G - S$ adjacent to s or we would have a paw with only one vertex in S (which is impossible by Observation 2). Suppose now that $t \in C$ is adjacent to x. Then t must be adjacent to either y or s or it would yield a paw with only one vertex in S. Thus, since C is triangle free, t must be adjacent to s. The same argument would show that any vertex adjacent to t in C must be adjacent to s and thus the whole component C is adjacent to s (by symmetry of x and y).

Now let $s \in S$ and let C_s denote a triangle-free component of $G - S$ such that there exist two vertices $x, y \in C_s$ that induce a triangle with s. Note that if such a component exists, then, by the above argument, it is the unique component in $G - S$ adjacent to s, more precisely $C_s = N(s) \cap (V(G - S))$, and let us consider only the vertices in S for which such a component exists.

Let $\mathcal{M}_s$ be a maximal matching in C_s. If $\mathcal{M}_s$ consists of more than k edges, then it means that any solution A to the instance (G, k) puts s in a complete multipartite component. In particular if $|C_s| \geq 4k+6$, as $C_s \subseteq N(s)$ and $|A| \geq k$, we have that $2k+6$ of the vertices of C_s are not adjacent to any edge of A and belong to the same complete multipartite component of $G\Delta A$ as s. Moreover, these $2k + 6$ vertices can only belong to two different parts of this complete multipartite component (or we would have a triangle in C_s), and thus $k+3$ of them belong to the same part. However, since their neighborhood in G and $G\Delta A$ are identical, it means we could have applied Reduction Rule 1. Hence, if $|C_s| \geq 4k + 6$, then the solution A cannot exist and we can conclude that (G, k) is a *no*-instance. Otherwise, let C'_s be the set of the vertices of $\mathcal{M}_s$ and note that the vertices in $C_s \setminus C'_s$ induce an independent set in G. In particular, the vertices in $C_s \setminus C'_s$ are singletons in $G - (S \cup C'_s)$. and hence no vertex in $C_s \setminus C'_s$ forms a triangle with another vertex in $G - (S \cup C'_s)$.

Let $S' = \bigcup_{s \in S} C'_s$, where $C'_s = \emptyset$ if the component C_s does not exists, *i.e.*, if there is no triangle in G containing s and two vertices in a triangle-free component. By the construction of C'_s for each $s \in S$, it follows that no vertex x in a triangle-free component of $V(G) \setminus (S \cup S')$ belongs to a triangle using only one vertex of $S \cup S'$. Moreover, either $|S'| \leq |S| \cdot (4k+6) \leq (4k+6)4k$, or there is $s \in S$ such that $|\mathcal{M}_s| > k$ and $|C_s| \geq 4k+6$ and we can conclude that (G,k) is *no*-instance. ◀

7 Main Result

▶ **Theorem 14.** *Paw-free Edge Editing has a kernel on $\mathcal{O}(k^6)$ vertices.*

Proof. To ensure that the reduction rules are applied in the correct order, that is, *e.g.*, that we never apply Reduction Rule 3 if Reduction Rule 1 can be applied, we restart the algorithm from the beginning on the reduced instance whenever it is reduced according to some reduction rule. Since every reduction rule decreases either number of vertices of G or the parameter, this increases the running time at most by the factor of $|G| + k$.

Let (G,k) be an instance of Paw-free Edge Editing. The algorithm first applies Reduction Rule 1. If Reduction Rule 1 cannot be applied anymore, the algorithm computes $\mathcal{H}$ a maximal packing of edge-disjoint paws. If $\mathcal{H}$ consists of more than k paws, answer *no*. If this is not the case, let S be the set of vertices belonging to a paw of $\mathcal{H}$. As S is the union of at most k paws, $|S| \leq 4k$. Then the algorithm applies Reduction Rules 3 and 4 until either $k < 0$, in which case it answers *no*, or they cannot be applied anymore.

Because $\mathcal{H}$ is maximal, Theorem 1 implies that the components $G - S$ are either triangle-free or complete multipartite. Let C be a complete multipartite component. If $|C| \geq (3k+3)(5k+5)$, then Lemma 5 implies that the algorithm can apply Reduction Rule 2. Moreover Lemma 6 implies that the number of complete multipartite components adjacent to S is bounded by $|S|$. Overall this implies that the number of vertices contained in complete multipartite components of $G - S$ adjacent to S is bounded by $4k(3k+3)(3k+5)$, or it is possible to apply Reduction Rule 2.

By applying Lemma 13, we either find out that (G,k) is a *no*-instance or find a set S' of at most $(4k+6)4k$ vertices such that if $x \notin S'$ belongs to a triangle-free component of $G - S$, then x does not belong to any triangle in G using only one vertex of S.

Because Reduction Rule 3 cannot be applied anymore, it means that for every pair of adjacent vertices s_1, s_2 in S, the number of vertices in triangle-free components adjacent to both s_1 and s_2 is bounded by $4k+6$. This means that, if S'' denotes the set of vertices in a triangle-free component forming a triangle with 2 vertices of S, then $|S''| \leq |S|^2(4k+6)$.

Then we construct recursively sets $S_0, S_1, \ldots, S_6$ such that S_i is a subset of vertices of G at distance i from S as follows: We set $S_0 := S$. Now we proceed in 6 rounds. In the i-th round we mark, for every vertex $x \in S_{i-1}$, arbitrary $4k+6$ neighbors of x at distance i from S in G and belonging to a triangle-free component of $G - S$. Afterwards, we let S_i be the set of vertices marked in this round and proceed to the next round. Note that $|\bigcup S_i| = \mathcal{O}(k^6)$.

Let G' be the graph induced on G by S, S', S'', all the sets S_i for $i \in [6]$ and all the complete multipartite components of $G - S$ adjancent to S. Note that, by construction of S' and S'', there is no triangle in G using a vertex which is not in G'. We claim that (G',k) has a solution if and only if (G,k) has a solution. As G' is a subgraph of G, it is clear that if (G,k) has a solution, then so does (G',k). Suppose now that (G',k) has a solution A, but (G,k) does not have a solution. In particular, it implies that $G \Delta A$ is not paw-free. Because of Lemma 11, we can assume that no complete multipartite component of $G' \Delta A$ has a vertex at distance 5 from S and that A is minimal. Let x_1, x_2, x_3, x_4 form a paw in $G \Delta A$,

with x_1, x_2, x_3 being the triangle. If x_1, x_2, x_3 is a triangle of $G'\Delta A$, then x_4 is a vertex of $G - G'$ adjacent to one of these vertices, say x_1. Since x_1 is at distance less than 5 from S, it means that during the marking process x_4 was not marked for x_1 and that x_1 has more than $4k + 6$ neighbors in triangle-free components of $G' - S$. However, Lemma 12 implies that x_1 cannot belong to a complete multipartite component of $G'\Delta A$, which is a contradiction. If x_1, x_2, x_3 is not a triangle of $G'\Delta A$, then, without loss of generality, we can assume that x_1 belongs to $G - G'$, x_2 and x_3 belong to G', and the edge x_2x_3 was added by A. If x_2 and x_3 belong to a triangle-free component of $G'\Delta A$, then $A \setminus \{x_2, x_3\}$ is a smaller solution to (G', k). Therefore, x_2 and x_3 belong to a complete multipartite component of $G'\Delta A$. This means that they are at distance at most 5 from S and x_1 was not marked for both x_2 and x_3 during the marking process. Finally, this implies that both x_2 and x_3 already have more than $4k + 6$ neighbors in triangle-free components of $G' - S$ and thus cannot belong to a multipartite component of $G' - S$, a contradiction. ◀

8 Kernels for Deletion and Addition

In this section, we provide kernels for Paw-free Edge Deletion and Paw-free Edge Addition. To obtain the kernel for these problems, we observe that Reduction Rules 1–4 apply even if we are only allowed to delete respectively only allowed to add edges. This allows us to reduce the complete multipartite components. Furthermore, by deleting the edges, we cannot change a triangle-free component to a complete multipartite one and it actually suffice to keep the vertices that actually appear in triangle together with $4k + 6$ of each of such vertex, which can be bounded using Lemma 13. For the edge addition, we just observe that every connected component of G that contains a paw, and hence a triangle, has to be modified to a complete multipartite graph and we can basically conclude by Lemma 12.

▶ **Theorem 15.** *Paw-free Edge Deletion admits a kernel with $\mathcal{O}\left(k^4\right)$ vertices.*

Proof. Let (G, k) be an instance of Paw-free Edge Deletion. First note that Reduction Rules 1–4 are still safe in this context, and Lemma 12 still applies. Therefore the algorithm applies Reduction Rule 1 until it cannot be applied anymore. It then computes $\mathcal{H}$ a maximal packing of edge-disjoint paws. If $\mathcal{H}$ consists of more than k paws, answer *no*. If this is not the case, let S be the set of vertices belonging to a paw of $\mathcal{H}$. As S is the union of at most k paws, $|S| \leq 4k$. Then the algorithm apply Reduction Rules 3 and 4 until either $k < 0$, in which case it answers *no*, or they cannot be applied anymore.

Again, by possibly applying Reduction Rule 2, we can assume that the set of vertices in all the multipartite components of $G - S$ adjacent to S is smaller than $4k(3k + 3)(3k + 5)$. By applying Lemma 13, we either find out that (G, k) is a *no*-instance or find a set S' of at most $(4k + 6)4k$ vertices such that if $x \notin S'$ belongs to a triangle-free component of $G - S$, then x does not belong to any triangle in G using only one vertex of S.

Because Reduction Rule 3 cannot be applied anymore, it means that for every pair of adjacent vertices s_1, s_2 in S, the number of vertices in triangle-free components adjacent to both s_1 and s_2 is bounded by $4k + 6$. This means that, if S'' denote the set of vertices in a triangle-free component, forming a triangle with 2 vertices of S, then $|S''| \leq |S|^2(4k + 6)$.

Note also that Lemma 12 still applies, and let S_1 be the set obtained by picking for every vertex s in $S \cup S' \cup S''$, $4k + 6$ neighbors in triangle-free components of $G - S$.

Let G' be the graph induced on G by S, S', S'', S_1, as well as all the vertices on complete multipartite components of $G - S$. We want to show that (G, k) has a solution if and only if (G', k) has a solution. Let A be a solution of (G', k) and suppose $G\Delta A$ has a paw

x_1, x_2, x_3, x_4, with x_1, x_2, x_3 being a triangle and x_4 being adjacent to x_3. Because of the choice of the sets S' and S'', all the triangles of G are contained in G'. Note also that, since the solution can only remove edges, x_1, x_2, x_3 is a triangle in G. In particular, x_1, x_2, x_3 is a triangle in G' and $x_4 \notin V(G')$. This implies that $x_3 \in S \cup S' \cup S''$ and x_4 was not picked for the $4k + 6$ neighbors of x_3. In particular, this means that x_3 has $4k + 6$ neighbors which belong to a triangle-free component of $G' - S$ in G' and thus, by Lemma 12, x_3 cannot belong to a complete multipartite component of $G'\Delta A$. However, since x_1, x_2 and x_3 form a triangle in $G'\Delta A$, we reach a contradiction. ◀

▶ **Theorem 16.** *Paw-free Edge Addition admits a kernel with $\mathcal{O}\left(k^3\right)$ vertices.*

Proof. Again, Reduction Rules 1–4 are still safe in this context, with the difference for Rules 3 and 4 that, instead of removing edges and decreasing k, we can directly conclude that (G, k) is a *no*-instance. Note also that a paw-free connected component can safely be removed from the graph.

So the algorithm starts by removing all the paw-free components of G and applying Reduction Rule 1 until it cannot be applied anymore. It then computes $\mathcal{H}$ a maximal packing of edge-disjoint paws. If $\mathcal{H}$ consists of more than k paws, answer *no*. If this is not the case, let S be the set of vertices belonging to a paw of $\mathcal{H}$. As S is the union of at most k paws, $|S| \leq 4k$. From now on we can assume that Rules 3 and 4 cannot be applied.

Again, by possibly applying Reduction Rule 2, we can assume that the set of vertices in all the multipartite components of $G - S$ adjacent to S is smaller than $4k(3k + 3)(3k + 5)$.

Consider a connected component C_1 of G. This component cannot be paw-free, or the algorithm would have removed it from the graph. So let $S_1 = C_1 \cap S$ and R_1 the vertices of C_1 contained in triangle-free component of $G - S$. Because C_1 is not triangle-free, it means that any solution A to (G, k) leaves C_1 as a complete multipartite component. In particular, it implies that R_1 is smaller than $4k + 6$. Indeed, if R_1 is bigger than $4k + 6$, then $2k + 6$ vertices will have the same neighborhood in $G\Delta A$ as in G. Moreover, since R_1 is triangle-free, it means that these vertices belong to at most 2 parts of the complete multipartite component. This implies that at least $k + 3$ of these vertices belong to the same part and Rule 1 applies. Moreover, since G has at most k connected components which are not paw-free, it implies that the set of vertices contained in triangle-free components of $G - S$ is smaller than $(4k + 6)k$.

Overall, it implies that our reduced instance has size at most $4k(3k + 3)(3k + 5) + (4k + 6)k + 4k = \mathcal{O}\left(k^3\right)$, which ends the proof. ◀

9 Conclusion

In this paper we studied Paw-free Edge Editing and gave a polynomial kernel of size $\mathcal{O}\left(k^6\right)$. The only unresolved graph H on 4 vertices, for which the kernelization complexity of H-free Edge Editing problem remains open is the claw. In fact, for this problem even the kernelization complexity of H-Edge Deletion and H-Edge Addition remain open. Settling the kernelization complexity might require using the power of structure theorem of claw free graphs [8]. Thus, a natural start here could be looking at editing/deletion/addition to basic graphs, on which structure theorem of claw free graphs is built. We leave these as natural directions to pursue.

References

1. N. R. Aravind, R. B. Sandeep, and Naveen Sivadasan. Dichotomy results on the hardness of H-free edge modification problems. *SIAM J. Discrete Math.*, 31(1):542–561, 2017. doi:10.1137/16M1055797.
2. Hans L Bodlaender, Leizhen Cai, Jianer Chen, Michael R Fellows, Jan Arne Telle, and Dániel Marx. Open problems in parameterized and exact computation-iwpec 2006. *UU-CS*, 2006, 2006.
3. Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.
4. Leizhen Cai and Yufei Cai. Incompressibility of H-free edge modification problems. *Algorithmica*, 71(3):731–757, 2015. doi:10.1007/s00453-014-9937-x.
5. Yufei Cai. Polynomial kernelisation of H-free edge modification problems. Mphil thesis, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong SAR, China, 2012.
6. Yixin Cao, Yuping Ke, and Hanchun Yuan. Polynomial kernels for paw-free edge modification problems. *CoRR*, abs/2003.11273, 2020. arXiv:2003.11273.
7. Yixin Cao, Ashutosh Rai, R. B. Sandeep, and Junjie Ye. A polynomial kernel for diamond-free editing. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 10:1–10:13, 2018. doi:10.4230/LIPIcs.ESA.2018.10.
8. Maria Chudnovsky and Paul D. Seymour. Claw-free graphs. IV. decomposition theorem. *J. Comb. Theory, Ser. B*, 98(5):839–938, 2008. doi:10.1016/j.jctb.2007.06.007.
9. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
10. R. Diestel. *Graph Theory, 4th Edition*. Springer, 2012.
11. Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
12. Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.
13. Sylvain Guillemot, Frédéric Havet, Christophe Paul, and Anthony Perez. On the (non-)existence of polynomial kernels for P_ℓ-free edge modification problems. *Algorithmica*, 65(4):900–926, 2013. doi:10.1007/s00453-012-9619-5.
14. Stefan Kratsch and Magnus Wahlström. Two edge modification problems without polynomial kernels. *Discrete Optimization*, 10(3):193–199, 2013. doi:10.1016/j.disopt.2013.02.001.
15. John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.
16. Dániel Marx and R. B. Sandeep. Incompressibility of H-Free Edge Modification Problems: Towards a Dichotomy. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 72:1–72:25, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2020.72.
17. Stephan Olariu. Paw-free graphs. *Information Processing Letters*, 28(1):53–54, 1988. doi:10.1016/0020-0190(88)90143-3.
18. R. B. Sandeep and Naveen Sivadasan. Parameterized lower bound and improved kernel for diamond-free edge deletion. In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, pages 365–376, 2015. doi:10.4230/LIPIcs.IPEC.2015.365.

A General Kernelization Technique for Domination and Independence Problems in Sparse Classes

Carl Einarson
Royal Holloway, University of London, UK
einarson.carl@gmail.com

Felix Reidl
Birkbeck, University of London, UK
f.reidl@dcs.bbk.ac.uk

Abstract

We unify and extend previous kernelization techniques in sparse classes [6, 17] by defining *water lilies* and show how they can be used in bounded expansion classes to construct linear bikernels for (r, c)-Dominating Set, (r, c)-Scattered Set, Total r-Domination, r-Roman Domination, and a problem we call $(r, [\lambda, \mu])$-Domination (implying a bikernel for r-Perfect Code). At the cost of slightly changing the output graph class our bikernels can be turned into kernels. We also demonstrate how these constructions can be combined to create "multikernels", meaning graphs that represent kernels for multiple problems at once.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Dominating Set, Independence, Kernelization, Bounded Expansion

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.11

Related Version A full version of the paper is available at `https://arxiv.org/abs/2002.09028`.

1 Introduction

Dominating Set is arguably one of the touchstone for kernelization in sparse graph classes: after a linear kernel in planar graphs [1] and a polynomial kernel in graphs defined by an excluded topological minor [2, 12] results for linear kernels in bounded genus graphs [3] apex-minor-free graphs [9], H-minor-free graphs [10], and finally H-topological-minor-free graphs [11] followed in quick succession. The most general results to date are linear kernels for bounded expansion classes [6] (generalizing all aforementioned classes) and an almost-linear kernels for nowhere dense classes [14] (generalizing bounded expansion classes). These latter two results even hold for the general problem of r-Dominating Set, where a vertex dominates everything in its closed r-neighbourhood. Together with an almost-linear kernel for the related r-Independence problem [17], these results led us to the guiding question: Do the kernelization techniques developed for r-Domination/r-Independence in sparse classes carry over to related problems?

Bounded expansion classes. Nešetřil and Ossona de Mendez introduced bounded expansion classes as a generalization of classes excluding a (topological) minor and various useful notions of sparsity (*e.g.* embeddability in a surface, bounded degree). In short, a class $\mathcal{G}$ has *bounded expansion* (BE) if any minor obtained by contracting disjoint subgraphs of radius at most r in any member $G \in \mathcal{G}$ is $\nabla_r(\mathcal{G})$-degenerate, where $\nabla_r(\mathcal{G})$ is a class constant independent of G. There are various equivalent definitions for BE classes [15, 19, 18, 16], all of which have in common that they define families of graph invariants $\{f_r\}_{r\in\mathbb{N}}$ where r is a parameter governing the "depth" at which the invariant is measured. BE classes then are precisely those graph classes for which f_r is finite for every member of the class. We will not need

to work with these invariants directly, instead building on higher-level results discussed in Section 2. Consequently, we broadly refer to these invariants as *expansion characteristics*. For an in-depth discussion see [16].

A selection of problems. The commonality of the following problems is that they can be expressed via *universal neighbourhood constraints*, meaning that a solution X needs to intersect every "neighbourhood" (a slightly flexible term as we will see in the following) in at least/at most a certain value. We define an *r-dominating set* of a graph G to be any set D that satisfies $|N^r[u] \cap D| \geq 1$ for all $u \in V(G)$, where $N^r[u]$ contains all vertices at distance $\leq r$ from u. We arrive at a natural extension of the problem by replacing the right hand side of this *domination constraint* by an arbitrary constant. We call a set that satisfies the constraint $|N^r[u] \cap D| \geq c$ an *(r, c)-dominating set* and the corresponding decision problem

(r, c)-Domination parametrised by k

Input: A graph G and an integer k.
Problem: Is there a set $D \subseteq V(G)$ of size at most k such that $|N^r[v] \cap D| \geq c$ for all $v \in G$?

For $r = 1$ this problem has received some attention in the literature under the name "k-Domination" (*e.g.* [4]), for $c = 1$ we recover the above discussed r-Domination. Two other domination problems of interest, Total r-Domination and r-Roman Domination, can be found in the full version.

The problem of *independence* turns out to be closely related to that of domination. We define an *r-scattered set* of a graph G to be any set I that satisfies $|N^r[u] \cap I| \leq 1$ for all $u \in V(G)$. Note that an r-scattered set is equivalent to a *$2r$-independent set* (all vertices in I are pairwise at distance $> 2r$) and the domination/independence duality that holds in BE-classes (see below) has usually been described with this terminology. However, the natural extension to *(r, c)-scattered sets* that satisfy the constraints $|N^r[u] \cap I| \leq c$ does not correspond to independent sets. We therefore opt to speak in terms of scattered instead of independent sets, in particular, we consider the following parameterized problem:

(r, c)-Scattered Set parametrised by k

Input: A graph G and an integer k.
Problem: Is there a set $I \subseteq V(G)$, $|I| \geq k$ such that $|N^r[v] \cap I| \leq c$ for all $v \in V(G)$?

Finally, we consider the problem that arises when combining the domination- and scatter-constraints into the form $\lambda \leq |N^r[u] \cap D| \leq \mu$, which leads to the following, rather general, parameterized problem:

$(r, [\lambda, \mu])$-Domination parametrised by k

Input: A graph G and an integer k.
Problem: Is there a set $D \subseteq V(G)$, $|D| \leq k$ s.t. every $v \in G$ satisfies $\lambda \leq |N^r[v] \cap D| \leq \mu$?

$(r, [c, \infty])$-Domination is equivalent to (r, c)-Dominating Set and $(r, [0, c])$-Domination to (r, c)-Scattered Set. For $\lambda = \mu = 1$ it is equivalent to Perfect Code (see full version).

Kernelization in sparse classes. The definition of a kernel (see [5]) for a problem restricted to a certain input class demands that the output belongs to this class as well, *e.g.* a planar kernelization needs to output a planar graph. This turns out to be too restrictive for very general notions of sparseness and we are left with the choice of either outputting an annotated instance belonging to a different problem, called a *bikernel*, or to modify the graph to "simulate" the annotation in the original problem, but these modifications take the instance out of the original graph class. Here we settle for the following compromise: a

parametrised graph problem $\mathcal{P} \subseteq \mathcal{G} \times \mathbb{N}$ for a BE-class $\mathcal{G}$ admits a *BE kernel* if there is a kernelization that outputs an instance in $\mathcal{G}' \times \mathbb{N}$ with $\nabla_r(\mathcal{G}') \leq g(\nabla_r(\mathcal{G}))$ for some function g and all $r \in \mathbb{N}$. This is justified by the idea that all nice algorithmic properties stemming from $\mathcal{G}$ being BE carry over from $\mathcal{G}$ to $\mathcal{G}'$ with only changes to some constants – if other properties of the class are of primary interest (embedding in a surface, excluded minors, *etc.*) then the BE-view is simply too coarse.

Our results. Inspired by the kernelization for r-Dominating Set [6] and r-Independent Set [17] in sparse classes, we unify and extend these techniques by defining a structure we call *water lilies* and show how their existence can be used to find small *cores*, that is, subset of vertices that either are guaranteed to contain a solution (*solution core*) or that already fully represent the neighbourhood-constraints governing the problem (*constraint core*). We define and prove the existence of water lilies in BE-classes in Section 4, building on our proof of a constant-factor approximation for (r, c)-Dominating Set in BE-classes from Section 3.

In Section 5 we use water lilies to prove linear bikernels for all the above listed problems into appropriate annotated variants and how most of these bikernels can be turned into BE-kernels. Finally we demonstrate how these constructions can be combined to create "multikernels", meaning graphs that represent kernels for multiple problems at once.

As mentioned above, we only present a selection of kernels obtainable by our method and we also omit some proofs (marked with $\star$). Please see the full version of this paper[1] for more kernels and further details.

2 Notation and previous results

For a maximization problem $\mathcal{P}$ defined via universal neighbourhood constraints and a graph G we call a set $L \subseteq V(G)$ a *constraint core* if for every set $D \subseteq V(G)$ it holds that D is a solution to $\mathcal{P}$ in G already if the constraints only hold for vertices in L. Analogous, for a minimization problem $\mathcal{P}$ defined via universal neighbourhood constraints, we call a set $U \subseteq V(G)$ a *solution core* if a minimum solution to P already exists inside U. In both cases, note that $V(G)$ is always a trivial core and that a superset of any core is a core as well.

A set $D \subseteq V(G)$ is an (r, c)-*dominating set* if for every vertex $v \in V(G)$ it holds that $|N^r[v] \cap D| \geq c$. Importantly, this constraint must also hold for vertices contained in D, therefore such a set can only exist if $|N^r[v]| \geq c$ for all $v \in G$. We write $\mathbf{dom}_r^c(G)$ to denote the size of a minimum (r, c)-dominating set in G and let $\mathbf{dom}_r^c(G) = \infty$ if no such set exists. A set $I \subseteq V(G)$ is $2r$-*independent* if every pair of vertices $u, v \in I$ has distance at least $2r+1$. We write $\mathbf{ind}_{2r}(G)$ to denote the size of a maximum $2r$-independent set in G. Related, a set $I \subseteq V(G)$ is an (r, c)-*scattered set* if for all vertices $v \in G$ it holds that $|N^r[v] \cap I| \leq c$. An $(r, 1)$-scattered set is equivalent to a $2r$-independent set, but this relationship breaks down for $c > 1$. We defined $\mathbf{sct}_r^c(G)$ as the size of a maximum (r, c)-scattered set in G. In all cases, for $c = 1$ we will omit the superscript.

Important BE properties

We adapted the following results to use the notation introduced above for the sake of a unified presentation. In particular, we will be using $\mathbf{sct}_r$ instead of $\mathbf{ind}_{2r}$. The function wcol_r is one of the expansion characteristics mentioned above (see *e.g.* [19] for a definition), here it is enough to know that for every member G of a BE-class, $\mathrm{wcol}_r(G)$ is bounded by a constant for every $r \in \mathbb{N}$.

[1] Available at `https://arxiv.org/abs/2002.09028`.

▶ **Theorem 1** (Dvořák [7]). *For every graph G and integer $r \in \mathbb{N}$ it holds that* $\mathbf{sct}_r(G) \leq \mathbf{dom}_r(G) \leq \mathrm{wcol}_{2r}^2(G)\,\mathbf{sct}_r(G)$.

Dvořák recently showed an improved bound [8], we will use the above simpler expression. In the same work he also proved the following relationship between r-scattered sets and (r, c)-scattered sets (translated into our terminology):

▶ **Theorem 2** (Dvořák [8]). *For every graph G and integers $c, r \in \mathbb{N}$ it holds that* $\frac{1}{2c\,\mathrm{wcol}_{2r}(G)}\,\mathbf{sct}_r^c(G) \leq \mathbf{sct}_r(G) \leq \mathbf{sct}_r^c(G)$.

▶ **Theorem 3** (Dvořák's algorithm [7]). *For every BE class $\mathcal{G}$ and $r \in \mathbb{N}$ there exists a constant c_r^{dvrk} and a polynomial-time algorithm that computes an r-dominating set D of G and an r-scattered set $A \subseteq D$ with $|D| \leq \mathsf{c}_r^{dvrk}|A|$.*

In particular, the r-scattered set A witnesses that D is indeed a $\mathsf{c}_r^{\mathrm{dvrk}}$-approximation of a minimum r-dominating of G. This algorithm can further be modified to compute a dominating set for a specific set $X \subseteq V(G)$ only; in that case it outputs the sets A and D, $A \subseteq D \cap X$, where D dominates all of X in G and A is r-scattered in G. We will call this algorithm the *warm-start* variant since we only need to mark the vertices $V(G) \setminus X$ as already dominated and then run the original algorithm (an alternative is a small gadget construction [6]).

Given a vertex set $X \subseteq V(G)$ we call a path *X-avoiding* if its internal vertices are not contained in X. A *shortest X-avoiding path* between vertices x, y is shortest among all X-avoiding paths between x and y.

▶ **Definition 4** (r-projection). *For a vertex set $X \subseteq V(G)$ and a vertex $u \notin X$ we define the* r-projection *of u onto X as the set*

$$P_X^r(u) := \{v \in X \mid \text{there exists an } X\text{-avoiding } u\text{-}v\text{-path of length} \leq r\}$$

▶ **Definition 5** (r-shadow). *For a vertex set $X \subseteq V(G)$ and a vertex $u \notin X$ we define the* r-shadow *of u onto X as the set*

$$S_X^r(u) := \{v \in V(G) \mid \text{every } u\text{-}v\text{-path of length} \leq r \text{ has an internal vertex in } X\}$$

The shadow $S_X^r(u)$ contains precisely those vertices that are "cut off" by the set $P_X^r(u)$. We will frequently need the union of shadow and projection and therefore introduce the shorthand $SP_X^r(u) := S_X^r(u) \cup P_X^r(u)$.

Two vertices that have the same r-projection onto X do not, however, necessarily have the same shadow since the precise distance at which the projection lies might differ. To distinguish such cases, it is useful to consider the *projection profile* of a vertex to its projection:

▶ **Definition 6** (r-projection profile). *For a vertex set $X \subseteq V(G)$ and a vertex $u \notin X$ we define the* r-projection profile *of u wrt X as a function $\pi_{G,X}^r[u] \colon X \to [r] \cup \infty$ where $\pi_{G,X}^r[u](v)$ for $v \in X$ is the length of a shortest X-avoiding path from u to v if such a path of length at most r exists and ∞ otherwise.*

We say that a function $\nu \colon X \to [r] \cup \infty$ is *realized on X* (as a projection profile) if there exists a vertex $u \notin X$ for which $\nu = \pi_{G,X}^r[u]$ and we denote the set of all realized profiles by $\Pi_G^r(X)$. We will usually drop the subscript G if the graph is clear from the context. It will be convenient to define an equivalence relation that groups vertices outside of X by their projection profile. Define $u \sim_X^r v \iff \pi_X^r[u] = \pi_X^r[v]$ for pairs $u, v \in V(G) \setminus X$.

It turns out that in BE classes, the number of possible projection profiles realised on a set X is bounded linearly in the size of X.

▶ **Lemma 7** (Adapted from [6, 14]). *For every BE class $\mathcal{G}$ and $r \in \mathbb{N}$ there exists a constant c_r^{proj} such that for every $G \in \mathcal{G}$ and $X \subseteq V(G)$, the number of r-projection profiles realised on X is at most $\mathsf{c}_r^{proj}|X|$.*

In our notation this can alternatively be written as $|\Pi^r(X)| = |(V(G) \setminus X)/\sim_X^r| \leq \mathsf{c}_r^{\mathrm{proj}}|X|$. We will crucially rely on the following two results for BE classes:

▶ **Lemma 8** (Projection closure [6]). *For every BE class $\mathcal{G}$ and $r \in \mathbb{N}$ there exists a constant c_r^{projcl} and a polynomial-time algorithm that, given $G \in \mathcal{G}$ and $X \subseteq V(G)$, computes a superset $X' \supseteq X$, $|X'| \leq \mathsf{c}_r^{projcl}|X|$, such that $|P_{X'}^r(u)| \leq \mathsf{c}_r^{projcl}$ for all $u \in V(G) \setminus X'$.*

▶ **Lemma 9** (Shortest path closure [6]). *For every BE class $\mathcal{G}$ and $r \in \mathbb{N}$ there exists a constant c_r^{pathcl} and a polynomial-time algorithm that, given $G \in \mathcal{G}$ and $X \subseteq V(G)$, computes a superset $X' \supseteq X$, $|X'| \leq \mathsf{c}_r^{pathcl}|X|$, such that for all $u, v \in X$ with $dist(u, v) \leq r$ it holds that $dist_{G[X']}(u, v) = dist(u, v)$.*

It will be useful to combine the above two lemmas in the following way:

▶ **Definition 10** (Projection kernel). *Given a graph G and a set $X \subseteq V(G)$, an (r, c)-*projection kernel *of (G, X) is an induced subgraph $\hat{G}$ of G with $X \subseteq V(\hat{G})$ and the following properties:*

1. *$N_{\hat{G}}^d(v) \cap X = N_G^d(v) \cap X$ for all $v \in X$ and $d \leq r$; and*
2. *if the signature $\nu \colon X \to [r] \cup \infty$ is realized on X by p distinct vertices in G, then ν is realized by at least $\min\{c, p\}$ distinct vertices in $\hat{G}$.*

▶ **Lemma 11.** *For every BE class $\mathcal{G}$ and $c, r \in \mathbb{N}$ there exists a constant $\mathsf{c}_{r,c}^{total}$ and a polynomial-time algorithm that, given $G \in \mathcal{G}$ and $X \subseteq V(G)$, computes an (r, c)-projection kernel $\hat{G}$ of (G, X) with $|\hat{G}| \leq \mathsf{c}_{r,c}^{total}|X|$.*

Proof. We first apply Lemma 8 to X and obtain a set $X_1 \supset X$, $|X_1| \leq \mathsf{c}_r^{\mathrm{projcl}}|X|$, such that the projections of outside vertices onto X_1 have size at most $\mathsf{c}_r^{\mathrm{projcl}}$.

Next, we apply Lemma 9 to X_1 and receive a set $X_2 \supset X_1$, $|X_2| \leq \mathsf{c}_r^{\mathrm{pathcl}}|X_1|$, such that the graph $G[X_2]$ preserves short distances (less than or equal to r) between vertices in X_1. Finally, let U contain up to c representatives for every equivalence class $[u] \in V(G)/\sim_{X_1}^r$ (if the class is smaller than c we include all of it). By Lemma 7 we have that $|U| \leq c \cdot \mathsf{c}_r^{\mathrm{proj}}|X_1|$.

Construct now X_3 by taking the union $X_2 \cup U$ as well as shortest paths from every member $u \in X_2 \cup U$ to all of $P_{X_1}^r(u)$. By definition, each of these paths has length at most r and therefore contains at most $r - 1$ internal vertices. Since, by construction of X_1, $|P_{X_1}^r(u)| \leq \mathsf{c}_r^{\mathrm{proj}}$; it follows that we add at most $\mathsf{c}_r^{\mathrm{proj}}(r-1)$ vertices per vertex in $X_2 \cup U$. Taking the above bounds together, we have that $|X_3| \leq (r-1)\,\mathsf{c}_r^{\mathrm{proj}}(\mathsf{c}_r^{\mathrm{pathcl}} + c \cdot \mathsf{c}_r^{\mathrm{proj}})|X| =: \mathsf{c}_{r,c}^{\mathrm{total}}|X|$. It remains to be shown that $\hat{G} := G[X_3]$ has the desired properties.

Property 1 follows directly from the fact that already $G[X_2] \subseteq \hat{G}$ preserves short distances among vertices inside $X_1 \supseteq X$. In particular, each vertex in $X_1 \setminus X$ has the same r-projection profile onto X in G and $\hat{G}$.

To see that Property 2 holds, consider any profile ν realized on X by vertices $S \subseteq V(G) \setminus X$ in G. First consider the case $S \setminus X_1 \neq \emptyset$. Then by construction, the set U contains $\min\{c, |S \setminus X_1|\}$ vertices from $S \setminus X_1$ that realize ν in G and whose projection onto X_1 is the same in G and $\hat{G}$. Since $X_1 \supseteq X$, we conclude that their projection on X in $\hat{G}$ must be ν. By the above, the vertices in $S \cap X_1$ must have the profile ν as well. Now assume $S \subseteq X_1$, therefore no vertex outside of X_1 has the profile ν in G. As argued above, S has the profile ν in $\hat{G}$ as well, therefore $\hat{G}$ contains $|S| \geq \min\{c, |S|\}$ vertices with profile ν, as claimed. ◀

Note that the above construction implies that $\Pi_{\hat{G}}^r(X) \supseteq \Pi_G^r(X)$, however, it is not necessarily true that $\Pi_{\hat{G}}^r(X) = \Pi_G^r(X)$.

The following is a slight restatement of Theorem 4 in [13]. We emphasise that the proof by Kreutzer *et al.* is actually constructive and can be implemented to run in polynomial time.

▶ **Lemma 12** (UQW in BE classes [13]). *For every BE class $\mathcal{G}$ and* distance $d \in \mathbb{N}$ *there exists a constant c_d^{UQW} and a polynomial-time algorithm that, given $G \in \mathcal{G}$, a* size $t \in \mathbb{N}$ *and $X \subseteq V(G)$ with $|X| \geq \mathsf{c}_d^{UQW} \cdot 2^t$, computes a set S of size at most $(\mathsf{c}_d^{UQW})^2$ and $X' \subseteq X \setminus S$ of size at least t such that X' is d-scattered in $G - S$.*

3 Approximating (r, c)-Dominating Set

▶ **Theorem 13.** *Let $\mathcal{G}$ be a BE class and fix $r, c \in \mathbb{N}$. There exists a constant $\mathsf{c}_{r,c}^{cdom}$ and an algorithm that, for every $G \in \mathcal{G}$, computes in polynomial time an (r, c)-dominating set of size at most $\mathsf{c}_{r,c}^{cdom} \mathbf{dom}_r^c(G)$ or concludes correctly that G cannot be (r, c)-dominated.*

Proof. We compute a sequence of dominating sets $D_1, D_2, \ldots, D_c$ with the invariants that a) D_i (r, i)-dominates G and b) $|D_{i+1}| \leq 5\mathsf{c}_r^{\mathrm{dvrk}}\mathsf{c}_r^{\mathrm{proj}}|D_i| + \mathsf{c}_r^{\mathrm{dvrk}} \mathbf{dom}_r^{i+1}(G)$.

To start the process, let D_1 be an $\mathsf{c}_r^{\mathrm{dvrk}}$-approximate r-dominating set for G, this set clearly satisfies invariant a). We proceed in two steps to construct D_{i+1} from D_i. Build the set U_i as follows: for every projection $\mu \in \Pi^r(D_i)$ realized by an equivalence class $[v] \in (V(G) \setminus D_i)/\sim^r_{D_i}$ we pick one (arbitrary) vertex from $S^r_{D_i}(v) \setminus D_i$ and add it to U_i, if such a vertex exists. Then for every vertex $u \in D_i$ that is not $(i+1)$-dominated by $D_i \cup U_i$, we add an arbitrary vertex from $N^r[u] \setminus D_i$ to U_i (note that if no such vertex exists we conclude that G cannot be (r, c)-dominated).

By construction, the size of U_i is bounded by $|U_i| \leq |\Pi^r(D_i)| + |D_i| \leq (\mathsf{c}_r^{\mathrm{proj}} + 1)|D_i|$. Further note that every vertex in $D_i \cup U_i$ is $(r, i+1)$-dominated by $D_i \cup U_i$: due to invariant a), the set D_i (r, i)-dominates $D_i \cup U_i$ and U_i now additionally dominates itself (at least) once and, by construction, those vertices in D_i that are not yet $(r, i+1)$-dominated by D_i.

Define the set R_i to contain all vertices that are *not* $(r, i+1)$-dominated by $D_i \cup U_i$, note that in particular $N^r[R_i] \cap U_i = \emptyset$. Let $G' = G - (D_i \cup U_i)$. Apply Dvořák's warm-start algorithm to find a distance-r dominator D'_i for R_i in G' and a r-scattered set $A'_i \subseteq D'_i \cap R_i$ with $|A'_i| \leq |D'_i| \leq \mathsf{c}_r^{\mathrm{dvrk}}|A'_i|$.

▷ Claim. $|A'_i| \leq (\mathsf{c}_r^{\mathrm{proj}} + 1)|D_i| + \mathbf{dom}_r^{i+1}(G)$.

Proof. Let X be an $(r, i+1)$-dominating set of G of minimum size and assume that $|A'_i| > (\mathsf{c}_r^{\mathrm{proj}}+1)|D_i| + \mathbf{dom}_r^{i+1}(G) \geq |U_i \cup X|$. Then there exists $a \in A'_i$ such that $N^r_{G'}[a] \cap (U_i \cup X) = \emptyset$. Since X $(r, i+1)$-dominates a but $D_i \cup U_i$ does not (because $a \in R_i$) there must be at least one vertex $b \in X \cap (N^r_G[a] \setminus N^r_{G'}[a])$ that is not contained in $D_i \cup U_i$. This means that $b \in S^r_{D_i \cup U_i}(a)$ and since $N^r_G[a] \cap U_i = \emptyset$, we have that $S^r_{D_i \cup U_i}(a) = S^r_{D_i}(a)$ and therefore even $b \in S^r_{D_i}(a)$. But then, since $b \notin D_i \cup U_i$, we could have added b to U_i during the first construction phase in order to dominate the class $[a]$. The existence of a leads us to a contradiction and we conclude that $|A'_i| \leq (\mathsf{c}_r^{\mathrm{proj}} + 1)|D_i| + \mathbf{dom}_r^{i+1}(G)$. ◁

Finally, construct the set $D_{i+1} = D'_i \cup D_i \cup U_i$. Since D'_i r-dominates R_i which, by construction, were the only vertices not yet $(r, i+1)$-dominated by $D_i \cup U_i$, we conclude that D_{i+1} is indeed an $(r, i+1)$-dominating set of G; thus invariant a) is preserved. To see that invariant b) holds, let us bound the size of D_{i+1}:

$$\begin{aligned}
|D_{i+1}| &\leq |D'_i| + |D_i| + |U_i| \leq \mathsf{c}_r^{\mathrm{dvrk}}|A'_i| + |D_i| + (\mathsf{c}_r^{\mathrm{proj}} + 1)|D_i| \\
&\leq \mathsf{c}_r^{\mathrm{dvrk}}(\mathsf{c}_r^{\mathrm{proj}} + 1)|D_i| + \mathsf{c}_r^{\mathrm{dvrk}} \mathbf{dom}_r^{i+1}(G) + (\mathsf{c}_r^{\mathrm{proj}} + 2)|D_i| \\
&\leq 5\mathsf{c}_r^{\mathrm{dvrk}}\mathsf{c}_r^{\mathrm{proj}}|D_i| + \mathsf{c}_r^{\mathrm{dvrk}} \mathbf{dom}_r^{i+1}(G).
\end{aligned}$$

Resolving the recurrence provided by this inequality, we finally obtain the bound $|D_c| \leq (5\mathsf{c}_r^{\mathrm{dvrk}}\mathsf{c}_r^{\mathrm{proj}})^{c+1} \mathbf{dom}_r^c(G)$, and the claim follows with $\mathsf{c}_{r,c}^{\mathrm{cdom}} := (5\mathsf{c}_r^{\mathrm{dvrk}}\mathsf{c}_r^{\mathrm{proj}})^{c+1}$. ◀

4 Water lilies

▶ **Definition 14** (Water lily). *A* water lily *of* radius r, depth $d \leq r$ *and* adhesion c *in a graph G is a tuple (R, C) of disjoint vertex sets with the following properties:*

- *C is r-scattered in $G - R$,*
- *$N^r_{G-R}[C]$ is (d, c)-dominated by R in G.*

We call R the roots*, C the* centres*, and the sets $\{N^r_{G-R}[x]\}_{x \in C}$ the* pads *of the water lily. A water lily is* uniform *if all centres have the same d-projection onto R,* e.g. *$\pi^d_R[x]$ is the same function for all $x \in C$. The* ratio *of a water lily is any guaranteed lower bound on $|C|/|R|$.*

The following lemma lies at the heart of our unification of previous techniques [6, 14, 17]. It streamlines the construction of BE-kernels considerably, as we will see in the following section.

▶ **Lemma 15.** *For every BE class $\mathcal{G}$ and $c, r, d \in \mathbb{N}$, $d \leq r$, there exist constants $\mathsf{c}^{scale}_{c,r,d}$, $\mathsf{c}^{margin}_{c,r,d}$, $\mathsf{c}^{base}_{r,d}$ with the following property: for every $G \in \mathcal{G}$ which has an (r, c)-dominating set, $t \in \mathbb{N}$ and $A \subseteq V(G)$ with $|A| \geq \mathsf{c}^{scale}_{c,r,d} \cdot (\mathsf{c}^{base}_{r,d})^t \cdot \mathbf{dom}^c_d(G)$ there exists a uniform water lily (R, C), $C \subseteq A$, with depth d, radius r, adhesions c and with $|R| \leq \mathsf{c}^{margin}_{c,r,d}$, $|C| \geq t$. Moreover, such a water lily can be computed in polynomial time.*

Proof. Given G, we use Theorem 13 to compute a (d, c)-dominating set D' of size at most $\mathsf{c}^{\text{cdom}}_{r,c} \cdot \mathbf{dom}_d(G)$ in polynomial time or conclude that no such set exits. Afterwards, we compute the $(r + d)$-projection closure D of D', by Lemma 8 we have that $|D| \leq \mathsf{c}^{\text{projcl}}_{r+d}|D'|$ and thus $|D| \leq \mathsf{c}^{\text{projcl}}_{r+d}\mathsf{c}^{\text{cdom}}_{r,c}\,\mathbf{dom}_d(G)$. Let $A'' := A \setminus D$, we will later choose $\mathsf{c}^{\text{scale}}_{c,r,d}$ so that A'' is still large enough for the following arguments to go through.

Define the equivalence relation $\sim_D$ over A'' via $a \sim_D a' \iff \pi^{r+d}_D[a] = \pi^{r+d}_D[a']$. By Lemma 7, the number of classes in $A''/\sim_D$ is bounded by $\mathsf{c}^{\text{proj}}_{r+d}|D|$; by an averaging argument we have at least one class $[a] \in A''/\sim_D$ of size $\big|[a]\big| \geq |A''|/(\mathsf{c}^{\text{proj}}_{r+d}|D|) \geq (|A| - |D|)/(\mathsf{c}^{\text{proj}}_{r+d}|D|)$.

Let R''' be $P^{r+d}_D(a)$, i.e. the $(r + d)$-projection of $[a]$'s members on D. By our earlier application of Lemma 8 we have that $|R'''| = |P^{r+d}_D(a)| \leq \mathsf{c}^{\text{projcl}}_{r+d}$. Again, we will choose $\mathsf{c}^{\text{scale}}_{c,r,d}$ large enough to apply Lemma 12 with distance r and size $\mathsf{c}^{\text{proj}}_d|R'''|t$ to the set $[a]$ and receive a subset $A' \subseteq [a]$ of size at least $\mathsf{c}^{\text{proj}}_d(\mathsf{c}^{\text{UQW}}_r + \mathsf{c}^{\text{projcl}}_{r+d}) \cdot t$ and a set $R'' \subseteq V(G) \setminus A'$, $|R''| \leq \mathsf{c}^{\text{UQW}}_r$, such that A' is r-scattered in $G - R''$. Let $R' := R'' \cup R'''$, by the above bounds on R'' and R''' it follows that $|R'| \leq \mathsf{c}^{\text{UQW}}_r + \mathsf{c}^{\text{projcl}}_{r+d}$. By Lemma 7 and the fact that $|A'| \geq \mathsf{c}^{\text{proj}}_d(\mathsf{c}^{\text{UQW}}_r + \mathsf{c}^{\text{projcl}}_{r+d}) \cdot t \geq |\Pi^d(R')| \cdot t$ there exists a set $C \subseteq A'$ of size at least t such that all members of C have the same d-projection onto R'.

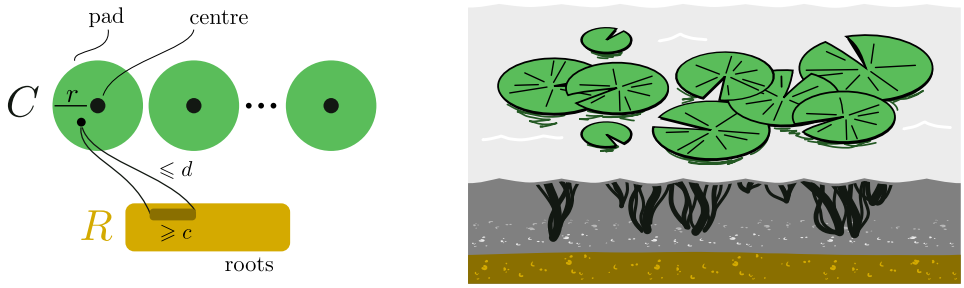


Figure 1 Schematic of a water lily (R, C) with radius r, depth d and adhesion c. Removing the "tangled" roots R creates disjoint r-neighbourhoods around C which we imagine like lily pads floating on a pond.

We construct the set R from R' as follows: for every projection profile $\mu \in \Pi^d(R')$ realized by a class $[u] \in N^r_{G-R'}[C]/\sim^d_{R'}$ we add $\max\{0, c - |P^d_{R'}(u)|\}$ vertices from the shadow $S^d_{R'}(u) \cap D'$. Since D' (d,c)-dominates all of G, such vertices must exist. By construction, $|R| \leq c|R'|$ and R (c,d)-dominates all of $N^r_{G-R'}[C]$ and thus in particular $N^r_{G-R}[A']$. Note further that all vertices we added lie inside $S^{r+d}_{R'}[C]$, therefore the projection profiles of C are not changed by this operation (all paths of length at most $r+d$ from C to vertices in R/R' pass through R'). We conclude that the uniformity condition holds on (R, C). This construction also provides us with the bound $|R| \leq c(\mathsf{c}^{\mathrm{UQW}}_r + \mathsf{c}^{\mathrm{projcl}}_{r+d}) =: \mathsf{c}^{\mathrm{margin}}_{c,r,d}$.

Finally, let us determine a value for $\mathsf{c}^{\mathrm{scale}}_{c,r,d}$ that suffices for the above construction to go through. In order to apply Lemma 12, we need that $|[a]| \geq \mathsf{c}^{\mathrm{UQW}}_r \cdot 2^{\mathsf{c}^{\mathrm{proj}}_d(\mathsf{c}^{\mathrm{UQW}}_r + \mathsf{c}^{\mathrm{projcl}}_{r+d})\cdot t}$, accordingly we need that $(|A| - |D|)/(\mathsf{c}^{\mathrm{proj}}_{r+d}|D|) \geq \mathsf{c}^{\mathrm{UQW}}_r \cdot 2^{\mathsf{c}^{\mathrm{proj}}_d(\mathsf{c}^{\mathrm{UQW}}_r + \mathsf{c}^{\mathrm{projcl}}_{r+d})\cdot t}$, which in particular holds if we ensure that $|A|/(2\mathsf{c}^{\mathrm{proj}}_{r+d}\mathsf{c}^{\mathrm{projcl}}_{r+d}\mathsf{c}^{\mathrm{cdom}}_{r,c}\,\mathbf{dom}^c_d(G)) \geq \mathsf{c}^{\mathrm{UQW}}_r \cdot 2^{\mathsf{c}^{\mathrm{proj}}_d(\mathsf{c}^{\mathrm{UQW}}_r + \mathsf{c}^{\mathrm{projcl}}_{r+d})\cdot t}$. Hence setting $\mathsf{c}^{\mathrm{scale}}_{c,r,d} = 2\mathsf{c}^{\mathrm{proj}}_{r+d}\mathsf{c}^{\mathrm{projcl}}_{r+d}\mathsf{c}^{\mathrm{cdom}}_{r,c}\mathsf{c}^{\mathrm{UQW}}_r$ and $\mathsf{c}^{\mathrm{base}}_{r,d} = 2^{\mathsf{c}^{\mathrm{proj}}_d(\mathsf{c}^{\mathrm{UQW}}_r + \mathsf{c}^{\mathrm{projcl}}_{r+d})}$ suffices. ◀

We can impose even more structure on a water lily in the following sense: let us define a *pad signature* as a function $\sigma\colon C \to \Sigma^*$ (for some alphabet Σ) that can be computed by a polynomial-time algorithm receiving the following inputs:

- The depth d, radius r and adhesion c of the water lily;
- the centre a, its pad $N^r_{G-R}[a]$, the roots R;
- the subgraph $G[R \cup N^r_{G-R}[a]]$ alongside potential vertex/edge labels from the host graph G.

We say that σ is *bounded* if the size of its image can be bounded by a constant.

Every pad signature σ gives rise to an equivalence relation $\sim_\sigma \subseteq C \times C$ for a water lily (R, C) via $a \sim_\sigma a' \iff \sigma(a) = \sigma(a')$. Note that if σ is bounded, then $\sim_\sigma$ has finite index. A water lily is *σ-uniform* if all its centres belong to the same equivalence class under $\sim_\sigma$; or alternatively if all centres have the same image under σ. For a bounded signature σ, we find a $\sim_\sigma$-uniform water lily of ratio τ by first finding a water lily (R', C') with ratio $p \cdot \tau$, where p is an upper bound on the image of σ, and then return R' together with the largest class in $C'/\sim_\sigma$. Accordingly:

▶ **Corollary 16.** *For every BE class $\mathcal{G}$, $c, r, \tau \in \mathbb{N}$ and pad signature σ with finite index there exists a constant $\mathsf{c}^{lily} = \mathsf{c}^{lily}_{c,2r,r,\tau,\sigma}$ with the following property: for every $G \in \mathcal{G}$ which has an (r,c)-dominating set and $A \subseteq V(G)$ with $|A| \geq \mathsf{c}^{lily} \cdot \mathbf{dom}^c_d(G)$ there exists a σ-uniform water lily (R, C), $C \subseteq A$, $|R| \leq \mathsf{c}^{lily}$, of depth r, radius $2r$, adhesion c and ratio τ. Moreover, such a water lily can be computed in polynomial time.*

Let us define a particular bounded pad signature that will be useful in the remainder: let $\nu(a) := \big(\{\pi^d_R[x] \mid x \in N^i_{G-R}(a)\} \mid 0 \leq i \leq r\big)$, where the right-hand side is to be understood as encoded in a string by some suitable scheme. Two centres are equivalent under $\sim_\nu$ if they have the same projection-types at the same distance (though potentially at different multiplicities) inside their respective pads. Since $|R|$ has constant size according to Lemma 15 and there are at most $\mathsf{c}^{\mathrm{proj}}_d|R|$ possible projection profiles according to Lemma 7, the image of ν has size at most $r^{\mathsf{c}^{\mathrm{proj}}_d|R|} \leq r^{\mathsf{c}^{\mathrm{proj}}_d \mathsf{c}^{\mathrm{lily}}}$ and therefore ν is a bounded pad signature.

We will sometimes combine ν with a finite number of vertex labels that arise during the construction of bikernels. If vertices are labelled by $f\colon V(G) \to \Sigma$ for some finite alphabet Σ, then we understand ν to be the above equivalence relation further refined by the equivalence relation $u \sim_f v \iff f(u) = f(v)$.

5 Bikernels into annotated problems

We show in the following that a range of problems over hereditary BE-classes admit linear bikernels in the same class (see the full version for r-Roman Domination and Total r-Domination). The target problem in all three cases is a suitable annotated version of the original problem, which we define just ahead of each proof.

Annotated (r,c)-Domination parametrised by k
Input: A graph G, a set $L \subseteq V(G)$ and an integer k.
Problem: Is there a set $D \subseteq V(G)$ of size at most k such that $|N^r[v] \cap D| \geq c$ for all $v \in L$?

▶ **Theorem 17.** *(r,c)-Dominating Set over a hereditary BE-class $\mathcal{G}$ admits a linear bikernel into Annotated (r,c)-Dominating Set over the same class $\mathcal{G}$. Moreover, the resulting graph is an (r,c)-projection kernel of the original graph.*

Proof. Let (G,k) be an input where G is taken from a BE class. As a first step, we deal with the case $\mathbf{dom}^{\mu}_r(G)$ large by computing an (r,μ)-dominating set using the algorithm from Theorem 13. If it returns a solution larger than $\mathsf{c}^{\mathrm{cdom}}_{r,c}k$, we conclude that $\mathbf{dom}^{\mu}_r(G) > k$ in which case we return a trivial no-instance. Otherwise, we show that (r,c)-Dominating Set admits a linear constraint core and then show how to construct a BE-kernel from that core.

▷ Claim. (r,c)-Dominating Set has a linear constraint core in BE classes.

Proof. Let $L \subseteq V(G)$ be a constraint core of G with $|L| \geq \mathsf{c}^{\mathrm{lily}}_{c,2r,r,2}\,\mathbf{dom}^{c}_r(G)$. By Corollary 16, we can find in polynomial time a uniform water lily (R,C), $C \subseteq L$, $|R| \leq \mathsf{c}^{\mathrm{lily}}$ of depth r, radius $2r$, adhesion c and ratio 2. Let $a \in C$ be an arbitrary centre, we claim that $L \setminus \{a\}$ is still a constraint core, that is, every set that (r,c)-dominates $L \setminus \{a\}$ will also (r,c)-dominate a.

To that end, let D be a minimum (r,c)-dominating set and define $D' := D \setminus N^r_{G-R}[C]$. If D' (r,c)-dominates any part of C, it dominates all of C (and therefore a) as (R,C) is uniform. Thus assume that D' does not (r,c)-dominate C. Consider the case where a set $S \subseteq D \cap N^r_{G-R}[C]$ exists such that every vertex in S dominates more than one vertex in C. If $|S| \geq c$ then S alone already (r,c)-dominates all of C and thus in particular a. In all remaining cases, every set $N^r_{G-R}[a']$, $a' \in C$ must contain at least one vertex from D and we conclude that $|D \setminus D'| \geq |C| \geq 2|R|$. Let $\tilde{D} := D' \cup R$, we claim that $\tilde{D}$ is an (r,c)-dominating set of G. Simply note that the only vertices that are not (r,c)-dominated by D' lie inside $N^{2r}_{G-R}[C]$ – but this is precisely the set that is (r,c)-dominated by R. We arrive at a contradiction since $|D| = |D \setminus D'| + |D'| \geq 2|R| + |D'| > |R| + |D'| \geq |\tilde{D}|$ and we assumed D to be minimum. Thus $L \setminus \{a\}$ is a constraint core for (r,c)-Dominating Set in G. We iterate this procedure until $|L| < \mathsf{c}^{\mathrm{lily}}_{c,2r,r,2}\,\mathbf{dom}^{c}_r(G)$ and end up with a linear constraint core. ◁

In the following, let $L \subseteq V(G)$ be a constraint core for (G,k) with $|L| \leq \mathsf{c}^{\mathrm{lily}}\,\mathbf{dom}^{c}_r(G)$ and let $O = V(G) \setminus L$. If $|L| > \mathsf{c}^{\mathrm{lily}}k$, we can conclude that $k > \mathbf{dom}^{c}_r(G)$ and output a trivial no-instance, thus assume from now on that $|L| \leq \mathsf{c}^{\mathrm{lily}}k$. We apply Lemma 11 with $X = L$ and r, c as here to obtain a projection kernel $\hat{G}$ with $|\hat{G}| \leq \mathsf{c}^{\mathrm{total}}_{r,c}|L| = O(k)$ which a) preserves $\leq r$-neighbourhoods in L and b) realizes every r-projection onto L that is realized p times in G at least $\min\{c,p\}$ times. We claim that (G,k) is equivalent to the annotated instance $(\hat{G},L,k)$.

Assume that D is an (r,c)-dominating set of G, clearly it is also a solution to the annotated instance (G,L,k). Partition D into $D_L = D \cap L$ and $D_O = D \setminus L$. Consider $x \in D_O$ and note that $|[x] \cap D_O| < c$ for the r-neighbourhood class $[x] \in O/\sim^r_L$ since otherwise we could remove a vertex from $[x] \cap D_O$ from D and still (r,c)-dominate all of L. With this observation, construct the set $\hat{D}_O$ as follows: for every vertex $x \in D_O$ we include $|[x] \cap D_O|$

vertices from $O \cap V(\hat{G})$ in $\hat{D}_O$, by property b) of the projection kernel $\hat{G}$ we know that at least c such vertices are available. Then the set $\hat{D} := D_L \cup \hat{D}_O$ (r,c)-dominates all of L in $\hat{G}$, by property a) of $\hat{G}$, and we are done. In the other direction, let $\hat{D}$ be an (r,c)-dominator of L in $\hat{G}$. By property a) and b) of $\hat{G}$ the set $\hat{D}$ therefore also (r,c)-dominates L in G, and since L is a constraint core of G it then (r,c)-dominates all of G. We conclude that $(\hat{G}, L, k)$ is equivalent to (G,k) and $|\hat{G}| = O(k)$. ◀

Annotated (r,c)-Scattered Set parametrised by k

Input: A graph G, a set $U \subseteq V(G)$ and an integer k.

Problem: Is there a set $I \subseteq U$ of size at least k such that $|N^r[v] \cap I| \leq c$ for all $v \in V(G)$?

The following proof makes use of the pad equivalence $\sim_\nu$ defined in Section 4: recall two centres u, v of a water lily (R,C) satisfy $u \sim_\nu v$ if they have the same projection-types onto R at the same distance (for distances smaller than the lily's depth) inside their respective pads.

▶ **Theorem 18.** *(r,c)-Scattered Set over a hereditary BE-class $\mathcal{G}$ admits a linear bikernel into Annotated (r,c)-Scattered Set over the same class $\mathcal{G}$. Moreover, the resulting graph is an (r,c)-projection kernel of the original graph.*

Proof. Let (G,k) be an instance of (r,c)-Scattered Set where G is taken from a BE class. As a first step, we deal with the case that $\mathbf{sct}^c_r(G)$ is large. We compute an $\mathsf{c}^{\mathrm{dvrk}}_r$-approximate r-dominating set D using Theorem 3. If $|D| > \mathsf{c}^{\mathrm{dvrk}}_r \operatorname{wcol}^2_{2r}(G) \cdot k$, we conclude by Theorems 1 and 2 that $\mathbf{sct}^c_r(G) \geq \mathbf{sct}_r(G) > k$ and we output a trivial yes-instance. Otherwise, assume $|D| \leq \mathsf{c}^{\mathrm{dvrk}}_r \operatorname{wcol}^2_{2r}(G) \cdot k$ and define $\mathsf{c}^{\mathrm{lily}} := \mathsf{c}^{\mathrm{lily}}_{1,2r,r,2,\nu}$. We first show that (r,c)-Scattered Set admits a linear solution core.

▷ Claim. (r,c)-Scattered Set has a linear solution core in BE classes.

Proof. Let $U \subseteq V(G)$ be solution core of G with $|U| \geq \mathsf{c}^{\mathrm{lily}} \mathbf{dom}_r(G)$. Using Corollary 16, we find in polynomial time a ν-uniform water lily (R,C), $C \subseteq U$, $|R| \leq \mathsf{c}^{\mathrm{lily}}$ of depth r, radius $2r$, adhesion 1 and ratio 2. Let $a \in C$ be an arbitrary centre, we claim that $U \setminus \{a\}$ is still a solution core, i.e. there exists an optimal (r,c)-scattered set that does not contain a.

To that end, let I be a minimum (r,c)-scattered set and assume $a \in I$. We claim that there exists an (r,c)-scattered set I' of the same size which excludes a. First observe that every vertex that lives in a pad $N^{2r}[a']$, $a' \in C$, has at least c neighbours in R at distance $\leq r$. Therefore $|N^{2r}_{G-R}[C] \cap I| \leq |R|$ as otherwise we would find a vertex in R whose r-neighbourhood contains more than c vertices of I. Since $|C| \geq 2|R|$ there are at least $|R|$ centres $C' \subseteq C$ such that their pads $N^{2r}_{G-R}[C']$ do not intersect I. Since (R,C) is uniform and $a \in I$, we know that $|N^r[a'] \cap I| = |N^r[a] \cap I| < c$ for every centre $a \in C$.

Take $a' \in C'$ and let $I' := I \setminus \{a\} \cup \{a'\}$. To see that I' is (r,c)-scattered, consider any vertex $u' \in N^r[a']$ (note that vertices at distance $> r$ from a' are not affected by the exchange of a by a'). By ν-uniformity, there exists a vertex $u \in N^r[a]$ with $\pi^r_R[u] = \pi^r_R[u']$. In particular, $P^r_R(u) \cup S^r_R(u) = P^r_R(u') \cup S^r_R(u')$; therefore $(N^r[u] \cap I) \setminus \{a\} = (N^r[u'] \cap I') \setminus \{a'\}$ and we conclude that $|N^r[a'] \cap I'| \leq c$. It follows that $U \setminus \{a\}$ is a solution core. We iterate the above procedure until $|U| \leq \mathsf{c}^{\mathrm{lily}} \mathbf{dom}^c_r(G)$ and end up with a linear solution core. ◁

In the following, let $U \subseteq V(G)$ be a solution core for (G,k) with $|U| \leq \mathsf{c}^{\mathrm{lily}} \mathbf{dom}_r(G) \leq \mathsf{c}^{\mathrm{lily}}|D| = O(k)$. We apply Lemma 11 with $X = U$ and r, c as here to obtain a projection kernel $\hat{G}$ with $|\hat{G}| \leq \mathsf{c}^{\mathrm{total}}_{r,c}|U| = O(k)$ that a) preserves $\leq r$-neighbourhoods in U and b) realizes every r-projection onto U that is realized p times in G at least $\min\{c,p\}$ times. Since distances in $\hat{G}[U]$ are as in $G[U]$, it is easy to see that any set $I \subseteq U$ is (r,c)-scattered in $\hat{G}$ iff it is (r,c)-scattered in G. Since U is further a solution core for G, we conclude that (G,k) is equivalent to the annotated instance $(\hat{G}, U, k)$. ◀

ANNOTATED $(r, [\lambda, \mu])$-DOMINATION parametrised by k

Input: A graph G, sets $L, U \subseteq V(G)$ and an integer k.

Problem: Is there a set $D \subseteq U$ of size at most k such that $|N^r[v] \cap D| \geq \lambda$ for all $v \in L$ and $|N^r[v] \cap D| \leq \mu$ for all $v \in V(G)$?

▶ **Theorem 19.** *$(r, [\lambda, \mu])$-DOMINATION over a hereditary BE-class $\mathcal{G}$ admits a linear bikernel into ANNOTATED $(r, [\lambda, \mu])$-DOMINATION over the same class $\mathcal{G}$. Moreover, the resulting graph is an (r, c)-projection kernel of the original graph.*

Proof. Since the cases where either $\mu = \infty$ or $\lambda = 0$ are equivalent to (r, c)-DOMINATING SET or (r, c)-SCATTERED SET and thus covered by Theorems 17 and 18, we here only consider the case of $\lambda \neq 0$ and $\mu \neq \infty$. Note that any solution to the problem is in particular an (r, μ)-dominating set. As a first step, we therefore deal with the case that $\mathbf{dom}_r^\mu(G)$ is too large by computing an (r, μ)-dominating set using the algorithm described in Theorem 13. If the algorithm returns a solution larger than $\mathsf{c}_{r,c}^{\mathrm{cdom}} k$, we conclude that $\mathbf{dom}_r^\mu(G) > k$ and therefore that (G, k) must be a no-instance; in which case we output a trivial no-instance. Otherwise, let $\hat{D}$ be the resulting (r, c)-dominating set.

Let (G, L, U, k) be an instance of ANNOTATED $(r, [\lambda, \mu])$-DOMINATION with $L = U = V(G)$. Clearly, (G, L, U, k) is equivalent to (G, k). In the following, we gradually reduce the size of L and U while maintaining this equivalence. To that end, we will use the pad signature ν which is to be understood to take the "vertex labels" L, U into account.

Assume that $|L| > (\mathsf{c}^{\mathrm{lily}} + 1)|\hat{D}|$ with $\mathsf{c}^{\mathrm{lily}} := \mathsf{c}_{r,2r,\mu+1,\nu}^{\mathrm{lily}}$. Then, using $\hat{D}$ in the construction used in the proof of Lemma 15, we find a ν-uniform water lily (R, C) with $C \subseteq L \setminus \hat{D}$ of depth r, radius $2r$ and ratio $(\mu + 1)$.

▷ Claim. Let $a \in C$. Then the instances (G, L, U, k) and $(G, L \setminus \{a\}, U, k)$ are equivalent.

Proof. Any solution for (G, L, U, k) is also a solution to $(G, L \setminus \{a'\}, U, k)$, therefore we only have to show the opposite direction. Let D be a solution for $(G, L \setminus \{a\}, U, k)$. Since $R \subseteq L \cap U$, the set D can intersect at most $\mu|R|$ pads or otherwise we would violate an upper constraint for at least one of the vertices in R. It follows that at least $|R|$ pads of (R, C) cannot contain any vertex of D; let the centres of these pads be $C' \subseteq C$. Choose $a' \in C'$ distinct from a (since $|C'| \geq |R| \geq \lambda > 1$ such a vertex exists). Note that $a' \in L$, therefore $|N^r[a'] \cap D| \geq \lambda$. But since $N_{G-R}^r[a'] \cap D = \emptyset$, these solution vertices must lie in $SP_R^r(a')$. Now simply observe that, by uniformity of (R, C), $SP_R^r(a) = SP_R^r(a')$ and therefore $|N^r[a'] \cap D| \geq |SP_R^r(a) \cap D| \geq \lambda$. Accordingly, D is also a solution for (G, L, U, k). ◁

We repeat the above procedure until $|L \setminus \hat{D}| \leq \mathsf{c}^{\mathrm{lily}} k$. Now assume that $|U \setminus (L \cup \hat{D})| > \mathsf{c}^{\mathrm{lily}} k$ and let (R, C) be a ν-uniform water lily with $C \subseteq U \setminus (L \cup \hat{D})$ of depth r, radius $2r$ and ratio $(\mu + 1)|R|$.

▷ Claim. Let $a \in C$. Then the instances (G, L, U, k) and $(G, L, U \setminus \{a\}, k)$ are equivalent.

Proof. By construction of (R, C), every vertex $x \in N_{G-R}^{2r}[C]$ is (r, μ)-dominated by $R \cap \hat{D}$. Importantly, $R \cap \hat{D} \subseteq R \cap U$, therefore any solution D of (G, L, U, k) can intersect $N^r[R]$ in at most $\mu|R|$ vertices. In particular, at most $\mu|R|$ pads of (R, C) can contain vertices of D, let us call the centres of these empty pads $C' \subseteq C$.

If $a \notin D$, clearly D is a solution of $(G, L, U \setminus \{a\}, k)$ and there is nothing to prove. Assume therefore that $a \in D$. Let $a' \in C'$ be an arbitrary centre of an empty pad. We claim that $D' := D \setminus \{a\} \cup \{a'\}$ is a solution to $(G, L, U \setminus \{a\}, k)$. To that end, consider any vertex $x \in N^r[a] \cup N^r[a']$, we will show that D' fulfils any constraints associated with x.

▶ Case 1. $x \in N^r_{G-R}[a]$. By ν-uniformity, there exists a vertex $x' \in N^r_{G-R}[a']$ such that $SP^r_{G-R}(x) = SP^r_{G-R}(x')$ and x' is contained in L (U) iff x is contained in L (U). For the special case that $x = a$ we let $x' = a'$. Assume $x \in L$, then $x' \in L$ and accordingly $|N^r[x'] \cap D| \geq \lambda$. Since $N^{2r}_{G-R}[a'] \cap D = \emptyset$, we have that $N^r[x'] \cap D = SP^r_R(x') \cap D = SP^r_R(x') \cap D' = SP^r_R(x) \cap D'$, therefore $|N^r[x] \cap D'| = |N^r[x'] \cap D| \geq \lambda$ and the lower-bound constraint for x is satisfied by D'. If $x \in R$, simply note that $|N^r[x] \cap D'| \leq |N^r[x] \cap D| \leq \mu$, hence the upper-bound constraint for x is satisfied by D'.

▶ Case 2. $x \in N^r_{G-R}[a']$. Again, by ν-uniformity, there exists a vertex $\hat{x} \in N^r_{G-R}[a]$ such that $SP^r_{G-R}(x) = SP^r_{G-R}(\hat{x})$ and $\hat{x}$ is contained in L (U) iff x is contained in L (U). For the special case that $x = a'$ we let $\hat{x} = a$. If $x \in L$, simply note that $|N^r[x] \cap D'| \geq |N^r[x] \cap D| \geq \lambda$, hence the lower-bound constraint for x is satisfied by D'. Assume $x \in R$. Then $\hat{x} \in R$ and accordingly $|N^r[\hat{x}] \cap D| \leq \mu$. More specifically, since $a \in N^r[\hat{x}] \cap D$, we know that $|SP^r_R[\hat{x}] \cap D| \leq \mu - 1$. Because $N^r[x] \cap D' = (SP^r_R[x] \cap D') \cup \{a'\} = (SP^r_R[\hat{x}] \cap D) \cup \{a'\}$ we conclude that $|N^r[x] \cap D'| \leq \mu$ and the upper-bound constraint for x is satisfied by D'.

▶ Case 3. $x \in SP^r_R(a) = SP^r_R(C)$. Simply note that by uniformity $|N^r[x] \cap D| = |N^r[x] \cap D'|$ and therefore D' satisfies all constraints for x.

Therefore D' is indeed a solution for $(G, L, U \setminus \{a\}, k)$ of equal size and we conclude that the instances (G, L, U, k) and $(G, L, U \setminus \{a\}, k)$ are equivalent, as claimed. ◁

We repeat the above procedure until $|U \setminus (L \cup \hat{D})| \leq \mathsf{c}^{\text{lily}} k$ and end up with an instance (G, L, U, k) which is equivalent to our initial instance (G, k) and further satisfies $|L| \leq \mathsf{c}^{\text{lily}} k$ and $|U| \leq |L| + |\hat{D}| + |U \setminus (L \cup \hat{D})| \leq (2\mathsf{c}^{\text{lily}} + \mathsf{c}^{\text{cdom}}_{r,c})k$.

Finally, let us construct the bikernel from this annotated instance. Note that, by construction, $L \subseteq U$. Let $\hat{U}$ be the shortest-path closure of U in G as per Lemma 9, then $|\hat{U}| \leq \mathsf{c}^{\text{pathcl}}_r |U|$ and $\hat{G} := G[\hat{U}]$ preserves all distances up to length r between vertices in U. In particular, $N^r_{\hat{G}}[v] \cap U = N^r_G[v] \cap U$. Since the annotated instance asks for solutions contained entirely in U and $L \subseteq U$, we conclude that the instance (G, L, U, k) and $(\hat{G}, L, U, k)$ are equivalent, therefore the latter is also equivalent to (G, k) which finally proves the claim. ◀

If we sacrifice the constraint to construct a (bi)kernel that is contained in the same hereditary graph class, we are able to construct BE-kernels by reducing from the annotated problem back into the original problems. In the following constructions, we usually tried to minimize the increase in the parameter k, not the increase of the expansion characteristics of the class.

▶ **Theorem 20.** *(r, c)-Dominating Set admits a linear BE-kernel.*

Proof. For an instance (G, k) of (r, c)-Dominating Set, where G is taken from a BE class, we first construct a bikernel $(\hat{G}, L, k)$ of Annotated (r, c)-Dominating Set according to Theorem 17. Recall that $\hat{G}$ is an (r, c)-projection kernel of (G, L).

First consider $r \geq 2$. We construct G' from $\hat{G}$ by adding new vertices $a_1, \ldots, a_c, b_1, b_2, b_3$ to the graph. We connect every a_i, $1 \leq i \leq c$ to both b_1 and b_2; then connect b_1 to every vertex in $O := V(\hat{G}) \setminus L$ via a path of length $r - 1$ and connect b_2 to b_3 by such a path as well. From the construction it is clear that G' has size $O(k)$, we are left with proving that the two instances (G, k) and $(G', k + c)$ are equivalent.

Assume that D' is a minimum (r, c)-dominating set for G' of size $\leq k + c$. By a simple exchange argument, we can assume that D' contains all vertices a_i in order to (r, c)-dominate b_3. These vertices already (r, c)-dominate all of O and the paths leading from b_1 to O. As such, we can assume that an optimal solution D' does not contain internal vertices

of those paths (otherwise we might as well exchange an internal vertex for the path's endpoint in O). Then the set $\hat{D} := D' \setminus \{a_1, \dots a_c\}$ has size at most k and (r,c)-dominates all of L; thus $\hat{D}$ in particular is a solution to $(\hat{G}, L, k)$.

In the other direction, assume that $\hat{D}$ is a minimum solution for $(\hat{G}, L, k)$, that is, $\hat{D}$ (r,c)-dominates L in $\hat{G}$. Let $D' := \hat{D} \cup \{a_1, \dots, a_c\}$, it is easy to see that D' (r,c)-dominates G' and has size $|D'| = |D| + c$. For $r = 1$ we modify the construction as follows: we add vertices $a_1, \dots, a_c, b$ and connect all a_i to $O \cup \{b\}$. The argument for why the resulting instance is equivalent is very similar to the case $r \geq 2$ and we omit it here.

We conclude that $(\hat{G}, L, k)$ and $(G', k + c)$ are indeed equivalent, and thus also to (G, k). It is only left to show that the construction of G' increased the expansion characteristics by some arbitrary function independent of $|G|$. Simply note that we can construct G' from G by adding $c + 3$ apex-vertices (which increases the expansion characteristics only by an additive constant) and then remove or subdivide edges incident to them (which does not increase the expansion characteristics). ◀

▶ **Theorem 21.** *(r,c)-Scattered Set admits a linear BE-kernel.*

Proof. Let (G, k) be an input of (r,c)-Scattered Set where G is taken from a BE class. We first construct the annotated bikernel $(\hat{G}, U, k)$ according to Theorem 18 and then construct G' from $\hat{G}$ by adding vertices $a_1, a_2, b_1, \dots, b_c$ and edges $a_2 b_i$ for all $1 \leq i \leq c$. We further connect a_1 to all vertices in $O := V(\hat{G}) \setminus U$ via paths of length r and to a_2 via a path of length $r - 1$ (for $r = 1$ we identify a_1 and a_2). It is is clear that G' has size $O(k)$, we are left to prove that the instances $(\hat{G}, U, k)$ and $(G', k + c)$ are equivalent.

First, consider a maximal (r,c)-scattered set I' in G'. Since $O \cup \{b_1, \dots, b_c\} \subset N^r[a_1]$ we may assume, by a simple exchange argument, that $\{b_1, \dots, b_c\} \subseteq I'$. Accordingly, $O \cap I' = \emptyset$ and $I := I' \setminus \{b_1, \dots, b_c\}$ is an (r,c)-scattered set contained entirely in U. Therefore I is (r,c)-scattered in $\hat{G}$ as well and $|I| = |I'| + c$.

In the other direction, assume that $\hat{I} \subseteq U$ is a maximal (r,c)-scattered set in $\hat{G}$. Then $N^r_{G'}[a_1] \cap I = \emptyset$ and we can add up to c vertices from $N^r[a_1]$ to I. Since the vertices b_i all lie at distance $2r$ from O, we conclude that $I' := I \cup \{b_1, \dots, b_c\}$ is indeed (r,c)-scattered in G' and $|I'| = |I| + c$. We conclude that the instances $(\hat{G}, U, k)$ and $(G', k + c)$ are equivalent and hence (G, k) and $(G', k + c)$ are as well. The argument why the expansion characteristics only increase by a constant are similar to the arguments in Theorem 20. ◀

▶ **Theorem 22 (⋆).** *Total r-Domination, r-Roman Domination, and r-Perfect Code admit a linear BE-kernel.*

6 Multikernels

The following results are applicable to *e.g.* planar graphs or graph classes defined by an excluded minor of minimum degree two. In the following, let $\mathbf{dom}_r^{\text{total}}(G)$ denotes the total r-domination number and $\mathbf{dom}_r^{\text{roman}}(G)$ the r-Roman domination number of G. We will also write $\mathbf{dom}_r(G, L)$, $\mathbf{dom}_r^{\text{total}}(G, L)$, and $\mathbf{dom}_r^{\text{roman}}(G, L)$ for the annotate domination numbers (where only the set $L \subseteq V(G)$ has to be dominated).

▶ **Theorem 23 (⋆).** *Let $\mathcal{G}$ be a hereditary graph class that is further closed under adding pendant vertices. Given a graph $G \in \mathcal{G}$ and an integer r we can compute in polynomial time a graph $G' \in \mathcal{G}$ and an integer c with the following properties:*

- $|G'| = O(\mathbf{dom}_r(G)) = O(\mathbf{dom}_r^{total}(G)) = O(\mathbf{dom}_r^{roman}(G))$,
- $\mathbf{dom}_r(G') = \mathbf{dom}_r(G) + c$, $\mathbf{dom}_r^{total}(G') = \mathbf{dom}_r^{total}(G) + c$ *and*
- $\mathbf{dom}_r^{roman}(G') = \mathbf{dom}_r^{roman}(G) + 2c$.

Recall that an r-scattered set is equivalent to a $2r$-independent set and in particular that $\mathbf{sct}_r(G) = \mathbf{ind}_{2r}(G)$.

▶ **Theorem 24** ($\star$). *Let $\mathcal{G}$ be a hereditary graph class that is further closed under adding pendant vertices. Given a graph $G \in \mathcal{G}$ and integers $\lambda \leq \mu$ we can compute in polynomial time a graph $G' \in \mathcal{G}$ and integers $c_\lambda, \ldots, c_\mu$ with the following properties:*

- $|G'| = O(\mathbf{dom}_\lambda(G))$,
- *for all $\lambda \leq r \leq \mu$ it holds that $\mathbf{dom}_r(G') = \mathbf{dom}_r(G) + c_r$ and $\mathbf{ind}_{2r}(G') = \mathbf{ind}_{2r}(G) + c_r$.*

7 Conclusion

We defined the notion of *water lilies* and showed that in BE-classes these structures can be used to compute linear-sized cores, bikernels, and BE-kernels. These constructions are almost universal, to the point were we can combine them into "multikernels". It stands to reason that there might be a general formulation for these types of kernels. As a technical step, we also prove that (r, c)-Dominating Set admits a constant-factor approximation in BE-classes.

We are certain that our techniques directly translate to nowhere dense classes but leave this endeavour as future work. Given that the problems treated here all have constraints whose boundaries form intervals, we ask whether the following artificial problem admits a polynomial kernel in BE-classes: find a set D of size at most k such that $|N^r[v] \cap D| \notin \{0, 2\}$.

References

1. Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *J. ACM*, 51(3):363–384, 2004. doi:10.1145/990308.990309.
2. Noga Alon and Shai Gutner. Kernels for the dominating set problem on graphs with an excluded minor. *Electron. Colloquium Comput. Complex.*, 15(066), 2008. URL: http://eccc.hpi-web.de/eccc-reports/2008/TR08-066/index.html.
3. Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. *J. ACM*, 63(5):44:1–44:69, 2016. doi:10.1145/2973749.
4. Mustapha Chellali, Odile Favaron, Adriana Hansberg, and Lutz Volkmann. k-domination and k-independence in graphs: A survey. *Graphs Comb.*, 28(1):1–55, 2012. doi:10.1007/s00373-011-1040-3.
5. Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.
6. Pål Grønås Drange, Markus Sortland Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, Felix Reidl, Fernando Sanchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar. Kernelization and sparseness: the case of dominating set. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016*, volume 47 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.31.
7. Zdeněk Dvořák. Constant-factor approximation of the domination number in sparse graphs. *Eur. J. Comb.*, 34(5):833–840, 2013. doi:10.1016/j.ejc.2012.12.004.
8. Zdeněk Dvořák. On distance-dominating and-independent sets in sparse graphs. *Journal of Graph Theory*, 91(2):162–173, 2019. doi:10.1002/jgt.22426.
9. Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010*, pages 503–510. SIAM, 2010. doi:10.1137/1.9781611973075.43.

10 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (connected) dominating set on H-minor-free graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*, pages 82–93. SIAM, 2012. doi:10.1137/1.9781611973099.7.

11 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Kernels for (connected) dominating set on graphs with excluded topological minors. *ACM Trans. Algorithms*, 14(1):6:1–6:31, 2018. doi:10.1145/3155298.

12 Shai Gutner. Polynomial kernels and faster algorithms for the dominating set problem on graphs with an excluded minor. In *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Revised Selected Papers*, volume 5917 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2009. doi:10.1007/978-3-642-11269-0_20.

13 Stephan Kreutzer, Michał Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. The generalised colouring numbers on classes of bounded expansion. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016*, volume 58 of *LIPIcs*, pages 85:1–85:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.85.

14 Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Polynomial kernels and wideness properties of nowhere dense graph classes. *ACM Trans. Algorithms*, 15(2):24:1–24:19, 2019. doi:10.1145/3274652.

15 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion i. decompositions. *Eur. J. Comb.*, 29(3):760–776, 2008. doi:10.1016/j.ejc.2006.07.013.

16 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

17 Michał Pilipczuk and Sebastian Siebertz. Kernelization and approximation of distance-r independent sets on nowhere dense graphs. *arXiv preprint*, 2018. arXiv:1809.05675.

18 Felix Reidl, Fernando Sánchez Villaamil, and Konstantinos S. Stavropoulos. Characterising bounded expansion by neighbourhood complexity. *Eur. J. Comb.*, 75:152–168, 2019. doi:10.1016/j.ejc.2018.08.001.

19 Xuding Zhu. Colouring graphs with bounded generalized colouring number. *Discret. Math.*, 309(18):5562–5568, 2009. doi:10.1016/j.disc.2008.03.024.

Parameterized Complexity of Directed Spanner Problems

Fedor V. Fomin
Department of Informatics, University of Bergen, Norway
Fedor.Fomin@uib.no

Petr A. Golovach
Department of Informatics, University of Bergen, Norway
Petr.Golovach@uib.no

William Lochet
Department of Informatics, University of Bergen, Norway
William.Lochet@uib.no

Pranabendu Misra
Max Planck Institute for Informatics, Saarbrücken, Germany
pmisra@mpi-inf.mpg.de

Saket Saurabh
Institute of Mathematical Sciences, HBNI, Chennai, India
Department of Informatics, University of Bergen, Norway
saket@imsc.res.in

Roohani Sharma
Max Planck Institute for Informatics, Saarbrücken, Germany
rsharma@mpi-inf.mpg.de

Abstract

We initiate the parameterized complexity study of minimum t-spanner problems on directed graphs. For a positive integer t, a multiplicative t-spanner of a (directed) graph G is a spanning subgraph H such that the distance between any two vertices in H is at most t times the distance between these vertices in G, that is, H keeps the distances in G up to the distortion (or stretch) factor t. An additive t-spanner is defined as a spanning subgraph that keeps the distances up to the additive distortion parameter t, that is, the distances in H and G differ by at most t. The task of Directed Multiplicative Spanner is, given a directed graph G with m arcs and positive integers t and k, decide whether G has a multiplicative t-spanner with at most $m-k$ arcs. Similarly, Directed Additive Spanner asks whether G has an additive t-spanner with at most $m-k$ arcs. We show that

- Directed Multiplicative Spanner admits a polynomial kernel of size $\mathcal{O}(k^4t^5)$ and can be solved in randomized $(4t)^k \cdot n^{\mathcal{O}(1)}$ time,
- Directed Additive Spanner is W[1]-hard when parameterized by k even if $t = 1$ and the input graphs are restricted to be directed acyclic graphs.

The latter claim contrasts with the recent result of Kobayashi from STACS 2020 that the problem for undirected graphs is FPT when parameterized by t and k.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Graph spanners, directed graphs, parameterized complexity, kernelization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.12

Funding *Fedor V. Fomin*: Supported by the Research Council of Norway via the project MULTIVAL (grant no. 263317).
Petr A. Golovach: Supported by the Research Council of Norway via the project MULTIVAL (grant no. 263317).
William Lochet: Received funding from European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant no. 819416).

Saket Saurabh: Received funding from European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant no. 819416), and Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.

1 Introduction

Given a (directed) graph G, a *spanner* is a spanning subgraph of G that approximately preserves distances between the vertices of G. Graph spanners were formally introduced by Peleg and Schäffer in [14] (see also [15]). Originally, the concept was introduced for constructing network synchronizers [15]. However, graph spanners have a plethora of theoretical and practical applications in various areas like efficient routing and fast computing of shortest paths in networks, distributed computing, robotics, computational geometry and biology. We refer to the recent survey of Ahmed et al. [1] for the introduction to graph spanners and their applications.

We are interested in the classical *multiplicative* and *additive* graph spanners in unweighted graphs. Let G be a (directed) graph. For two vertices $u, v \in V(G)$, $\mathrm{dist}_G(u, v)$ denotes the *distance* between u and v in G, that is, the number of edges (arcs, respectively, for the directed case) of a shortest (u, v)-path. Let t be a positive integer. It is said that a spanning subgraph H of G is a *multiplicative t-spanner* if $\mathrm{dist}_H(u, v) \leq t \cdot \mathrm{dist}_G(u, v)$ for every two vertices $u, v \in V(G)$, i.e., H approximates distances in G within factor t. A spanning subgraph H of G is called an *additive t-spanner* if $\mathrm{dist}_H(u, v) \leq \mathrm{dist}_G(u, v) + t$ for every $u, v \in V(G)$, that is, H approximates the distances in G within the additive parameter t. The standard task in the graph spanner problems is, given an allowed distortion parameter t, find a sparsest t-spanner, i.e., a spanner with the minimum number of edges. We consider the parameterized versions of this task:

Multiplicative Spanner parameterized by $k + t$

Input:	A (directed) graph G and integers $t \geq 1$ and $k \geq 0$.
Task:	Decide whether there is a multiplicative t-spanner H with at most $\lvert E(G)\rvert - k$ edges (arcs, respectively).

and

Additive Spanner parameterized by $k + t$

Input:	A (directed) graph G and nonnegative integers t and k.
Task:	Decide whether there is an additive t-spanner H with at most $\lvert E(G)\rvert - k$ edges (arcs, respectively).

Informally, the task of these problems is to decide whether we can delete at least k edges (arcs, respectively, for the directed case) in such a way that all the distances in the obtained graph are "t-close" to the original ones.

Previous work. We refer to [1] for the comprehensive survey of the known results and mention here only these that directly concern our work. First, we point that the considered graph spanner problems are computationally hard. It was already shown by Peleg and Schäffer in [14] that deciding whether an undirected graph G has a multiplicative t-spanner with at most ℓ edges is NP-complete even for fixed $t = 2$. In fact, the problem is NP-complete for every fixed $t \geq 2$ [2]. Moreover, for every $t \geq 2$, it is NP-hard to approximate the minimum number of edges of a multiplicative t-spanner within the factor $c \log n$ for some

$c > 1$ [10]. The same complexity lower bounds for directed graphs were also shown by Cai [2] and Kortsarz [10]. Additive t-spanners for undirected graphs were introduced by Liestman and Shermer in [11, 12]. In particular, they proved in [12], that for every fixed $t \geq 1$, it is NP-complete to decide whether a graph G admits an additive t-spanner with at most ℓ edges. It was shown by Chlamtác et al. [4] that for every integer $t \geq 1$ and any constant $\varepsilon > 0$, there is no polynomial-time $2^{\log^{1-\varepsilon}}/t^3$-approximation for the minimum number of edges of an additive t-spanner unless $\mathsf{NP} \subseteq \mathsf{DTIME}(2^{\mathrm{polylog(n)}})$.

The aforementioned hardness results make it natural to consider these spanner problems in the parameterized complexity framework. The investigation of Multiplicative Spanner and Additive Spanner on undirected graphs was initiated by Kobayashi in [8] and [9]. In [8], it was proved that Multiplicative Spanner admits a polynomial kernel of size $\mathcal{O}(k^2t^2)$. For Additive Spanner, it was shown in [9] that the problem can be solved in time $2^{\mathcal{O}((k^2+kt)\log t)} \cdot n^{\mathcal{O}(1)}$, that is, the problem is FPT when parameterized by k and t.

Our results. We initiate the study of Multiplicative Spanner and Additive Spanner on directed graphs and further refer to them as Directed Multiplicative Spanner and Directed Additive Spanner, respectively. We show that Directed Multiplicative Spanner admits a kernel of size $\mathcal{O}(k^4t^5)$. We complement this result by observing that the problem can be solved in $(4t)^k \cdot n^{\mathcal{O}(1)}$ time by a Monte Carlo algorithm with false negatives. Then we prove that Directed Additive Spanner becomes much harder on directed graphs by showing that the problem is W[1]-hard even when $t = 1$ and the input graphs are restricted to be directed acyclic graphs (DAGs).

Organization of the paper. In Section 2, we introduce basic notions used in the paper. In Section 3, we prove that Directed Multiplicative Spanner admits a polynomial kernel and sketch an FPT algorithm. In Section 4, we show hardness for Directed Additive Spanner. We conclude in Section 5 by stating some open problems.

2 Preliminaries

Parameterized Complexity and Kernelization. We refer to the recent books [5, 6, 7] for the detailed introduction. In the Parameterized Complexity theory, the computational complexity is measured as a function of the input size n of a problem and an integer *parameter* k associated with the input. A parameterized problem is said to be *fixed-parameter tractable* (or FPT) if it can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$ for some function f. A *kernelization* algorithm for a parameterized problem Π is a polynomial algorithm that maps each instance (I, k) of Π to an instance (I', k') of Π such that

(i) (I, k) is a yes-instance of Π if and only if (I', k') is a yes-instance of Π, and
(ii) $|I'| + k'$ is bounded by $f(k)$ for a computable function f.

Respectively, (I', k') is a *kernel* and f is its *size*. A kernel is *polynomial* if f is polynomial. It is common to present a kernelization algorithm as a series of *reduction rules*. A reduction rule for a parameterized problem is an algorithm that takes an instance of the problem and computes in polynomial time another instance that is more "simple" in a certain way. A reduction rule is *safe* if the computed instance is equivalent to the input instance.

Graphs. Recall that an undirected graph is a pair $G = (V, E)$, where V is a set of vertices and E is a set of unordered pairs $\{u, v\}$ of distinct vertices called *edges*. A directed graph $G = (V, A)$ is a pair, where V is a set of vertices and A is a set of ordered pairs (u, v)

of distinct vertices called *arcs*. Note we do not allow loops and multiple arcs (that are irrelevant for distances). We use $V(G)$ and $E(G)$ ($A(G)$, respectively) to denote the set of vertices and the set of edges (set of arcs, respectively) of G. For a (directed) graph G and a subset $X \subseteq V(G)$ of vertices, we write $G[X]$ to denote the subgraph of G induced by X. For a set of vertices S, $G - S$ denotes the (directed) graph obtained by deleting the vertices of S, that is, $G - S = G[V(G) \setminus S]$; for a vertex v, we write $G - v$ instead of $G - \{v\}$. Similarly, for a set of edges (arcs, respectively) S (an edge or arc e, respectively), $G - S$ ($G - e$, respectively) denotes the graph obtained by the deletion of the elements of S (the deletion of e, respectively). A (directed) graph H is a spanning subgraph of G if $V(G) = V(H)$. We write $P = v_1 \cdots v_k$ to denote a *path* with the vertices $v_1, \ldots, v_k$ and the edges (arcs, respectively) $\{v_1, v_2\}, \ldots, \{v_{i-k}, v_k\}$; v_1 and v_k are the *end-vertices* of P and we say that P is an (v_1, v_k)*-path*. The *length* of a path is the number of edges (arcs, respectively) in the path. Also $A(P)$ denotes the arc set of the path P. For a (u, v)-path P_1 and a (v, w)-path P_2, we denote by $P_1 \circ P_2$ the *concatenation* of P_1 and P_2. We use similar notation for walks; the difference that the vertices of a walk $W = v_1 \cdots v_k$ are not required to be distinct and a walk may go through the same edges (arcs, respectively) several times. Notice that the concatenation of two paths is a walk but not necessarily a path. For two vertices $u, v \in V(G)$, $\mathrm{dist}_G(u, v)$ denotes the *distance* between u and v in G, that is, the length of a shortest (u, v)-path; we assume that $\mathrm{dist}_G(u, v) = +\infty$ if there is no (u, v)-path in G. Clearly, $\mathrm{dist}_G(u, v) = \mathrm{dist}_G(v, u)$ for undirected graphs but this not always the case fro directed graphs. Let t be a positive integer. It is said that a spanning subgraph H of G is a *multiplicative t-spanner* if $\mathrm{dist}_H(u, v) \leq t \cdot \mathrm{dist}_G(u, v)$ for every $u, v \in V(G)$. A spanning subgraph H of G is called an *additive t-spanner* if $\mathrm{dist}_H(u, v) \leq \mathrm{dist}_G(u, v) + t$ for every $u, v \in V(G)$.

3 Directed multiplicative t-spanners

In this section, we consider Directed Multiplicative Spanner. We show that the problem admits a polynomial kernel and then complement this result by obtaining an FPT algorithm. These results are based on *locality* of multiplicative spanners in the sense of the following folklore observation.

▶ **Observation 1.** *Let t be a positive integer. A spanning subgraph H of a directed graph G is a multiplicative t-spanner if and only if for every arc $(u, v) \in A(G)$, there is a (u, v)-path in H of length at most t.*

Let t be a positive integer and let G be a directed graph. For an arc $a = (u, v)$ of G, we say that a (u, v)-path P is a *t-detour* for a if the length of P is at most t and P does not contain a. By Observation 1, to solve Directed Multiplicative Spanner for (G, t, k), it is necessary and sufficient to identify k arcs that have t-detours that do not contain selected arcs. Then H can be constructed by deleting these arcs.

3.1 Polynomial kernel for Directed Multiplicative Spanner

In this subsection, we show that Directed Multiplicative Spanner admits a polynomial kernel.

▶ **Theorem 2.** *Directed Multiplicative Spanner has a kernel of size $\mathcal{O}(k^4 t^5)$.*

Proof. Let (G, t, k) be an instance of Directed Multiplicative Spanner. Clearly, if $k = 0$, then (G, t, k) is a yes-instance, and our algorithm returns a trivial yes-instance in this case. We assume from now that $k > 0$.

We say that $a \in A(G)$ is *t-good* if G has a t-detour for a. Let S be the set of t-good arcs. Clearly, S can be constructed in polynomial time by making use of Dijkstra's algorithm. We follow the idea of Kobayashi [8] for constructing a polynomial kernel for undirected case and show that if S is sufficiently big, then (G, t, k) is a yes-instance of Directed Multiplicative Spanner.

▷ Claim 3. If $|S| \geq \frac{1}{2}k(t+1)((k-1)t+2)$, then (G, t, k) is a yes-instance of Directed Multiplicative Spanner.

Proof of Claim 3. Let $|S| \geq \frac{1}{2}k(t+1)((k-1)t+2)$. For every $a \in S$, let P_a be a t-detour for a.

Let $S_0 = \emptyset$. For $i = 1, \ldots, k$, we iteratively construct sets of arcs $S_1, \ldots, S_k$ such that

$$S_0 \subset S_1 \subset \cdots \subset S_k \subseteq S$$

and sets of arcs R_i such that $R_i \subseteq S_i \setminus S_{i-1}$ and $|R_i| = (k-i)t+1$ for $i \in \{1, \ldots, k\}$ using the following procedure. For $i = 1, \ldots, k$,

- select an arbitrary set R_i of size $(k-i)t+1$ in $S \setminus S_{i-1}$,
- set $S_i = S_{i-1} \cup \bigcup_{a \in R_i} \big((A(P_a) \cap S) \cup \{a\}\big)$.

We show by induction, that the sets $S_1, \ldots, S_k$ and $R_1, \ldots, R_k$ exist. Since $|S \setminus S_0| = |S| \geq (k-1)t+1$, we conclude that R_1 of size $(k-1)t+1$ can be selected. Assume that the sets S_j and R_j have been constructed for $0 \leq j < i \leq k$. Observe that because $|\bigcup_{a \in R_j} \big((A(P_a) \cap S) \cup \{a\}\big)| \leq (t+1)|R_j|$,

$$|S_j \setminus S_{j-1}| \leq |R_j|(t+1) = ((k-j)t+1)(t+1)$$

for $1 \leq j < i$. Therefore,

$$|S_{i-1}| \leq \sum_{j=1}^{i-1}(((k-j)t+1)(t+1)). \tag{1}$$

Notice that

$$\frac{1}{2}k(t+1)((k-1)t+2) = \sum_{j=1}^{k}(((k-j)t+1)(t+1)). \tag{2}$$

Then by (1) and (2),

$$|S \setminus S_{i-1}| \geq \sum_{j=i}^{k}(((k-j)t+1)(t+1)) \geq (k-i)t+1.$$

This means that R_i can be selected and we can construct S_i.

Now we select arcs $a_i \in R_i$ for $i = k, k-1, \ldots, 1$. Since $|R_k| = 1$, the choice of a_k is unique. Assume that $a_k, \ldots, a_{i+1}$ have been selected for $1 < i+1 \leq k$. Then we select an arbitrary

$$a_i \in R_i \setminus \bigcup_{j=i+1}^{k} A(P_{a_j}).$$

Because $|\bigcup_{j=i+1}^{k} A(P_{a_j})| \leq (k-i)t$ and $|R_i| = (k-i)t+1$, a_i exists.

Let $i \in \{1,\ldots,k\}$. By the choice of a_i, we have that $a_i \notin A(P_{a_j})$ for $i < j \leq k$. From the other side, $a_i \notin A(P_j)$ for $1 \leq j < i$, because $a_i \in R_i$ and R_i does not contain the arcs of P_a for $a \in R_j$ for $1 \leq j < i$ by the construction of the sets $R_1,\ldots,R_k$. We obtain that the t-detours P_{a_i} for $i \in \{1,\ldots,k\}$ do not contain any a_j for $j \in \{1,\ldots,k\}$. By Observation 1, $H = G - \{a_1,\ldots,a_k\}$ is a multiplicative t-spanner. Therefore, (G,t,k) is a yes-instance of Directed Multiplicative Spanner. ◁

By Claim 3, we can apply the next rule:

▶ **Reduction Rule 1.** *If $|S| \geq \frac{1}{2}k(t+1)((k-1)t+2)$, then return a trivial yes-instance of Directed Multiplicative Spanner and stop.*

From now, we assume that $|S| < \frac{1}{2}k(t+1)((k-1)t+2)$.

The analog of Reduction Rule 1 is a main step of the kernelization algorithm of Kobayashi [8] for the undirected case, because it almost immediately allows to upper bound the total number of edges of the graph. However, the directed case is more complicated, since the arcs of t-detours for $a \in S$ may be outside S contrary to the undirected case, where all the edges of t-detours are in cycles of length at most $t+1$ and, therefore, have t-detours themselves. We use the following procedure to mark the crucial arcs of potential detours.

Marking Procedure. Let $G' = G - S$.

- **(i)** For every $(u,v) \in S$, find a shortest (u,v)-path P in G' and if the length of P is at most t, then *mark* the arcs of P.
- **(ii)** For every ordered pair of two distinct arcs $(u_1,v_1),(u_2,v_2) \in S$,
  - **(a)** find a shortest (u_1,u_2)-path P_1 in G' and if the length of P_1 is at most t, then *mark* the arcs of P_1,
  - **(b)** find a shortest (v_2,v_1)-path P_2 in G' and if the length of P_2 is at most t, then *mark* the arcs of P_2,
  - **(c)** find a shortest (v_1,u_2)-path P_3 in G' and if the length of P_3 is at most t, then *mark* the arcs of P_3.

Observe that marking can be done in polynomial time by Dijkstra's algorithm. Denote by L the set of marked arcs. Our final rule constructs the output instance.

▶ **Reduction Rule 2.** *Consider the graph $H = (V(G), S \cup L)$. Delete the isolated vertices of H, and for the obtained G^*, output (G^*,t,k).*

We argue that the rule is safe.

▷ Claim 4. (G,t,k) is a yes-instance of Directed Multiplicative Spanner if and only if (G^*,t,k) is a yes-instance.

Proof of Claim 4. Suppose that (G,t,k) is a yes-instance of Directed Multiplicative Spanner. Then, by Observation 1, there are k distinct arcs $a_1,\ldots,a_k \in S$ with their t-detours $P_1,\ldots,P_k$, respectively, such that $a_i \notin \bigcup_{j=1}^{k} A(P_j)$. Notice that $a_1,\ldots,a_k \in A(G^*)$. Consider $i \in \{1,\ldots,k\}$ and let $a_i = (u,v)$.

Suppose that P_i does not contain arcs from S. Then P_i is a (u,v)-path in $G' = G - S$. By the first step of Marking Procedure, there is a t-detour P'_i for a_i whose arcs are in G' and are marked. Then P'_i is a t-detour for a_i in G^* and $a_j \notin A(P'_i)$ for $j \in \{1,\ldots,k\}$.

Assume that P_i contains some arcs from S. Let $e_1,\ldots,e_s$ be these arcs (in the path order with respect to P_i starting from u). Note that $e_1,\ldots,e_s \in A(G^*)$ and they are distinct from $a_1,\ldots,a_k$. Let $e_j = (x_j,y_j)$ for $j \in \{1,\ldots,s\}$. Then P_i can be written as the concatenation

of the paths $P_i = Q_1 \circ x_1y_1 \circ Q_2 \circ \cdots \circ x_sy_s \circ Q_{s+1}$, where Q_1 is the (u, x_1)-subpath of P_i, Q_j is the (y_{j-1}, x_j)-subpath of P_i for $j \in \{2, \ldots, s\}$, and Q_{s+1} is the (y_s, v)-subpath of P_i; note that some of the paths $Q_1, \ldots, Q_{s+1}$ may be trivial, i.e., contain a single vertex. Let $j \in \{1, \ldots, s+1\}$. If Q_j is trivial, then $Q'_j = Q_j$ is a path in G^*, because the vertices incident to the arcs of S are vertices of G^*. Suppose that Q_j is not trivial. If $j = 1$, then by step (ii)(a) of Marking Procedure, there is a (u, x_1)-path Q'_1, whose arcs are in G' and are marked, and the length of Q'_1 is at most the length of Q_1. For $j = s + 1$, we have, by step (ii)(b), that there is a (y_s, v)-path Q'_{s+1}, whose arcs are in G' and are marked, and the length of Q'_{s+1} is at most the length of Q_{s+1}. Suppose that $2 \leq j \leq s$. Then by step (ii)(c), there is a (y_{j-1}, x_j)-path Q'_j, whose arcs are in G' and are marked, and the length of Q'_j is at most the length of Q_j. Consider the (u, v)-walk $W_i = Q'_1 \circ x_1y_1 \circ Q'_2 \circ \cdots \circ x_sy_s \circ Q'_{s+1}$. We have that W'_i is a (u, v)-walk of length at most t in G^* such that $a_j \notin A(W_i)$ for $j \in \{1, \ldots, k\}$. This implies that G^* has a t-detour P'_i in G^* such that $a_j \notin A(P'_i)$ for $j \in \{1, \ldots, k\}$.

We obtain that for every $i \in \{1, \ldots, k\}$, $a_i \in A(G^*)$ has a t-detour P'_i such that $a_1, \ldots, a_k \notin A(P'_i)$. By Observation 1, we conclude that $G^* - \{a_1, \ldots, a_k\}$ is a multiplicative spanner for G^*, that is, (G^*, t, k) is a yes-instance of Directed Multiplicative Spanner.

For the opposite direction, assume that (G^*, t, k) is a yes-instance of Directed Multiplicative Spanner. By Observation 1, there are k distinct arcs $a_1, \ldots, a_k \in A(G^*)$ with their t-detours $P_1, \ldots, P_k$, respectively, such that $a_i \notin \bigcup_{j=1}^{k} A(P_j)$. Since G^* is a subgraph of G, $a_1, \ldots, a_k$ have the same t-detours in G. By Observation 1, (G, t, k) is a yes-instance. ◁

To upper bound the size of G^*, observe that Marking Procedure marks at most t arcs for each $a \in S$ in step (i), that is, at most $|S|t$ arcs are marked in this step. In step (ii), we mark at most $3t$ arcs for each ordered pair of arcs of S. Hence, at most $3|S|(|S| - 1)t$ arcs are marked in total in the second step. Since $|S| < \frac{1}{2}k(t + 1)((k - 1)t + 2)$, we have that G^* has $\mathcal{O}(k^4t^5)$ arcs. Because G^* has no isolated vertices, the number of vertices is $\mathcal{O}(k^4t^5)$.

Since each of the reduction rules and Marking Procedure can be done in polynomial time, we conclude that the total running time of our kernelization algorithm is polynomial. ◀

3.2 FPT algorithm for Directed Multiplicative Spanner

Combining Theorem 2 with the brute-force procedure that guesses k arcs of G and verifies whether the deletion of these arcs gives a multiplicative t-spanner, we obtain the straightforward $2^{\mathcal{O}(k \log(kt))} + n^{\mathcal{O}(1)}$ algorithm for Directed Multiplicative Spanner. If we use the intermediate steps of the kernelization algorithm, then the running time may be improved to $(kt)^{2k} \cdot n^{\mathcal{O}(1)}$. Namely, we can construct the set S of t-good arcs and execute Reduction Rule 1 of the kernelization algorithm. Then we either solve the problem or obtain an instance, where the set S has size at most $\frac{1}{2}k(t + 1)((k - 1)t + 2) - 1 \leq k^2t^2$. Then for every $R \subseteq S$ of size k, we check whether $G - R$ is a multiplicative t-spanner by computing the distances between every pair of vertices. However, we can slightly improve the parameter dependence by making use of the *random separation* technique proposed by Cai, Chan, and Chan in [3] (we refer to [5, Chapter 5] for the detailed introduction to the technique). In this subsection, we briefly sketch a Monte Carlo algorithm with false negatives for Directed Multiplicative Spanner.

▶ **Theorem 5.** *Directed Multiplicative Spanner can be solved in time* $(4t)^k \cdot n^{\mathcal{O}(1)}$ *by a Monte Carlo algorithm with false negatives.*

Proof. Let (G,t,k) be an instance of Directed Multiplicative Spanner. If $k=0$ or $t=1$, then the problem is trivial: if $k=0$, then (G,t,k) is a yes-instance, and if $k>0$ and $t=1$, then (G,t,k) is a no-instance. From now we assume that $k\geq 1$ and $t\geq 2$.

By Observation 1, to solve Directed Multiplicative Spanner for (G,t,k), it is necessary and sufficient to identify k arcs that have t-detours that do not contain selected arcs. We use random separation to distinguish the arcs that have t-detours and the arcs of the detours. We randomly color the arcs of G by two colors *red* and *blue*. An arc is colored red with probability $\frac{1}{t}$ and is colored blue with probability $\frac{t-1}{t}$. Then we try to find k red arcs that have t-detours composed by blue arcs. Let R be the set of arcs colored red and let B the set of blue arcs. For $(u,v)\in R$, it can be checked in polynomial time whether (u,v) has a t-detour with blue arcs by finding the distance between u and v in $G_B=(V(G),B)$. Then we greedily construct the set S of all red arcs with blue t-detours. If $|S|\geq k$, then we conclude that (G,t,k) is a yes-instance by Observation 1.

Suppose that (G,t,k) is a yes-instance of Directed Multiplicative Spanner. Then by Observation 1, there are k distinct arcs $a_1,\ldots,a_k$ and their t-detours $P_1,\ldots,P_k$, respectively, such that $a_1,\ldots,a_k\notin L=\bigcup_{i=1}^{k}A(P_i)$. Notice that $|L|\leq tk$. Then the probability that the considered random coloring colors the arcs $a_1,\ldots,a_k$ red is t^{-k} and the probability that the arcs of L are colored blue is at least $(\frac{t-1}{t})^{tk}$. We have that

$$\left(\frac{t-1}{t}\right)^t=\left(1-\frac{1}{t}\right)^t\geq\frac{1}{4}.$$

Therefore, the probability that the arcs $a_1,\ldots,a_k$ are red and their t-detours are blue is at least $(4t)^{-k}$. Respectively, the probability that the random coloring fails to color the arcs $a_1,\ldots,a_k$ red and their t-detours blue is at most $1-\frac{1}{(4t)^k}$. This implies that if we iterate our algorithm for $(4t)^k$ colorings, then we either find a solution and stop or we conclude that (G,t,k) is a no-instance with the mistake probability at most $\left(1-\frac{1}{(4t)^k}\right)^{(4t)^k}\leq e^{-1}$. This gives us a Monte Carlo algorithm with running time $(4t)^k\cdot n^{\mathcal{O}(1)}$. ◀

The same approach can be used for undirected graphs and it can be shown that Multiplicative Spanner can be solved in $(4t)^k\cdot n^{\mathcal{O}(1)}$ time improving the running time given in [8].

The algorithm from Theorem 5 can be derandomized by using *universal sets* [13] instead of random colorings. Since this part is standard (see [5, Chapter 5]), we leave it to the interested readers.

4 Directed additive t-spanners

In this section, we consider Directed Additive Spanner and show that the problem is hard on DAGs even if $t=1$.

▶ **Theorem 6.** *Directed Additive Spanner is* W[1]*-hard on DAGs when parameterized by k only even if $t=1$.*

Proof. We reduce from the Independent Set problem. Given a graph G and a positive integer k, the problem asks whether G has an independent set of size at least k. Independent Set parameterized k is well-known to be one of the basic W[1]-complete problems (see [5, 6]).

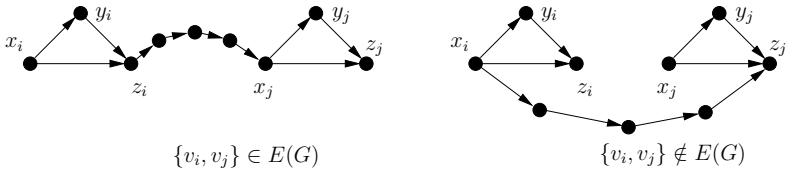


Figure 1 Construction of D.

Let (G, k) be an instance of Independent Set. Denote by $v_1, \ldots, v_n$ the vertices of G.

- For every $i \in \{1, \ldots, n\}$, construct three vertices x_i, y_i, z_i and arcs $(x_i, y_i), (y_i, z_i), (x_i, z_i)$.
- For every $i, j \in \{1, \ldots, n\}$ such that $i < j$, do the following:
  - if $\{v_i, v_j\} \in E(G)$, then construct a directed (z_i, x_j)-path P_{ij} of length 4,
  - if $\{v_i, v_j\} \notin E(G)$, then construct a directed (x_i, z_j)-path Q_{ij} of length 4.

Denote the obtained directed graph by D (see Figure 1). It is straightforward to verify that D is a DAG. We show that (G, k) is a yes-instance of Independent Set if and only if $(D, 1, k)$ is a yes-instance of Directed Additive Spanner.

Suppose that $I = \{v_{i_1}, \ldots, v_{i_k}\}$ is an independent set of size k in G. Let $R = \{(x_{i_1}, z_{i_1}), \ldots, (x_{i_k}, z_{i_k})\}$. We show that $D' = D - R$ is an additive 1-spanner for D.

We first claim that for every two vertices u and w of D, each shortest (u, w)-path in D contains at most one arc of R. The proof is by contradiction. Assume that there are $u, w \in V(D)$ and a shortest (u, w)-path P such that P contains at least two arcs of R. Let (x_i, z_i) and (x_j, z_j) be such arcs and let $i < j$. By the construction, (x_i, z_i) occurs before (x_j, z_j) in P. Since the arcs of R correspond to vertices of the independent set I, v_i and v_j are not adjacent in G. Therefore, D contains the (x_i, z_j)-path Q_{ij} of length 4. Since P is a shortest path containing (x_i, z_i) and (x_j, z_j), the (z_i, x_j)-subpath of P should have length at most 2. However, by the construction, the distance between z_i and x_j is at least 4; a contradiction proving the claim.

Now let u and w be two vertices of D. Let P be a shortest (u, w)-path in D. If P is a path in D', then $\mathrm{dist}_{D'}(u, w) = \mathrm{dist}_D(u, w)$. Suppose that P is not a path in D'. Then P contains a unique arc $(x_i, z_i) \in R$ by the proved claim. Let P_1 be the (u, x_i)-subpath of P and let P_2 be the (z_i, w)-subpath. Let $P' = P_1 \circ x_i y_i z_i \circ P_2$. Observe that P' is a path in D'. Since the length of P' is the length of P plus 1, $\mathrm{dist}_{D'}(u, w) \leq \mathrm{dist}_D(u, w) + 1$. This implies that D' is an additive 1-spanner of D.

Now we assume that $(D, 1, k)$ is a yes-instance of Directed Additive Spanner. Then there is a set of k arcs $R \subseteq A(D)$ such that $D' = D - R$ is an additive 1-spanner. Observe that if $(u, v) \in R$, then D has an (u, v)-path P that does not use the arc (u, v). Otherwise, $\mathrm{dist}_{D'}(u, v) = +\infty$ and $\mathrm{dist}_{D'}(u, v) > \mathrm{dist}_D(u, v) + 1$. Therefore, $R \subseteq \{(x_1, z_1), \ldots, (x_n, z_n)\}$. Let $R = \{(x_{i_1}, z_{i_1}), \ldots, (x_{i_k}, z_{i_k})\}$. We claim that $I = \{v_{i_1}, \ldots, v_{i_k}\}$ is an independent set of G. Assume, for the sake of contradiction, that this is not the case and there are $v_i, v_j \in I$ such that v_i and v_j are adjacent in G. Let $i < j$. Consider the vertices x_i and z_j of D. Since $\{v_i, v_j\} \in E(G)$, $P = x_i z_i \circ P_{ij} \circ x_j z_j$ is an (x_i, z_j)-path of length 6, that is, $\mathrm{dist}_D(x_i, z_j) \leq 6$. The path $P' = x_i y_i z_i \circ P_{ij} \circ x_j y_j z_j$ has length 8 and is a path in D'. Any other (x_i, z_j)-path in D' uses at least two paths of length 4: one of the paths $P_{ii'}$ and $Q_{ii'}$ for some $i' \in \{1, \ldots, n\}$ such that $i' \neq j$, and one of the paths $P_{j'j}$ and $Q_{j'j}$ for some $j' \in \{1, \ldots, n\}$ such that $j' \neq i$. This means that $\mathrm{dist}_{D'}(x_i, z_j) - \mathrm{dist}_D(x_i, z_j) \geq 2$ contradicting that D' is an additive 1-spanner. We conclude that I is an independent set of G and, therefore, (G, k) is a yes-instance of Independent Set. ◀

5 Conclusion

We proved that Directed Multiplicative Spanner admits a kernel of size $\mathcal{O}(k^4t^5)$ can be solved in $(4t)^k \cdot n^{\mathcal{O}(1)}$ randomized time. We also demonstrated that Directed Additive Spanner is W[1]-hard even when $t = 1$ and the input graphs are restricted to DAGs. The latter result leads to the question whether Directed Additive Spanner is tractable on some special classes of directed graphs, like planar directed graphs. We believe that this problem may be interesting even if the distortion parameter t is assumed to be a constant.

Another possible direction of research is considering different types of directed graph spanners. For example, what can be said about the roundtrips spanners introduced by Roditty, Thorup, and Zwick [16]? A spanning subgraph H of a directed graph G is a multiplicative *t-roundtrip-spanner* if for every two vertices u and v, $\mathrm{dist}_H(u,v)+\mathrm{dist}_H(v,u) \leq t(\mathrm{dist}_G(u,v)+\mathrm{dist}_G(v,u))$, that is, H approximates the sum of the distances between any two vertices in both directions. Is the analog of Directed Multiplicative Spanner for roundtrip spanners FPT? Notice that we cannot use Observation 1 that is crucial for our results for the new problem. Consider, for example, the directed graph G constructed as follows: construct two vertices u and v and an arc (u,v), and then add a (u,v)-path P_1 and a (v,u)-path P_2 of arbitrary length $\ell \geq 2$ that are internally vertex disjoint. Then it is easy to see that $H = G - (u,v)$ is a 2-roundtrip spanner for G. However, H has no short detour for (u,v). It is also possible to define additive t-roundtrip-spanners and consider the analog of Directed Additive Spanner. We conjecture that this problem is at least as hard as Directed Additive Spanner.

Let us also mention that we are not aware of results about the parameterized complexity of the weighted variants of Multiplicative Spanner and Additive Spanner on both directed and undirected graphs. Here, the input graph is supplied with the edge (arc) weights and the length of a path is the sum of the weights of its edges (arcs, respectively). Then the distance between vertices is the length of a shortest path in this metric.

References

1. Abu Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen G. Kobourov, and Richard Spence. Graph spanners: A tutorial review. *CoRR*, abs/1909.03152, 2019. arXiv:1909.03152.
2. Leizhen Cai. NP-completeness of minimum spanner problems. *Discret. Appl. Math.*, 48(2):187–194, 1994. doi:10.1016/0166-218X(94)90073-6.
3. Leizhen Cai, Siu Man Chan, and Siu On Chan. Random separation: A new method for solving fixed-cardinality optimization problems. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, volume 4169 of *Lecture Notes in Computer Science*, pages 239–250. Springer, 2006. doi:10.1007/11847250_22.
4. Eden Chlamtác, Michael Dinitz, Guy Kortsarz, and Bundit Laekhanukit. Approximating spanners and directed steiner forest: Upper and lower bounds. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 534–553. SIAM, 2017. doi:10.1137/1.9781611974782.34.
5. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
6. Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

7 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization*. Cambridge University Press, Cambridge, 2019. Theory of parameterized preprocessing.

8 Yusuke Kobayashi. NP-hardness and fixed-parameter tractability of the minimum spanner problem. *Theor. Comput. Sci.*, 746:88–97, 2018. doi:10.1016/j.tcs.2018.06.031.

9 Yusuke Kobayashi. An FPT algorithm for minimum additive spanner problem. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 11:1–11:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.11.

10 Guy Kortsarz. On the hardness of approximating spanners. *Algorithmica*, 30(3):432–450, 2001. doi:10.1007/s00453-001-0021-y.

11 Arthur L. Liestman and Thomas C. Shermer. Additive spanners for hypercubes. *Parallel Process. Lett.*, 1:35–42, 1991. doi:10.1142/S0129626491000197.

12 Arthur L. Liestman and Thomas C. Shermer. Additive graph spanners. *Networks*, 23(4):343–363, 1993. doi:10.1002/net.3230230417.

13 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 182–191. IEEE Computer Society, 1995. doi:10.1109/SFCS.1995.492475.

14 David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989. doi:10.1002/jgt.3190130114.

15 David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, 1989. doi:10.1137/0218050.

16 Liam Roditty, Mikkel Thorup, and Uri Zwick. Roundtrip spanners and roundtrip routing in directed graphs. *ACM Trans. Algorithms*, 4(3):29:1–29:17, 2008. doi:10.1145/1367064.1367069.

A Polynomial Kernel for Funnel Arc Deletion Set

Marcelo Garlet Milani
Technische Universität Berlin, Chair of Logic and Semantics, Germany
m.garletmillani@tu-berlin.de

Abstract

In Directed Feedback Arc Set (DFAS) we search for a set of at most k arcs which intersect every cycle in the input digraph. It is a well-known open problem in parameterized complexity to decide if DFAS admits a kernel of polynomial size. We consider $\mathcal{C}$-Arc Deletion Set ($\mathcal{C}$-ADS), a variant of DFAS where we want to remove at most k arcs from the input digraph in order to turn it into a digraph of a class $\mathcal{C}$. In this work, we choose $\mathcal{C}$ to be the class of *funnels*. Funnel-ADS is NP-hard even if the input is a DAG, but is fixed-parameter tractable with respect to k. So far no polynomial kernel for this problem was known. Our main result is a kernel for Funnel-ADS with $\mathcal{O}(k^6)$ many vertices and $\mathcal{O}(k^7)$ many arcs, computable in $\mathcal{O}(nm)$ time, where n is the number of vertices and m the number of arcs of the input digraph.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases graph editing, directed feedback arc set, parameterized algorithm, kernels, funnels

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.13

Related Version A full version of the paper is available at `https://arxiv.org/abs/1911.05520`.

Acknowledgements We thank the referees for their numerous helpful comments.

1 Introduction

In graph editing problems, we are given a (directed or undirected) graph G and a number k, and we search for a set of at most k vertices, edges or arcs whose removal or addition produces a graph with a desired property. There are several variants of these problems, and in this paper we consider the problem of removing arcs from a digraph in order to obtain a digraph in a given class $\mathcal{C}$. When $\mathcal{C}$ is the class of all directed acyclic graphs (DAGs), the problem is called Directed Feedback Arc Set (DFAS). If we remove vertices instead of arcs, the problem is called Directed Feedback Vertex Set (DFVS).

There are simple reductions between DFAS and DFVS. We can reduce DFAS to DFVS by taking the line digraph of the input. Removing a vertex from the reduced instance corresponds to removing an arc from the input instance and vice versa. For a reduction in the other direction, we split each vertex v into two vertices, say, v_o and v_i, connect them with an arc (v_i, v_o) and shift all outgoing arcs of v to v_o and all incoming arcs to v_i. In the context of *parameterized complexity*, such reductions are called *parameterized* as the parameter k is preserved. Hence, parameterized results are often stated for DFVS.

In a breakthrough paper it was proven that there is an algorithm for DFVS with running time $4^k k! \cdot n^{\mathcal{O}(1)}$ [4], showing that the problem is *fixed-parameter tractable* (FPT) with respect to k. After obtaining an FPT result, it is natural to ask if the problem also admits a polynomial *kernel*, that is, if there is a polynomial-time algorithm which reduces the input instance to an instance of size at most $\mathcal{O}(k^c)$ for some constant c. Such an algorithm is called a *kernelization* algorithm.

The existence of a polynomial kernel for DFVS is a fundamental open question in the field of parameterized complexity. One approach towards solving this question is to consider different parametrizations or restrictions of the input digraph. By considering progressively smaller parameters or more general digraph classes, one can hope to eventually close the gap between the restricted cases and the general case of DFVS.

On tournaments, DFVS admits a polynomial kernel [1]; this was extended to generalizations of tournaments as well [3]. When parameterized by solution size k and the size ℓ of a treewidth η-modulator, DFVS admits a kernel of size $(k \cdot \ell)^{\mathcal{O}(\eta^2)}$ [10].

One can also restrict the output instead, that is, we can consider $\mathcal{C}$-Vertex Deletion Set ($\mathcal{C}$-VDS) or $\mathcal{C}$-Arc Deletion Set ($\mathcal{C}$-ADS), where, for a fixed digraph class $\mathcal{C}$, we search for a set of at most k vertices (arcs) whose removal turns the input into a digraph in $\mathcal{C}$. Unlike DFVS and DFAS, $\mathcal{C}$-VDS and $\mathcal{C}$-ADS can belong to different complexity classes depending on $\mathcal{C}$: While Out-Forest-ADS can be solved in polynomial time, Out-Forest-VDS is NP-hard [12]. Further, note that even if $\mathcal{C}' \subseteq \mathcal{C}$, a polynomial kernel for $\mathcal{C}$-ADS does not immediately imply a polynomial kernel for $\mathcal{C}'$-ADS, and the implication also does not work in the other direction. Indeed, while the problem is trivial when $\mathcal{C}$ is the class of all independent sets or the class of all digraphs, it is NP-hard if $\mathcal{C}$ is the class of DAGs, which contains all independent sets and is a subclass of all digraphs. In a sense, the complexity landscapes of $\mathcal{C}$-ADS and $\mathcal{C}$-VDS are much more fine-grained than the landscape of DFVS, and may allow for smaller steps towards more general results.

Out-Forest-ADS and Pumpkin-ADS can be solved in polynomial time [12], while Out-Forest-VDS and Pumpkin-VDS are NP-hard and admit polynomial kernels [2, 12] of size $\mathcal{O}(k^2)$ and $\mathcal{O}(k^3)$, respectively [2]. $\mathcal{F}_\eta$-VDS admits a polynomial kernel for constant η, where $\mathcal{F}_\eta$ is the class of all digraphs with (undirected) treewidth at most η [10].

In this work we consider Funnel-ADS and provide a polynomial kernel with $\mathcal{O}(k^6)$ vertices and $\mathcal{O}(k^7)$ arcs. A digraph is a funnel if it is a DAG and every source to sink path has an arc which is not in any other source to sink path. Funnel-ADS is NP-hard even if the input is DAG, but it can be solved in $\mathcal{O}(3^k \cdot (n+m))$ time [11], where k is the solution size. Out-forests and pumpkins are also funnels, but there are also dense funnels like complete bipartite digraphs (where all arcs go from the first partition to the second but not back).

Our results rely on characterizations for funnels based on forbidden subgraphs and on a "labeling" of the vertices [11]. We believe the techniques used here can be generalized to other digraph classes which are also similarly characterized, and hope they provide further insight about the classes $\mathcal{C}$ for which $\mathcal{C}$-ADS admits a polynomial kernel.

2 Preliminaries

A (partial) function $f : A \to B$ is a set of tuples $(a, f(a)) \in A \times B$ where for every $a \in A$ there is at most one $b \in B$ with $(a, b) \in f$ (that is, $f(a) = b$). We write $\mathrm{Dom}(f)$ for the set of values $a \in A$ for which f is defined. Hence, $\emptyset$ is the undefined function, and $f' \supseteq f$ if $f'(x) = f(x)$ for every $x \in \mathrm{Dom}(f)$. All our functions are *partial*, that is, $\mathrm{Dom}(f)$ is not necessarily A.

A *parameterized language* L is *fixed-parameter tractable* with respect to the parameter k if there is some algorithm with running time $f(k) \cdot n^{\mathcal{O}(1)}$ deciding whether $(x, k) \in L$, where f is some computable function, $n = |x|$ and k is the parameter (refer to [5, 6] for an introduction to parameterized complexity). We say that L *admits a problem kernel* if there is a polynomial-time algorithm which transforms an instance (x, k) into an instance (x', k') such that $(x, k) \in L$ if and only if $(x', k') \in L$, $k' \leq k$ and $|x'| \leq f(k)$ for some computable function f. If f is a polynomial, we say that L *admits a polynomial kernel* with respect to k.

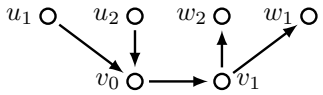


Figure 1 D_1, a forbidden subgraph for funnels.

When describing a kernelization algorithm, it is common to define *reduction rules*. These rules have a *condition* and an *effect*, and we say that a reduction rule is *applicable* if the condition is true. The effect of the reduction rule produces a new instance (x', k') of the problem, and a rule is said to be *safe* if $(x', k') \in L$ if and only if the original instance is in L. We refer the reader to [8, 9] for surveys on kernelization and to [7] for a book on the topic.

We only consider directed graphs (digraphs) without loops or parallel arcs (but we allow arcs in opposite directions) in this paper. Let D be a digraph. The set of arcs of D is denoted by $A(D)$, and its set of vertices is $V(D)$. The set of outneighbors (inneighbors) in D of a vertex $v \in V(D)$ is denoted by $\mathsf{out}_D(v)$ ($\mathsf{in}_D(v)$); the outdegree (indegree) of v is $\mathsf{outdeg}_D(v) = |\mathsf{out}_D(v)|$ ($\mathsf{indeg}_D(v) = |\mathsf{in}_D(v)|$). If the digraph D is clear from context, we omit it from the index. For a set $U \subseteq V(D)$ we write $\mathsf{out}(U)$ for the set $\{\mathsf{out}(u) \mid u \in U\} \setminus U$ (and analogously for $\mathsf{in}(U)$). A vertex v is a *source* if $\mathsf{indeg}(v) = 0$, and it is a sink if $\mathsf{outdeg}(v) = 0$. We write $H \subseteq D$ if H a *subgraph* of D; the subgraph of D *induced* by U is given by $D[U]$. We write $D - X$ for the operation of deleting a set of vertices or arcs X from D. Similarly, we add a set of arcs or vertices to D with $D + X$.

A *directed acyclic graph* (DAG) is a digraph which does not contain any directed cycle. A digraph D is a funnel if D is a DAG and for every path P from a source to a sink of D of length at least one there is some arc $a \in A(P)$ such that for any different path Q from a (possibly different) source to a sink we have $a \notin A(Q)$. We repeat below several known characterizations for funnels, as they are particularly useful for our results.

▶ **Theorem 1** ([11], Theorem 1). *Let D be a DAG. The following statements are equivalent.*

a. *D is a funnel.*

b. *$V(D)$ can be partitioned into two sets F and M such that:* (1) *F induces an out-forest;* (2) *M induces an in-forest; and* (3) *$(M \times F) \cap A(D) = \emptyset$.*

c. *No digraph in $\mathcal{F} = \{D_i \mid i \in \{0, 1, \dots\}\}$ is contained in D as a (not necessarily induced) subgraph, where (see Figure 1 for an example)*
- *$V(D_k) = \{u_1, u_2, w_1, w_2\} \cup \{v_i \mid 0 \leq i \leq k\}$, and*
- *$A(D_k) = \{(u_1, v_0), (u_2, v_0), (v_k, w_1), (v_k, w_2)\} \cup \{(v_i, v_{i+1}) \mid 1 \leq i \leq k-1\}$*

d. *D does not contain D_0 as a butterfly minor.*

The digraphs in $\mathcal{F}$ are called *forbidden subgraphs for funnels*. For a digraph D we define a *labeling* as a function $\ell : V(D) \to \{\mathrm{F}, \mathrm{M}\}$. We say that ℓ is a *funnel labeling* for D if $\mathrm{Dom}(\ell) = V(D)$, the set $F = \{v \in V(D) \mid \ell(v) = \mathrm{F}\}$ induces an out-forest in D, the set $M = \{v \in V(D) \mid \ell(v) = \mathrm{M}\}$ induces an in-forest in D and $(M \times F) \cap A(D) = \emptyset$. Due to Theorem 1(b), a digraph D is a funnel if and only if there exists a funnel labeling for D.

In the *feedback arc set* problem, we are given a digraph D and a $k \in \mathbb{N}$ as an input, and we search for a set $S \subseteq A(D)$ such that $D - S$ is a DAG and $|S| \leq k$. We consider a variant of this problem where we want $D - S$ to be a funnel instead, which is formally defined below.

Funnel Arc Deletion Set (FADS)

Input A digraph D and a number $k \in \mathbb{N}$.

Question Is there a set $S \subseteq A(D)$ with $|S| \leq k$ such that $D - S$ is a funnel?

To make better use of Theorem 1(b), we consider a more general problem in which some vertices might already be labeled with F or M, and the funnel we obtain in the end must respect this labeling. Formally, the problem is defined as follows.

Funnel Arc Deletion Labeling (FADL)

Input A digraph D, a labeling $\ell : V(D) \to \{\mathrm{F}, \mathrm{M}\}$ and a number $k \in \mathbb{N}$.

Question Are there a set $S \subseteq A(D)$ and a labeling $\hat{\ell} \supseteq \ell$ such that $\hat{\ell}$ is a funnel labeling for $D - S$ and $|S| \leq k$?

We say that (D, ℓ, k) is the *input instance* and $(S, \hat{\ell})$ is a solution for the input instance. This more general version of the problem allows us to decide which label a vertex will take and encode this in the instance itself. While technically not necessary, using FADL instead of FADS simplifies the kernelization algorithm and also the proofs. Due to space constraints, proofs marked with ($\star$) are deferred to the full version of the paper.

3 Basic reduction rules

We construct our kernelization algorithm by defining a series of reduction rules and then showing that, if no reduction rule is applicable, the input size is bounded in a polynomial of k. Our strategy is to partition the vertex set into labeled and unlabeled vertices, then bound the number of unlabeled vertices (Section 3.1) and use this to bound the number of labeled vertices (Section 3.2) as well. In this section we define some reduction rules which are useful both in Section 3.1 as well as in Section 3.2. For brevity, we assume that a reduction rule is no longer applicable to the input instance after it has been defined.

Let (D, ℓ, k) be the input instance. From Theorem 1(c) we can see that a funnel has no vertex v with $\mathsf{indeg}(v) > 1$ and $\mathsf{outdeg}(v) > 1$. Further, $\mathsf{indeg}(v) \leq 1$ if $\ell(v) = \mathrm{F}$, and $\mathsf{outdeg}(v) \leq 1$ if $\ell(v) = \mathrm{M}$. Hence, by simply counting the number of vertices disrespecting each case, we can obtain a lower bound for the number of arcs that need to be removed from D in order to obtain a funnel. As removing one arc changes the degree of two vertices, we obtain a bound of at most $2k$ such vertices. The safety of the following reduction rule follows easily from Theorem 1.

▶ **Reduction Rule 1** (Lower Bound). *Let $V_I \subseteq V(D)$ be the set of vertices with indegree greater than one, let V_O be the set of vertices with outdegree greater than one and let $V_X = V_O \cap V_I$. Output a trivial "no" instance if*

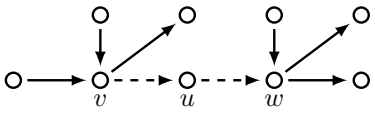
$$\sum_{u \in V_O, \ell(u) = M} (\mathsf{outdeg}(u) - 1) + \sum_{u \in V_I, \ell(u) = F} (\mathsf{indeg}(u) - 1) + \sum_{u \in V_X, u \notin \mathrm{Dom}(\ell)} (\min\{\mathsf{indeg}(u), \mathsf{outdeg}(u)\} - 1) > 2k.$$

The following reduction rule is based on [11], with some modifications since the original reduction rule is applied as an intermediate step in an FPT algorithm and is not safe for kernelization. For certain vertices it is possible to optimally decide which label they should receive in an optimal solution. For example, vertices with outdegree greater than $k + 1$ can always be labeled with F, as otherwise we would need to remove at least $k + 1$ of its outgoing arcs, which is not possible.

▶ **Reduction Rule 2** (Set Label). *Let $v \in V(D)$ be an unlabeled vertex.*

Set $\ell(v) := F$ if at least one of the following is true: (1) $\mathsf{indeg}(v) = 0$*;* (2) *v has a single inneighbor u and $\ell(u) = F$;* (3) *there are at least $\mathsf{indeg}(v) + 1$ vertices $u \in \mathsf{out}(v)$ with $\ell(u) = M$ or $\ell(u) = F \wedge \mathsf{indeg}(u) = 1$; or* (4) $\mathsf{outdeg}(v) > k + 1$.

Figure 2 A digraph which is not a funnel. Removing the arcs (v, u) and (u, w) results in a funnel.



Set $\ell(v) \coloneqq M$ if at least one of the following is true: (1) $\mathsf{outdeg}(v) = 0$*;* (2) v *has a single outneighbor* u *and* $\ell(u) = M$*;* (3) *there are at least* $\mathsf{outdeg}(v) + 1$ *vertices* $u \in \mathsf{in}(v)$ *with* $\ell(u) = F$ *or* $\ell(u) = M \wedge \mathsf{outdeg}(u) = 1$*; or* (4) $\mathsf{indeg}(v) > k + 1$.

Proof of safety of Set Label (RR 2). Clearly, a solution for the reduced instance is also a solution for the original instance. For the other direction, we consider only the case where we set $\ell(v) \coloneqq \mathrm{F}$, as the other case is symmetric. Let ℓ_r be the labeling obtained by the reduction rule. Let $(S, \hat{\ell})$ be a solution for the original instance. We set $\hat{\ell}_r \coloneqq \hat{\ell}$ and $\hat{\ell}_r(v) \coloneqq \mathrm{F}$. If $\hat{\ell}(v) = \mathrm{F}$, then clearly $(S, \hat{\ell}_r)$ is a solution for the reduced instance. So assume $\hat{\ell}(v) = \mathrm{M}$. This implies that $\mathsf{outdeg}(v) \leq k + 1$, as otherwise $|S| > k$.

If $\mathsf{indeg}(v) = 0$, or $\mathsf{indeg}(v) = 1$ and there is some $u \in \mathsf{in}(v)$ with $\ell(u) = \mathrm{F}$, then $\hat{\ell}_r$ is clearly also a funnel labeling for $D - S$.

Let $U = \{u \in \mathsf{out}(v) \mid \ell(u) = \mathrm{M} \text{ or } \ell(u) = \mathrm{F} \wedge \mathsf{indeg}(u) = 1\}$. If $|U| \geq \mathsf{indeg}(v) + 1$, we construct an S_r from S as follows. We add all incoming arcs of v to S_r and remove from S_r all outgoing arcs (v, u) where $u \in U$. Since $\hat{\ell}(v) = \mathrm{M}$, at least $\mathsf{outdeg}(v) - 1 \geq \mathsf{indeg}(v)$ many outgoing arcs of v are in S. Hence, we remove at least $\mathsf{indeg}(v)$ arcs from S and add at most $\mathsf{indeg}(v)$. Thus, $|S_r| \leq |S|$.

The digraph $D - S_r$ does not contain cycles, as all incoming arcs of v were removed, so any cycle in $D - S_r$ is also in $D - S$, which is a funnel. To see that $\hat{\ell}_r$ is a funnel labeling of $D - S_r$, first note that we can always keep arcs (v, u) in $D - S_r$ where $\ell(u) = \mathrm{M}$. We can also keep arcs (v, u) in $D - S_r$ where $\ell(u) = \mathrm{F}$ and $\mathsf{indeg}(u) = 1$. As v has no incoming arcs in $D - S_r$, it lies in an out-forest. Hence, $\hat{\ell}_r$ is a funnel labeling of $D - S_r$. ◀

Replacing an arc in a funnel by a directed path cannot create any cycles nor any forbidden subgraph for funnels. The next reduction rule reverses this operation: We can contract certain paths where all vertices have in- and outdegree one to a single arc. However, we cannot replace any such path: In the example in Figure 2, if we remove u and add the arc (v, w), then the size of an optimal solution set decreases by one. Some cases where contracting an arc is safe are identified below.

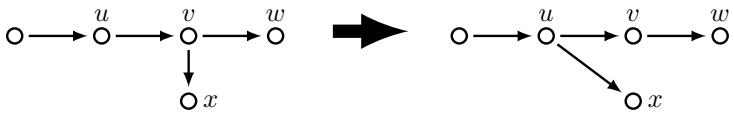
▶ **Reduction Rule 3** (Dissolve Vertex) **(⋆).** *Let u, v, w be a path such that the following is true:* (1) $v, u \in \mathrm{Dom}(\ell)$ *implies* $\ell(v) = \ell(u)$*; and* (2) $v, w \in \mathrm{Dom}(\ell)$ *implies* $\ell(v) = \ell(w)$.

If $\mathsf{indeg}(v) = \mathsf{outdeg}(v) = 1$ *and* $(\mathsf{indeg}(w) = 1 \vee \mathsf{outdeg}(u) = 1)$*, delete the vertex* v *and add the arc* (u, w).

3.1 Bounding the number of unlabeled vertices

From Lower Bound (RR 1) we know there are few vertices with both in- and outdegree greater than one. In this section we bound the number of unlabeled vertices by considering the remaining unlabeled vertices, that is, vertices v with $\mathsf{indeg}(v) \leq 1$ or $\mathsf{outdeg}(v) \leq 1$. Our strategy is to group such vertices into subgraphs of D with specific properties which we define later, and then develop reduction rules to both bound the maximum number of such subgraphs and also their size in any "yes" instance of FADL.

Figure 3 Example application of Shift Neighbors (RR 4).



Even if the previous reduction rules are not applicable, there can still exist some "large" subgraph $H \subseteq D$ for which there is a "small" set $S \subseteq A(D)$ such that the weakly-connected component of H is a funnel in $D - S$. Our goal is to bound the size of such subgraphs H.

We first define a specific type of subgraph of D which behaves like a funnel in the sense that the degrees of the vertices match Theorem 1(b). We call such subgraphs *local funnels* and formally define them below.

▶ **Definition 2.** *An induced subgraph $H \subseteq D$ is a* local funnel *in D if H is a funnel, H has only one source and its vertex set can be partitioned into $F \uplus M = V(H)$ such that $\mathsf{indeg}_D(v) \leq 1$ for all $v \in F$; $\mathsf{outdeg}_D(v) \leq 1$ for all $v \in M$; and $(M \times F) \cap A(H) = \emptyset$.*

Unlike *local* funnels, we might still have to remove many arcs from an *induced* funnel in D, as it can have, for example, several vertices v with $\mathsf{indeg}_D(v) > 1$ and $\mathsf{outdeg}_D(v) > 1$. Our goal is to bound the size of each unlabeled local funnel (that is, each local funnel where none of the vertices have a label) and the number of unlabeled local funnels in D. We start by "pushing" as many vertices as we can to the neighborhood of the roots of the in- and out-forests of a local funnel. Consider for example a path u, v, w as in Figure 3, whose vertices have indegree one but can have higher outdegree. Intuitively, a cycle containing v and x must also contain u. To destroy this cycle, we can remove the unique incoming arc of u, as this will potentially destroy further cycles that contain u but not v. Hence, replacing the arc (v, x) with (u, x) in this case does not change the size of the solution.

By moving vertices in an out-tree towards its root s, we increase the outdegree of s. If the outdegree of s increases beyond $k + 1$, we can apply Set Label (RR 2) to s, giving it a label. By further applying Set Label (RR 2) to the neighbors of s which are in its out-tree, we can label the entire tree. As we are only considering unlabeled local funnels in this section, we can use the idea above to limit the branching of any in- or out-tree of an unlabeled local funnel.

We provide here a somewhat more general reduction rule which can also be applied if some vertices are labeled. Later, this reduction rule will again be useful to bound the number of labeled vertices. However, we need to carefully consider the possible labels of the vertices, as in some cases the rule would not be safe.

▶ **Reduction Rule 4** (Shift Neighbors)**.** *Let u, v, w be a path.*

- *If $\mathsf{indeg}(u) = \mathsf{indeg}(v) = \mathsf{indeg}(w) = 1$, $(u, M) \notin \ell$, $(v, M) \notin \ell$ and there is an $x \in \mathsf{out}(v) \setminus \mathsf{out}(u)$ with $w \neq x$, then remove the arc (v, x) and add the arc (u, x).*
- *If $\mathsf{outdeg}(u) = \mathsf{outdeg}(v) = \mathsf{outdeg}(w) = 1$, $(v, F) \notin \ell$, $(w, F) \notin \ell$ and there is an $x \in \mathsf{in}(v) \setminus \mathsf{in}(w)$ with $u \neq x$, then remove the arc (x, v) and add the arc (x, w).*

Before proving that Shift Neighbors (RR 4) is safe, we need two simple observations about certain cases where we can safely exchange two arcs or add an arc.

▶ **Observation 3** ($\star$)**.** Let H be a funnel with funnel labeling ℓ and let $x, u, v \in V(H)$ such that $(v, x) \in A(H)$, $(u, x) \notin A(H)$ and at least one of the following is true: *(1)* $\ell(u) = \mathrm{F}$; or *(2)* $\ell(u) = \mathrm{M} = \ell(v)$ and $\mathsf{outdeg}_H(u) = 0$. Let $H' = H - (v, x) + (u, x)$. Then ℓ is also a funnel labeling for H' if H' is a DAG.

▶ **Observation 4 (⋆).** Let H be a DAG and $x, u, v \in V(H)$ such that $\{u\} = \mathsf{in}(v)$. Then $H + (u, x)$ contains a cycle if and only if $H + (v, x)$ contains a cycle.

Proof of safety of Shift Neighbors (RR 4). Consider the case where $\mathsf{indeg}(u) = \mathsf{indeg}(v) = \mathsf{indeg}(w) = 1$, $(u, \mathrm{M}) \notin \ell$, $(v, \mathrm{M}) \notin \ell$ and there is an $x \in \mathsf{out}(v) \setminus \mathsf{out}(u)$ with $w \neq x$. The other case follows analogously. Let (D', ℓ, k) be the reduced instance and $(S_r, \hat{\ell}_r)$ be a solution for it. We construct a solution $(S, \hat{\ell})$ for the input instance (D, ℓ, k).

First observe that, if $(u, x) \in S_r$, we can replace it with (v, x) in S, which means that $D' - S_r$ and $D - S$ are isomorphic. By setting $\hat{\ell} \coloneqq \hat{\ell}_r$, we obtain the desired solution. If $(u, x) \notin S_r$, we consider the following cases.

Case 1: $\hat{\ell}_r(v) = \mathrm{F}$. We set $\hat{\ell} \coloneqq \hat{\ell}_r$ and $S \coloneqq S_r$. Let $D^\star = D - S$. Clearly, $D^\star = D' - S_r - (u, x) + (v, x)$. As u is the only inneighbor of v, from Observation 4 we know $D^\star$ is a DAG. From Observation 3, we know that $\hat{\ell} = \hat{\ell}_r$ is a funnel labeling for $D^\star$.

Case 2: $\hat{\ell}_r(v) = \mathrm{M} = \hat{\ell}_r(u)$. If $D' - S_r + (u, v)$ is a DAG, we can assume that $(u, v) \notin S_r$, implying $(u, x) \in S_r$ (which was already considered).

If $D' - S_r + (u, v)$ is not a DAG, then it contains a cycle with v and u, implying $(u, v) \in S_r$. In particular, $\mathsf{indeg}_{D' - S_r}(v) = 0$. We set $\hat{\ell} \coloneqq \hat{\ell}_r$ and $\hat{\ell}(v) \coloneqq \mathrm{F}$. Clearly, $\hat{\ell}$ is a funnel labeling for $D' - S_r$. From Observation 3 we have that $\hat{\ell}$ is a funnel labeling for $D - S_r$ as well.

Case 3: $\hat{\ell}_r(v) = \mathrm{M}$ and $\hat{\ell}_r(u) = \mathrm{F}$. We set $\hat{\ell} \coloneqq \hat{\ell}_r$, $\hat{\ell}(v) \coloneqq \mathrm{F}$ and $S \coloneqq S_r$. As $\{u\} = \mathsf{in}_D(v)$ and $\hat{\ell}_r(u) = \mathrm{F}$, $\hat{\ell}$ is a funnel labeling for $D' - S_r$. Let $D^\star = D - S$.

From Observation 4 we know $D^\star = D' - S_r - (u, x) + (v, x)$ is a DAG since $D' - S_r$ is a DAG. Hence, from Observation 3 we obtain that $(S_r, \hat{\ell})$ is a solution for the input instance. In all cases a solution for the reduced instance implies a solution for the original instance.

Now assume there is a solution $(S, \hat{\ell})$ for the original instance. We show that there is solution $(S_r, \hat{\ell}_r)$ for the reduced instance. As in the previous direction, if $(v, x) \in S$, we can replace it with (u, x) and obtain the desired solution. So assume $(v, x) \notin S$.

If $(u, v) \in S$, let $S_1 = S \cup \{(y, u)\}$, where $\{y\} = \mathsf{in}(u)$. Clearly, $\hat{\ell}$ is a funnel labeling for $D - S_1$. We set $\hat{\ell}_r \coloneqq \hat{\ell}$ and $\hat{\ell}_r(u) \coloneqq \mathrm{F}$. As $\mathsf{indeg}_{D - S_1}(u) = 0$, $\hat{\ell}_r$ is also a funnel labeling for $D - S_1$. From Observation 3 we know that $\hat{\ell}_r$ is a funnel labeling for $D_1 = D' - S_1$. Since $\mathsf{indeg}_{D_1}(v) = 0 = \mathsf{indeg}_{D_1}(u)$ and $\hat{\ell}_r(u) = \mathrm{F}$, we have that $\hat{\ell}_r$ is a funnel labeling for $D_1 + (u, v)$. Hence, $(S \setminus \{(u, v)\}, \hat{\ell}_r)$ is a solution for the reduced instance.

In the following we consider the remaining cases where $\{(u, v), (v, x)\} \cap S = \emptyset$. Note that the case $\hat{\ell}(u) = \mathrm{M}$ and $\hat{\ell}(v) = \mathrm{F}$ does not happen under this assumption.

Case 1: $\hat{\ell}(v) = \mathrm{F} = \hat{\ell}(u)$. We set $\hat{\ell}_r \coloneqq \hat{\ell}$ and $S_r \coloneqq S$. Clearly, $D' - S_r = D - S - (v, x) + (u, x)$. From Observation 4 there is no cycle in $D - S + (u, x)$ and, hence, $D' - S_r$ is a DAG. Thus, from Observation 3 we have that $\hat{\ell}_r$ is a funnel labeling for $D' - S_r$.

Case 2: $\hat{\ell}(v) = \mathrm{M} = \hat{\ell}(u)$. Since $(v, x) \notin S$, we have $(v, w) \in S$ and $\hat{\ell}(x) = \mathrm{M}$. Further, we know that $D' - S$ is a DAG due to Observation 4. Let $S_1 = S \cup \{(u, v)\}$. Clearly, $\hat{\ell}$ is a funnel labeling for $D - S_1$, and $D' - S_1$ is also a DAG. From Observation 3 we have that $\hat{\ell}$ is a funnel labeling for $D' - S_1$.

We set $\hat{\ell}_r \coloneqq \hat{\ell}$ and $\hat{\ell}_r(v) \coloneqq \mathrm{F}$. Since $\mathsf{indeg}_{D' - S_1}(w) = 0 = \mathsf{indeg}_{D' - S_1}(v)$, we have that $\hat{\ell}_r$ is a funnel labeling for $D' - S_1 + (v, w)$, regardless of the label of w. By setting $S_r \coloneqq (S \setminus \{(v, w)\}) \cup \{(u, v)\}$, we get that $\hat{\ell}_r$ is a funnel labeling for $D' - S_r$ and $|S_r| \leq |S|$.

Case 3: $\hat{\ell}(v) = \mathrm{M}$ and $\hat{\ell}(u) = \mathrm{F}$. Let $S_r = S$ and $\hat{\ell}_r = \hat{\ell}$. Since $(u, v) \notin S_r$, from Observation 4 we know that $D - S_r - (v, x) + (u, x)$ is a DAG. From Observation 3 we have that $\hat{\ell}_r$ is a funnel labeling for $D' - S_r$.

In all cases we found a solution $(S_r, \hat{\ell}_r)$ for the reduced instance, concluding the proof. ◀

It is not always possible to exhaustively apply Shift Neighbors (RR 4): If u, v, w forms a cycle, we would shift x indefinitely through this cycle. To prevent this from happening, we need the following reduction rule:

▶ **Reduction Rule 5** (Break Cycle)**.** *Let C be a cycle in D. If every vertex in C has indegree (outdegree) one and either every vertex in C is unlabeled or every vertex in C is labeled with F (M), then delete one arc of C and decrease k by one.*

Proof of safety of Break Cycle (RR 5). Let (v, u) be the arc removed by the reduction rule. Clearly, a solution for the reduced instance together with the arc (v, u) is a solution for the original instance. Let $(S, \hat{\ell})$ be a solution for the original instance, and assume that $(v, u) \notin S$. Let (w, x) be an arc of C contained in S. Without loss of generality, we assume that (w, x) is the only incoming arc of x. The case where it is the only outgoing arc of w follows analogously.

We can assume that $\hat{\ell}(v) = \mathrm{F}$ for all $v \in V(C)$: If they were not labeled by ℓ when the rule was applied, then by repeatedly applying Set Label (RR 2) (starting with x) we can label them with F. Because $\mathsf{indeg}_D(v) = 1$ for every $v \in C$, it follows that C is the only cycle in D using the arc (w, x). Hence, $D' = D - S + (w, x) - (v, u)$ is a DAG. Further, as $\hat{\ell}(w) = \hat{\ell}(x) = \mathrm{F}$, it is easy to see that $\hat{\ell}$ is a funnel labeling for D'. ◀

If Shift Neighbors (RR 4) is not applicable, then many vertices in a long path P in a local funnel must share a common out- or inneighbor w. However, from Set Label (RR 2) we know that w receives a label if it has too many neighbors. The next and final reduction rule needed for bounding the number of unlabeled vertices exploits this property and allows us to label some vertex u in P if its predecessor v in P is adjacent to a labeled vertex w.

▶ **Reduction Rule 6** (Labeled Neighbor)**.** *Let (v, u) be an arc between unlabeled vertices. Set $\ell(u) \coloneqq F$ if $\mathsf{indeg}(u) = \mathsf{indeg}(v) = 1$ and $\exists w \in \mathsf{out}(v) : \ell(w) = M$. Set $\ell(v) \coloneqq M$ if $\mathsf{outdeg}(u) = \mathsf{outdeg}(v) = 1$ and $\exists w \in \mathsf{in}(u) : \ell(w) = F$.*

Proof of safety of Labeled Neighbor (RR 6). Assume, without loss of generality, that the first case of the rule was applied. The proof for the second case follows analogously (note that it is not possible for both cases to be applied simultaneously). Let (D, ℓ_r, k) be the reduced instance. First note that $\ell_r \supseteq \ell$, which means that a solution for the reduced instance is already a solution for the original instance. Hence, it suffices to show that a solution $(S, \hat{\ell})$ for the original instance implies a solution $(S_r, \hat{\ell}_r)$ for the reduced instance.

If $\hat{\ell}(u) = \mathrm{F}$, we set $\hat{\ell}_r \coloneqq \hat{\ell}$ and $S_r \coloneqq S$ and we are done. So assume that $\hat{\ell}(u) = \mathrm{M}$.

Case 1: $(v, u) \in S$. We set $S_r \coloneqq S$, $\hat{\ell}_r \coloneqq \hat{\ell}$ and $\hat{\ell}_r(u) \coloneqq \mathrm{F}$. As $\mathsf{indeg}_{D-S}(u) = 0$, we know that $\hat{\ell}_r$ is also a funnel labeling for $D - S$.

Case 2: $(v, u) \notin S$ and $\hat{\ell}(v) = \mathrm{F}$. We set $S_r \coloneqq S$, $\hat{\ell}_r \coloneqq \hat{\ell}$ and $\hat{\ell}_r(u) \coloneqq \mathrm{F}$. As $\hat{\ell}_r(v) = \mathrm{F} = \hat{\ell}_r(u)$, we may keep the arc (v, u) and $\hat{\ell}_r$ is a funnel labeling for $D - S_r$.

Case 3: $(v, u) \notin S$ and $\hat{\ell}(v) = \mathrm{M}$. Then $(v, w) \in S$. We set $\hat{\ell}_r \coloneqq \hat{\ell}$, $\hat{\ell}_r(u) \coloneqq \mathrm{F}$, $\hat{\ell}_r(v) \coloneqq \mathrm{F}$, $S_r \coloneqq (S \setminus \{(v, w)\}) \cup \{(y, v)\}$, where y is the unique inneighbor of v.

The digraph $D - S_r$ is a DAG: if it has a cycle, the cycle would have to use the arc (v, w), yet $\mathsf{indeg}_{D-S_r}(v) = 0$, a contradiction. We now argue that $\hat{\ell}_r$ is a funnel labeling for $D - S_r$. Since $\mathsf{indeg}_{D-S_r}(v) = 0$, $\mathsf{indeg}_{D-S_r}(u) = 1$ and $\hat{\ell}_r(u) = \mathrm{F}$, the vertex v is the unique inneighbor of u in the out-forest of the funnel $D - S_r$. Finally, as $\hat{\ell}_r(w) = \mathrm{M}$, the arc (v, w) is allowed in the funnel. Hence, $\hat{\ell}_r$ is a funnel labeling for $D - S_r$. In all cases we find a solution $(\hat{\ell}_r, S_r)$ for the reduced instance, concluding the proof. ◀

▶ **Lemma 5.** *Let s be some source (sink) of some unlabeled local funnel H in the reduced digraph D. Let $P_1, P_2, \dots P_a$ be a sequence of paths in H starting (ending) at s such that $\mathsf{indeg}(u) \leq 1$ ($\mathsf{outdeg}(u) \leq 1$) for each u in each P_i, and $V(P_j) \not\subseteq V(P_i)$ for all $1 \leq i, j \leq a$ where $i \neq j$. Let E be the set of end (start) points of all P_i. Then all of the following hold.*
(1) $\mathsf{outdeg}(u) > 1$ ($\mathsf{indeg}(u) > 1$) *for any inner vertex u of any P_i.*
(2) $\mathsf{out}(\bigcup_{i=1}^{a} V(P_i) \setminus E) \subseteq \mathsf{out}(s)$ ($\mathsf{in}(\bigcup_{i=1}^{a} V(P_i) \setminus E) \subseteq \mathsf{in}(s)$),
(3) $V(P_i) \cap V(P_j) = \{s\}$ *for each $1 \leq i, j \leq a$ where $i \neq j$, and*
(4) $a \leq k+1$ *and* $|V(P_i)| \leq k+2$ *for each $1 \leq i \leq a$.*

Proof. We consider the case where s is a source of H. The other case follows analogously.

Let u be some inner vertex of some P_i and w the unique outneighbor of u in P_i. By assumption on P_i, we have $\mathsf{indeg}_D(w) = 1$. As Dissolve Vertex (RR 3) is not applicable, we have that $\mathsf{outdeg}(u) > 1$ (proving (1)). In particular, u has some outneighbor x not in P_i.

Let v be the inneighbor of u in P_i. Since $\mathsf{indeg}_D(v) = \mathsf{indeg}_D(u) = \mathsf{indeg}_D(w) = 1$ and Shift Neighbors (RR 4) is not applicable, we have $x \in \mathsf{out}_D(v)$. By repeating this argument to the predecessors of u in P_i, we prove (2) (and also that $a \leq k+1$, as $\mathsf{outdeg}_D(s) \leq k+1$ due to Set Label (RR 2)).

Assume there are two paths P_i and P_j intersecting at more than one vertex. Let u be the last vertex of the intersection. Note that, if u is the last vertex of P_i or P_j, then one path has to contain the other. Hence, u has two outneighbors w_i and w_j lying on P_i and P_j, respectively, and $w_i \neq w_j$. But due to (2), we have $w_i, w_j \in \mathsf{out}_D(s)$, implying $\mathsf{indeg}_D(w_i) > 1$ and $\mathsf{indeg}_D(w_j) > 1$, a contradiction to our assumptions on P_i and P_j (proving (3)).

Let $v_1, v_2, \dots, v_m$ be the sequence of vertices of a path P_i. From (1) we know that there is some $w \in \mathsf{out}_D(v_{m-1})$ outside of P_i. We also have $w \in \mathsf{out}_D(v_j)$ for all $1 \leq j \leq m-1$, implying $\mathsf{indeg}_D(w) \geq m-1$. If $m-1 > k+1$, then $\ell(w) = \mathrm{M}$, as Set Label (RR 2) is not applicable. However, as $\mathsf{indeg}_D(v_{m-1}) = 1 = \mathsf{indeg}_D(v_{m-2})$, $w \in \mathsf{out}_D(v_{m-2})$ and Labeled Neighbor (RR 6) is not applicable, we have $\ell(v_{m-1}) = \mathrm{F}$, a contradiction to the assumption that H is unlabeled. Hence, $m-1 \leq k+1$, implying $|V(P_i)| \leq k+2$ (proving (4)). ◀

▶ **Lemma 6 ($\star$).** *Let H be an unlabeled local funnel in D. Then $|V(H)| \in \mathcal{O}(k^3)$.*

We conclude by bounding the number of maximal vertex-disjoint unlabeled local funnels in D. Since we can always partition unlabeled vertices with in- or outdegree at most one into local funnels, by bounding the number of local funnels in such a partitioning, together with the bound on the size of each local funnel, we obtain a bound for the number of unlabeled vertices with in- or outdegree at most one.

Let $\mathcal{H} = \{H_1, H_2, \dots H_a\}$ be a set of maximal vertex-disjoint unlabeled local funnels in D (in this context, maximal means that $H_i \cup H_j$ is not a local funnel for any two distinct $H_i, H_j \in \mathcal{H}$). Let s_i be the unique source of H_i for each i. We now show that, if there is a solution removing at most k arcs, then $|\mathcal{H}|$ is "small". By contraposition this means that, if $|\mathcal{H}|$ is "large", then we have a "no" instance and can stop the kernelization process.

We start with the simple observation that cycles intersecting inside a local funnel must also intersect outside it.

▶ **Observation 7 ($\star$).** Let C_i and C_j be two distinct cycles in D such that $V(C_i) \cap V(C_j) \subseteq H_\ell$ for some $H_\ell \in \mathcal{H}$. Then $V(C_i) \cap V(C_j) = \emptyset$.

We partition the set of maximal unlabeled local funnels $\mathcal{H}$ into three sets *(1)* $\mathcal{F} = \{H_i \in \mathcal{H} \mid$ there is some $v \in V(H_i)$ with $\mathsf{outdeg}_D(v) > 1\}$; *(2)* $\mathcal{M} = \{H_i \in \mathcal{H} \mid \mathsf{indeg}_D(s_i) > 1\}$; and *(3)* $\mathcal{X} = \{H_i \in \mathcal{H} \mid \mathsf{indeg}_D(s_i) = 1$ and $\forall v \in V(H_i) : \mathsf{outdeg}_D(v) = 1\}$.

▶ **Lemma 8.** *If there is a solution $(S, \hat{\ell})$ for (D, ℓ, k), then $|\mathcal{X}| \leq 2k^2$.*

Proof. Let $H_i \in \mathcal{X}$ and u be the unique inneighbor of s_i. Note that $\mathsf{outdeg}_D(s_i) = 1$. As Dissolve Vertex (RR 3) is not applicable, we have that $\mathsf{outdeg}_D(u) > 1$ and $\mathsf{indeg}_D(w) > 1$, where w is the unique outneighbor of s_i.

Case 1: $u \in \mathrm{Dom}(\ell)$. Then $\ell(u) = \mathrm{M}$ since Set Label (RR 2) is not applicable. As $\mathsf{outdeg}(u) > 1$, each $H_j \in \mathcal{X}$ with $s_j \in \mathsf{out}_D(u)$ requires one more arc of u to be in S.

Case 2: $u \notin \mathrm{Dom}(\ell)$. If $\mathsf{indeg}_D(u) = 1$, then there is some $v_i \in V(H_i)$ such that $(v_i, u) \in A(D)$, otherwise H_i would not be maximal. Hence, there is a cycle C_i containing u, s_i and v_i. If there is any other $H_j \in \mathcal{X}$ with $s_j \in \mathsf{out}_D(u)$ and with some $v_j \in V(H_j)$ such that (v_j, u), then the cycle C_j containing u, s_j and v_j is arc-disjoint to the cycle C_i due to Observation 7. Thus, S must contain at least one arc of each such C_j, implying there are at most k local funnels H_j that fall into this case.

If $\mathsf{indeg}_D(u) > 1$, one arc of u is in S as $\mathsf{outdeg}_D(u) > 1$. Further, $\mathsf{outdeg}_D(u) \leq k$. This means that there are at most k local funnels $H_j \in \mathcal{X}$ with $s_j \in \mathsf{out}_D(u)$. As there can be at most $2k$ such vertices u, we have that there are at most $2k^2$ local funnels $H_j \in \mathcal{X}$ which fall into this case. In the worst case, we have $|\mathcal{X}| \leq \max\{k + 1, 2k^2\} \leq 2k^2$. ◀

▶ **Lemma 9** (⋆). *If there is a solution $(S, \hat{\ell})$ for (D, ℓ, k), then $|\mathcal{F}| \leq 2k^2 + 3k$.*

▶ **Lemma 10** (⋆). *If there is a solution $(S, \hat{\ell})$ for (D, ℓ, k), then $|\mathcal{M}| \leq k^2 + 2k$.*

From Lemmas 8 to 10, we easily obtain a bound for the number of vertices in unlabeled local funnels. Together with the fact that Lower Bound (RR 1) is not applicable, we obtain a bound for the number of unlabeled vertices in D.

▶ **Lemma 11** (⋆). *Let D be a reduced digraph. Then there are $\mathcal{O}(k^5)$ vertices $v \in V(D)$ with $v \notin \mathrm{Dom}(\ell)$ and $\mathsf{indeg}(v) = 1 \vee \mathsf{outdeg}(v) = 1$.*

3.2 Bounding the number of labeled vertices

In Section 3.1 we exploited the property that unlabeled vertices have bounded degree, and that we can label them if their neighborhood has some special structure captured by the reduction rules. For the labeled vertices, however, we can apply neither of those strategies. Instead, we first exploit the fact that we know the label of a vertex and use this to decide if an arc is never in an optimal solution or if it is always in an optimal solution.

Arcs from M to F vertices clearly need to be removed. We show that we can also ignore arcs from F to M vertices, that is, we can remove them without changing k.

▶ **Reduction Rule 7** (Remove Arcs) (⋆). *Let $(v, u) \in A(D)$. If $\ell(v) = F$ and $\ell(u) = M$, remove (v, u). If $\ell(v) = M$ and $\ell(u) = F$, remove (v, u) and decrease k by 1.*

We now identify certain vertices that can be removed safely. Clearly, sources and sinks cannot be in any cycle in D. By carefully considering the neighborhood of a source or sink v, we can also prove that v is not "relevant" for any forbidden subgraph for funnels in D.

▶ **Reduction Rule 8** (Sources and Sinks) (⋆). *Let $v \in V(D)$ be a labeled vertex where $\mathsf{out}(v) \cup \mathsf{in}(v) \subseteq \mathrm{Dom}(\ell)$. Remove v if one of the following holds.*

1. $\mathsf{indeg}(v) = 0$ *and no* $u \in \mathsf{out}(v)$ *exists with* $\ell(u) = F$ *and* $\mathsf{indeg}(u) > 1$, *or*
2. $\mathsf{outdeg}(v) = 0$ *and no* $u \in \mathsf{in}(v)$ *exists with* $\ell(u) = M$ *and* $\mathsf{outdeg}(u) > 1$.

Having exhaustively applied Reduction Rules 7 and 8, we can bound the number of labeled vertices in D. Since Lower Bound (RR 1) is not applicable, we already have a bound for the number of vertices v with $\ell(v) = \mathrm{F} \wedge \mathsf{indeg}(v) > 1$ or $\ell(v) = \mathrm{M} \wedge \mathsf{outdeg}(v) > 1$. Hence, we only need to consider vertices in the set $L = \{v \in \mathrm{Dom}(\ell) \mid \ell(v) = \mathrm{F} \wedge \mathsf{indeg}(v) \leq 1 \text{ or } \ell(v) = \mathrm{M} \wedge \mathsf{outdeg}(v) \leq 1\}$.

To bound $|L|$, we exploit the bound on the number of unlabeled vertices from Lemma 11 and also the fact that such vertices have small degree as Set Label (RR 2) is not applicable. We first partition L into two subsets $L_1 = \{v \in L \mid \mathsf{in}(v) \cup \mathsf{out}(v) \not\subseteq \mathrm{Dom}(\ell)\}$ and $L_2 = L \setminus L_1$.

▶ **Lemma 12 (⋆).** $|L_1| \in \mathcal{O}(k^6)$.

▶ **Lemma 13.** $|L_2| \in \mathcal{O}(k)$.

Proof. Let $V_{\mathrm{F}} = \{v \mid \ell(v) = \mathrm{F}\}$ and $L_{\mathrm{F}} = V_{\mathrm{F}} \cap L_2$. The case for the vertices labeled with M follows analogously.

Since Remove Arcs (RR 7) is not applicable, we have $\ell(u) = \mathrm{F}$ for all $u \in \mathsf{out}(L_{\mathrm{F}}) \cup \mathsf{in}(L_{\mathrm{F}})$. Let

$R_1 = \{u \in V_{\mathrm{F}} \mid \mathsf{indeg}(u) > 1\}$, $R_2 = \{u \in L_{\mathrm{F}} \mid \mathsf{indeg}(u) \leq 1, \mathsf{out}(u) \cap R_1 \neq \emptyset\}$ and $R_3 = \{u \in L_{\mathrm{F}} \mid \mathsf{indeg}(u) \leq 1, \mathsf{out}(u) \cap R_1 = \emptyset\}$.

Note that $L_2 = R_2 \cup R_3$ and $R_1 \cap L_2 = \emptyset$.

A solution set $S \subseteq A(D)$ must contain at least $\mathsf{indeg}(v) - 1$ many incoming arcs of v for every $v \in R_1$. As each $u \in R_2$ has some $v \in R_1$ as outneighbor, we have $|R_2| \leq 2k$.

Let $v \in R_3$. We claim that v can reach some vertex of R_2. Since Sources and Sinks (RR 8) is not applicable and $\mathsf{out}(v) \cap R_1 = \emptyset$, we have $\mathsf{indeg}(v) = 1$ and $\mathsf{outdeg}(v) \geq 1$. This means that, if we successively follow the outneighbors of v, we reach a vertex of R_2 or find a cycle C using only vertices of R_3. However, as Break Cycle (RR 5) is not applicable, such a cycle C cannot exist: every vertex $v \in R_3$ has $\mathsf{indeg}(v) = 1$ and $\ell(v) = \mathrm{F}$, implying we could apply Break Cycle (RR 5) to C. Hence, every vertex of R_3 can reach some $u \in R_2$.

We greedily construct vertex-disjoint paths $P_1, P_2, \ldots, P_a$ ending in R_2 whose inner vertices lie in R_3. For a vertex $v \in R_3$ take an arbitrary $u \in R_2$ such that v can reach u. Consider a path P from v to u. If none of its vertices lie in any already constructed P_i, we just take the path P into our set of paths. Otherwise, assume that P intersects some P_i at w and let w be the first such vertex in P. Since the indegree of any vertex in $R_3 \cup R_2$ is at most one, we know that w is the starting point of P_i. Hence, we can obtain a path P_j by taking the path from v to w in P and then concatenating P_i. As w is the first vertex of P intersecting any other path, we get that P_j only intersects P_i. By replacing P_i with P_j, we obtain a path that also contains v. We repeat this process until we covered all $v \in R_3$.

Since $|R_2| \leq 2k$, we have $a \leq 2k$. We now prove that $|V(P_i)| \leq 4$ for any P_i in our set of vertex-disjoint paths. Note that $\mathsf{indeg}(u) \leq 1$ for any vertex $u \in V(P_i)$.

Since Dissolve Vertex (RR 3) is not applicable, any inner vertex u of P_i has $\mathsf{outdeg}(u) > 1$. Let w be the successor of u in P_i. As Shift Neighbors (RR 4) is not applicable, we have that $\mathsf{indeg}(w) > 1$ or $v \in \mathsf{out}(u)$ where v is the unique inneighbor of u. If $\mathsf{indeg}(w) > 1$, then $u \in R_2$ and is the endpoint of P_i, a contradiction to the assumption that u is an inner vertex of P_i. Otherwise, we know that $u \notin \mathsf{out}(w)$ as $\mathsf{indeg}(u) = 1$. If u is the only inner vertex of P_i, then $|V(P_i)| \leq 3$. Otherwise, its successor w in P_i is an inner vertex of P_i (since v is the starting point of P_i, and so $v \notin \mathsf{out}(u)$). Hence, we can apply the same argumentation to w and conclude that it has some outneighbor x with $\mathsf{indeg}(x) > 1$, implying $x \in R_2$ and $|V(P_i)| \leq 4$.

Since $|V(P_i)| \leq 4$ and $a \leq 2k$, we have that $|L_2| \leq 6k$. Because $L_2 = R_2 \cup R_3$, we have that $|L_2| \leq 8k \in \mathcal{O}(k)$, as desired. ◀

▶ **Lemma 14.** *Let (D, ℓ, k) be an FADL instance where Reduction Rules 1 to 8 are not applicable. Then $|V(D)| \in \mathcal{O}(k^6)$ and $|A(D)| \in \mathcal{O}(k^6)$.*

Proof. As Lower Bound (RR 1) is not applicable, there are at most $2k$ vertices v with $\mathsf{indeg}(v) > 1$ and $\mathsf{outdeg}(v) > 1$, and also at most $2k$ many vertices v with $\ell(v) = \mathrm{F} \wedge \mathsf{indeg}(v) > 1$ or $\ell(v) = \mathrm{M} \wedge \mathsf{outdeg}(v) > 1$. From Lemma 11 we know there are $\mathcal{O}(k^5)$ many unlabeled vertices $v \in V(D)$ with $\mathsf{indeg}(v) \leq 1$ or $\mathsf{outdeg}(v) \leq 1$. Finally, due to Lemmas 12 and 13 there are $\mathcal{O}(k^6)$ vertices v with $\ell(v) = \mathrm{F} \wedge \mathsf{indeg}(v) \leq 1$ or $\ell(v) = \mathrm{M} \wedge \mathsf{outdeg}(v) \leq 1$. As any vertex in D falls into one of these groups, we have $|V(D)| \in \mathcal{O}(k^6)$.

As Remove Arcs (RR 7) is not applicable, there is no arc (v, u) where $v, u \in \mathrm{Dom}(\ell)$ and $\ell(v) \neq \ell(u)$. Since there are $\mathcal{O}(k^5)$ many unlabeled vertices and every unlabeled vertex has in- and outdegree at most $k + 1$, there are $\mathcal{O}(k^6)$ arcs (v, u) where $v \notin \mathrm{Dom}(\ell)$ or $u \notin \mathrm{Dom}(\ell)$.

Now let (v, u) be some arc where $v, u \in \mathrm{Dom}(\ell)$. Note that $\ell(v) = \ell(u)$.

Case 1: $v, u \in L$. Then $\mathsf{outdeg}(v) = 1$ (if $\ell(v) = \mathrm{M}$) or $\mathsf{indeg}(u) = 1$ (if $\ell(u) = \mathrm{F}$). Thus, there can be at most $|L| \in \mathcal{O}(k^6)$ many arcs (v, u) where $v, u \in L$.

Case 2: $v, u \notin L$. As Lower Bound (RR 1) is not applicable, there can be at most $2k$ such vertices. Thus, there are at most $4k^2$ arcs between labeled vertices not in L.

Case 3: Exactly one of v, u is in L.

Case 3.1: $v \notin L \wedge \ell(v) = \mathrm{F}$ or $u \notin L \wedge \ell(u) = \mathrm{M}$. Then $\mathsf{indeg}(u) = 1$ or $\mathsf{outdeg}(v) = 1$. Hence, there can be at most $|L| \in \mathcal{O}(k^6)$ such arcs.

Case 3.2: $v \notin L \wedge \ell(v) = \mathrm{M}$ or $u \notin L \wedge \ell(u) = \mathrm{F}$. If $v \notin L$, then at least half of its outgoing arcs need to be in a solution set. Similarly, if $u \notin L$, at least half of its incoming arcs need to be in a solution set. Hence, there can be at most $2k$ many arcs falling into this case. By adding all cases together, we obtain that $|A(D)| \in \mathcal{O}(k^6)$, concluding the proof. ◀

4 Computing the Kernel

In Sections 3.1 and 3.2 we defined the reduction rules for the kernelization process and showed that, if none of the reduction rules are applicable to a digraph D, then the size of D is polynomially bounded on k. To conclude the proof that FADS admits a polynomial problem kernel, we show that it is possible to apply all reduction rules in $\mathcal{O}(nm)$ time and also reduce the FADL instance back into an FADS instance.

▶ **Lemma 15 (⋆).** *We can exhaustively apply Reduction Rules 1 to 8 in $\mathcal{O}(nm)$ time to an FADL instance (D, ℓ, k), where $n = |V(D)|$ and $m = |A(D)|$.*

▶ **Theorem 16.** *FADS admits a kernel with $\mathcal{O}(k^6)$ vertices and $\mathcal{O}(k^7)$ arcs which can be computed in $\mathcal{O}(nm)$ time, where $n = |V(D)|$, $m = |A(D)|$ and D is the input digraph.*

Proof. We start by reducing the FADS instance into an FADL instance (D, ℓ, k) by adding an empty labeling ℓ. Using Lemma 15, we can exhaustively apply all reduction rules to (D, ℓ, k) in $\mathcal{O}(nm)$ time.

From Lemma 14 we know $|V(D)| \in \mathcal{O}(k^6)$ and $|A(D)| \in \mathcal{O}(k^6)$. We now reduce the FADL instance back into an FADS instance (D', k) in order to obtain a kernel for the original problem. We first set $D' \coloneqq D$ and add $k + 2$ vertices $f_1, f_2, \ldots, f_{k+2}$ and $k + 2$ vertices $m_1, m_2, \ldots, m_{k+2}$ to D'. Let $v \in \mathrm{Dom}(\ell)$. If $\ell(v) = \mathrm{F}$, we add the arc (v, f_i) for each $1 \leq i \leq k + 2$. If $\ell(v) = \mathrm{M}$, we add the arc (m_i, v) for each $1 \leq i \leq k + 2$.

Trivially, a solution for the FADL instance is also a solution for the FADS instance. It is also easy to see that, if there is some arc set $S_r \subseteq A(D')$ and some funnel labeling $\hat{\ell}_r$ for $D' - S_r$ such that $\ell(v) \neq \hat{\ell}_r(v)$ for some $v \in \mathrm{Dom}(\ell)$, then $|S_r| > k$. Hence, a solution for (D', k) implies a solution for (D, ℓ, k).

We added $2k + 4$ vertices and $\mathcal{O}(k^7)$ many arcs to D', and so $|V(D')| \in \mathcal{O}(k^6)$ and $|A(D')| \in \mathcal{O}(k^7)$, thus concluding the proof. ◀

5 Conclusion

The kernelization algorithm provided in this paper heavily relies on the characterizations of Theorem 1 for funnels. Both the characterization by forbidden subgraphs as well as the labeling characterization allowed us to derive reduction rules based only on "local" substructures as the degree or neighborhood of a vertex. In a sense, this "locality" property saved us from computing any set of vertex-disjoint local funnels, despite the fact that the results and reduction rules from Section 3.1 heavily rely on local funnels.

The polynomial kernels for Out-Forest-VDS and Pumpkin-VDS due to [12] also rely on "localized" forbidden substructures. We consider that generalizing these results to larger digraph classes of unbounded treewidth, but which are characterized by forbidden substructures, to be a very interesting direction for future research.

Further, it would also be interesting to decide if Funnel-VDS admits a polynomial kernel or not (it is in FPT with respect to the solution size [11]), especially since a kernel for this problem would require considerably different ideas from the ones presented in this paper, as it is no longer clear how to exploit the vertex labeling in the vertex-deletion setting.

References

1 Faisal N Abu-Khzam. A kernelization algorithm for d-hitting set. *Journal of Computer and System Sciences*, 76(7):524–531, 2010.

2 Akanksha Agrawal, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Kernels for deletion to classes of acyclic digraphs. *Journal of Computer and System Sciences*, 92:9–21, 2018.

3 Jørgen Bang-Jensen, Alessandro Maddaloni, and Saket Saurabh. Algorithms and kernels for feedback set problems in generalizations of tournaments. *Algorithmica*, 76(2):320–343, 2016.

4 Jianer Chen, Yang Liu, Songjian Lu, Barry O'sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM (JACM)*, 55(5):21, 2008.

5 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015.

6 Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

7 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.

8 Stefan Kratsch. Recent developments in kernelization: A survey. *Bulletin of EATCS*, 2(113), 2014.

9 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization–preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond*, pages 129–161. Springer, 2012.

10 Daniel Lokshtanov, MS Ramanujan, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Wannabe bounded treewidth graphs admit a polynomial kernel for dfvs. In *Workshop on Algorithms and Data Structures*, pages 523–537. Springer, 2019.

11 Marcelo Garlet Millani, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Efficient algorithms for measuring the funnel-likeness of DAGs. In Jon Lee, Giovanni Rinaldi, and A. Ridha Mahjoub, editors, *Combinatorial Optimization*, pages 183–195, Cham, 2018. Springer International Publishing.

12 Matthias Mnich and Erik Jan van Leeuwen. Polynomial kernels for deletion to classes of acyclic digraphs. *Discrete Optimization*, 25:48–76, 2017.

FPT Approximation for Constrained Metric k-Median/Means

Dishant Goyal
Indian Institute of Technology Delhi, India
Dishant.Goyal@cse.iitd.ac.in

Ragesh Jaiswal[1]
Indian Institute of Technology Delhi, India
rjaiswal@cse.iitd.ac.in

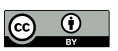
Amit Kumar
Indian Institute of Technology Delhi, India
amitk@cse.iitd.ac.in

Abstract

The Metric k-median problem over a metric space $(\mathcal{X}, d)$ is defined as follows: given a set $L \subseteq \mathcal{X}$ of *facility locations* and a set $C \subseteq \mathcal{X}$ of *clients*, open a set $F \subseteq L$ of k facilities such that the total service cost, defined as $\Phi(F, C) \coloneqq \sum_{x \in C} \min_{f \in F} d(x, f)$, is minimised. The metric k-means problem is defined similarly using squared distances (i.e., $d^2(.,.)$ instead of $d(.,.)$). In many applications there are additional constraints that any solution needs to satisfy. For example, to balance the load among the facilities in resource allocation problems, a capacity u is imposed on every facility. That is, no more than u clients can be assigned to any facility. This problem is known as the *capacitated* k-means/k-median problem. Likewise, various other applications have different constraints, which give rise to different *constrained* versions of the problem such as *r-gather*, *fault-tolerant*, *outlier* k-means/k-median problem. Surprisingly, for many of these constrained problems, no constant-approximation algorithm is known. Moreover, the *unconstrained* problem itself is known [1] to be W[2]-hard when parameterized by k. We give FPT algorithms with constant approximation guarantee for a range of constrained k-median/means problems. For some of the constrained problems, ours is the first constant factor approximation algorithm whereas for others, we improve or match the approximation guarantee of previous works. We work within the unified framework of Ding and Xu [24] that allows us to simultaneously obtain algorithms for a range of constrained problems. In particular, we obtain a $(3 + \varepsilon)$-approximation and $(9 + \varepsilon)$-approximation for the constrained versions of the k-median and k-means problem respectively in FPT time. In many practical settings of the k-median/means problem, one is allowed to open a facility at any client location, i.e., $C \subseteq L$. For this special case, our algorithm gives a $(2 + \varepsilon)$-approximation and $(4 + \varepsilon)$-approximation for the constrained versions of k-median and k-means problem respectively in FPT time. Since our algorithm is based on simple sampling technique, it can also be converted to a constant-pass log-space streaming algorithm. In particular, here are some of the main highlights of this work:

1. For the uniform capacitated k-median/means problems our results matches previously known results of Addad et al. [19].
2. For the r-gather k-median/means problem (clustering with lower bound on the size of clusters), our FPT approximation bounds are better than what was previously known.
3. Our approximation bounds for the fault-tolerant, outlier, and uncertain versions is better than all previously known results, albeit in FPT time.
4. For certain constrained settings such as chromatic, l-diversity, and semi-supervised k-median/means, we obtain the first constant factor approximation algorithms to the best of our knowledge.
5. Since our algorithms are based on a simple sampling based approach, we also obtain constant-pass log-space streaming algorithms for most of the above-mentioned problems.

[1] Part of this work was done while the author was on a sabbatical from IIT Delhi and visiting UC San Diego.



15th International Symposium on Parameterized and Exact Computation (IPEC 2020).
Editors: Yixin Cao and Marcin Pilipczuk; Article No. 14; pp. 14:1–14:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Facility location and clustering

Keywords and phrases k-means, k-median, approximation algorithms, parameterised algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.14

Related Version A full version of the paper is available at `https://arxiv.org/abs/2007.11773`.

Acknowledgements The authors would like to thank Anup Bhattacharya for useful discussions. Dishant Goyal would like to thank TCS Research Scholar Program.

1 Introduction

The metric k-means and k-median problems are similar. We combine the discussion of these problems by giving a definition of the k-service problem that encapsulates both these problems.

▶ **Definition 1** (k-service problem). *Let $(\mathcal{X}, d)$ be a metric space, $k > 0$ be any integer and $\ell \geq 0$ be any real number. Given a set $L \subseteq \mathcal{X}$ of feasible facility locations, and a set $C \subseteq \mathcal{X}$ of clients, find a set $F \subseteq L$ of k facilities that minimises the total service cost: $\Phi(F, C) \equiv \sum_{j \in C} \min_{i \in F} d^{\ell}(i, j)$.*

Note that the k-service problem is also studied with respect to a more general cost function $\sum_{j \in C} \min_{i \in F} \delta(i, j)$, where $\delta(i, j)$ denotes the cost of assigning a client $j \in C$ to a facility $i \in F$. We consider the special case $\delta(i, j) \equiv d^{\ell}(i, j)$. For $\ell = 1$, the problem is known as the *k-median* problem and for $\ell = 2$, the problem is known as the *k-means* problem. The above definition is motivated by the *facility location* problem [43] and differs from it in two ways. First, in the facility location problem, one is allowed to open any number of facilities. Second, one has to pay for an additional facility establishment cost for every open facility. Thus the k-service problem is basically the facility location problem for a fixed number of facilities and 0 facility establishment costs.

The k-service problem can also be viewed as a clustering problem, where the goal is to group the objects that are similar to each other. Clustering algorithms are commonly used in data mining, pattern recognition, and information retrieval [33]. However, the notion of a cluster differs for different applications. For example, some applications consider a cluster as a dense region of points in the data-space [25, 5], while others consider it as a highly connected subgraph of a graph [32]. Likewise, various models have been developed in the past that capture the clustering properties in different ways [47]. The k-means and k-median problems are examples of the *center-based* clustering model. In this model, the objects are mapped to the points in a metric space such that the distance between the points captures the degree of dissimilarity between them. In other words, the closer the two points are, the more similar they are to each other. In order to measure the quality of a clustering, a center (known as the cluster representative) is assigned to each cluster and the cost is measured based on the distances of the points to their respective cluster centers. Then the problem objective is to obtain a clustering with the minimum cost. To view the k-median instance as a clustering instance, consider the client set as a set of data points and the facility locations as the feasible centers. In a feasible solution, the clients which are assigned to the same facility are considered a part of the same cluster and the corresponding facility act as their cluster center. During our discussion, we will use the term *center* and *facility* interchangeably. Similarly, we can view the k-means problem as a clustering problem where the cost is measured with respect to the squared distances.

Various variants of the k-median/means problem have been studied in the clustering literature. For example, the Euclidean k-means problem (where $C \subseteq L = \mathbb{R}^d$) is NP-hard even for a fixed k or a fixed dimension d [22, 4, 41, 45]. This opens the question of designing a PTAS (polynomial-time approximation schemes) for the problem when either the number of clusters or the dimension is fixed. Indeed, various PTASs are known under such conditions [38, 26, 15, 35, 27, 18]. In general, it is known that the problem can not be approximated within a particular constant factor, unless $\mathsf{P} = \mathsf{NP}$ [8, 16].

The hardness results in the previous paragraph was for Euclidean setting. These problems may be harder in general metric spaces which is indeed what has been shown. The metric k-median problem is hard to approximate within a factor strictly smaller than $(1 + 2/e)$, and the metric k-means problem is hard to approximate within a factor strictly smaller than $(1 + 8/e)$ [29, 34]. On the positive side, various constant-factor approximation algorithms are known for the k-means (and k-median) problems in the metric and Euclidean settings [36, 14, 7, 30, 40, 11, 3]. Improving these bounds is not the goal of this paper. Instead, we undertake the task of improving/obtaining approximation bounds of a more general class of problems called the *constrained k-means/k-median* problem. Let us see what these problems are and why they are important.

For many real-world applications, the classical (unconstrained) k-means and k-median problems do not entirely capture the desired clustering properties. For example, consider the popular *k-anonymity* principle [44]. The principle provides anonymity to a public database while keeping it meaningful at the same time. One way to achieve this is to cluster the data in such a way to release only partial information related to the clusters obtained. Further, to protect the data from the *re-identification* attacks, the clustering should be done in such a way that each cluster gets at least r data-points. This method is popularly known as *r-gather* clustering [2] (see the formal definition in Table 1). Likewise, various other applications impose a specific set of constraints on the clusters. Such applications have been studied extensively. A survey on these applications is mentioned in Section 1.1 of [24]. We collectively mention these problems in Table 1 and their known approximation results in Table 2. We discuss these problems and their known results in detail in the full version of the paper.

An important distinction between the constrained problems and their unconstrained counterparts is the idea of *locality*. In simple words, the *locality* property says that the points which are close to each other should be part of the same cluster. This property holds for the unconstrained version of the problem. However, this may not necessarily hold for many of the constrained versions of the problem where minimising clustering cost is not the only requirement. To understand this, consider a center-set $F = \{f_1, f_2, \ldots, f_k\}$ and let $\{C_1, ..., C_k\}$ denote the clustering of the dataset such that the cost function gets minimised. That is, C_i contain all the points for which f_i is the closest center in the set F. Note that the clustering $\{C_1, ..., C_k\}$ just minimises the distance based cost function and may not satisfy any additional constraint that the clustering may need to satisfy in a constrained setting. In a constrained setting we may need an algorithm that, given a center-set $\{f_1, ..., f_k\}$ as input, outputs a clustering $\{\bar{C}_1, ..., \bar{C}_k\}$ which in addition to minimising $\sum_i \sum_{x \in \bar{C}_i} d^\ell(x, f_i)$ also satisfies certain clustering constraints. Such an algorithm is called a *partition algorithm*. In the unconstrained setting, the partition algorithm simply assigns points to closest center in F. However, designing such an efficient partition algorithm for the constrained versions of the problem is a non-trivial task. Ding and Xu [24] gave partition algorithms for all the problems mentioned in Table 1 (see Section 4 and 5.3 of [24]). Though these algorithms were specifically designed for the Euclidean space, they can be generalized to any metric space. We will see that such a partition algorithm is crucial in the design of our FPT algorithms.

Table 1 Constrained k-service problems with efficient partition algorithm (see Section 4 and 5.3 in [24] and references therein). The (*) marked problems were not discussed in [24]. Note that for problems 1 and 2, the bounds are cluster-wise and not facility-wise. Please see definition of $\Psi(F, \xi)$ and $\Psi^*(\xi)$ below (see eqn. (1) and defn. 2).

#	Problem	Description
1.	r-gather k-service problem* (r,k)-`GService`	Find clustering $\xi = \{C_1, ..., C_k\}$ with minimum $\Psi^*(\xi)$ such that for all i, $\lvert C_i\rvert \geq r_i$
2.	r-Capacity k-service problem* (r,k)-`CaService`	Find clustering $\xi = \{C_1, ..., C_k\}$ with minimum $\Psi^*(\xi)$ such that for all i, $\lvert C_i\rvert \leq r_i$
3.	l-Diversity k-service problem (l,k)-`DService`	Given that every client has an associated colour, find a clustering $\xi = \{C_1, ..., C_k\}$ with minimum $\Psi^*(\xi)$ such that for all i, the fraction of points sharing the same colour inside C_i is $\leq \frac{1}{l}$
4.	Chromatic k-service problem k-`ChService`	Given that every client has an associated colour, find a clustering $\xi = \{C_1, ..., C_k\}$ with minimum $\Psi^*(\xi)$ such that for all i, C_i should not have any two points with the same colour.
5.	Fault tolerant k-service problem (l,k)-`FService`	Given a value l_p for every client, find a clustering $\xi = \{C_1, ..., C_k\}$ and a set F of k centers, such that the sum of service cost of the points to l_p of nearest centers out of $F = \{f_1, f_2, \ldots, f_k\}$, is minimised.
6.	OWA k-service problem* k-`OWAService`	Given a vector $(w_1, ..., w_k)$ of non-increasing weights, find a center set $\{f_1, ..., f_k\}$ such that $\sum_{x \in C} \sum_{j=1}^{k} w_j \cdot \left(\overline{d}_j(x)\right)^{\ell}$ is minimised. Here, $\left(\overline{d}_1(x), ..., \overline{d}_k(x)\right)$ is a non-decreasing ordering of $\left(d(x, f_1), ..., d(x, f_k)\right)$.
7.	Semi-supervised k-service problem k-`SService`	Given a target clustering $\xi' = \{C'_1, ..., C'_k\}$ and constant α, find a clustering $\xi = \{C_1, ..., C_k\}$ and a center set F, such that the cost $\overline{\Psi}(F, \xi) \coloneqq \alpha \cdot \Psi(F, \xi) + (1 - \alpha) \cdot Dist(\xi', \xi)$ is minimised. $Dist$ denotes the set-difference distance.
8.	Uncertain k-service problem k-`UService`	Given a discrete probability distribution for every client, i.e., for a point $p \in C$ there is a set $D_p = \{p_1, \ldots, p_h\}$ such that p takes the value p_i with probability t_p^i and $\sum_{i=1}^{h} t_p^i \leq 1$. Find a clustering $\xi = \{C_1, ..., C_k\}$ so that the expected cost of $\Psi^*(\xi)$ is minimized.
9.	Outlier k-service problem* (k,m)-`OService`	Find a set $Z \subseteq C$ of size m and a clustering $\mathcal{C}' = \{C'_1, ..., C'_k\}$ of the set $C' \coloneqq C \setminus Z$, such that $\Psi^*(\xi')$ is minimized.

The partition algorithm gives us a way for going from center-set to clustering. What about the reverse direction? Given a clustering $\xi = \{C_1, C_2, \ldots, C_k\}$, can we find a center set that gives minimum clustering cost? The solution to this problem is simple. Construct a complete weighted bipartite graph $G = (V_l, V_r, E)$, where a vertex in V_l corresponds to a facility location in L, and a vertex in V_r corresponds to a cluster $C_j \in \xi$. The weight on an edge $(i, j) \in V_l \times V_r$ is equal to the cost of assigning the cluster C_j to the i^{th} facility, i.e., $\sum_{x \in C_j} d^{\ell}(x, i)$. Then we can easily obtain an optimal assignment by finding the minimum cost perfect matching in the graph G. Let us denote the minimum cost by $MCPM(\xi, L)$ Thus, it is sufficient to output an optimal clustering for a constrained k-service instance. In fact, all problems in Table 1 only requires us to output an optimal clustering for the problem.

Ding and Xu [24] suggested the following unified framework for considering any constrained k-means/k-median problem by modelling an arbitrary set of constraints using feasible clusterings. Note that they studied the problem in the Euclidean space where $C \subseteq L = \mathbb{R}^d$ whereas we study the problem in general metric space where L and C are discrete and

separate sets. We will use a few more definitions to define the problem. A k-center-set is a set of k distinct elements from L and for any k-center-set $F = \{f_1, ..., f_k\}$ and a clustering $\xi = \{C_1, ..., C_k\}$, we will use the cost function:

$$\Psi(F, \xi) \equiv \min_{\text{permutation } \pi} \left\{ \sum_{i=1}^{k} \sum_{x \in C_i} d^{\ell}(x, f_{\pi(i)}) \right\}. \tag{1}$$

▶ **Definition 2** (Constrained k-service problem)**.** *Let $(\mathcal{X}, d)$ be a metric space, $k > 0$ be any integer and $\ell \geq 0$ be any real number. Given a set $L \subseteq \mathcal{X}$ of feasible facility locations, a set $C \subseteq \mathcal{X}$ of clients, and a set $\mathbb{C}$ of feasible clusterings, find a clustering $\xi = \{C_1, C_2, \ldots, C_k\}$ in $\mathbb{C}$, that minimizes the following objective function: $\Psi^*(\xi) \equiv \min_{k\text{-center-set } F} \Psi(F, \xi)$.*

Note that $\Psi^*(\xi)$ is $MCPM(\xi, L)$, the minimum cost perfect matching as discussed earlier. The key component of the above definition is the set of feasible clusterings $\mathbb{C}$. Using this, we can define any constrained version of the problem. Note that $\mathbb{C}$ can have an exponential size. However, for many problems it can be defined concisely using a simple set of mathematical constraints. For example, $\mathbb{C}$ for the r-gather problem can be defined as $\mathbb{C} := \{\xi \mid \text{for every cluster } C_i \in \xi, |C_i| \geq r_i\}$, where $\xi = \{C_1, C_2, \ldots, C_k\}$ is a partitioning of the client set. Note that we consider the *hard assignment* model for the problem. That is, one cannot open more than one facility at a location. It differs from the *soft assignment* model where one can open multiple facilities at a location. The soft version can be stated in terms of the hard version – by allowing L to be a multi-set and creating k-copies for each location in L. It has been observed that the soft-assignment models are easier and allow better approximation guarantees than the hard-assignment models [21, 39]. For our discussion, we will call a center-set a *soft center-set* if it contains facility location multiple times, otherwise we call it a *hard center-set*. In fact, a soft center-set is a multi-set. We will avoid using the term multi-set to keep our discussion simple.

As observed in past works [24, 9], any constrained version of k-median/means can be solved using a partition algorithm for this version and a solution to a very general "list" version of the clustering problem which we discuss next. Let us define this problem which we call the *list k-service* problem [2]. This will help us solve the constrained k-service problem.

▶ **Definition 3** (List k-service problem)**.** *Let α be a fixed constant. Let $\mathcal{I} = (L, C, k, d, \ell)$ be any instance of the k-service problem and let $\xi = \{C_1, C_2, \ldots, C_k\}$ be an arbitrary clustering of the client set C. The goal of the problem is: given $\mathcal{I}$, find a list $\mathcal{L}$ of k-center-sets (i.e., each element of the list is a set of k distinct elements from L) such that, with probability at least $(1/2)$, $\mathcal{L}$ contains a k-center-set F such that $\Psi(F, \xi) \leq \alpha \cdot \Psi^*(\xi)$.*

Note that the clustering algorithm in the above setup does not get access to the clustering ξ and yet is supposed to find good centers (constant α approximation) for this clustering. Given this, it is easy to see that finding a single set of k centers that are good for ξ is not possible. However, finding a reasonably small list of k-center-sets such that at least one of the k-center-sets in the list is good may be feasible. This is main realization behind the formulation of the list version of the problem. The other reason is that since the target clustering is allowed to be a completely arbitrary partition of the client set C, we can use the solution of the list k-service problem to solve any constrained k-service problem as

[2] This notion of list version of the clustering problem was implicitly present in the work of Ding and Xu [24]. Bhattacharya et al. [9] formalized this as the *list k-means problem*.

long as there is a partition algorithm. The following theorem combines the list k-service algorithm and the partition algorithm for a constrained version of the problem to produce a constant-approximation algorithm this problem.

▶ **Theorem 4.** *Let $\mathcal{I} = (L, C, k, d, \ell, \mathbb{C})$ be any instance of any constrained k-service problem and let $A_{\mathbb{C}}$ be the corresponding partition algorithm. Let B be an algorithm for the list k-service problem that runs in time T_B for instance (L, C, k, d, ℓ). There is an algorithm that, with probability at least $1/2$, outputs a clustering $\xi \in \mathbb{C}$, which is an α-approximation for the constrained k-service instance. The running time of the algorithm is $O(T_B + |\mathcal{L}| \cdot T_A)$, where T_A is the running time of the partition algorithm.*

Proof. The algorithm is as follows. We first run algorithm B to obtain a list $\mathcal{L}$. For every k-center-set in the list, the algorithm runs the partition algorithm $A_{\mathbb{C}}$ on it. Then the algorithm outputs that k-center-set that has the minimum clustering cost. Let F' be this k-center-set and ξ' be the corresponding clustering. We claim that (F', ξ') is an α-approximation for the constrained k-service problem with probability at least $1/2$.

Let ξ^* be an optimal solution for the constrained k-service instance $(L, C, k, d, \ell, \mathbb{C})$ and F^* denote the corresponding k-center-set. By the definition of the list k-service problem, with probability at least $1/2$, there is a k-center-set F in the list $\mathcal{L}$, such that $\Psi(F, \xi^*) \leq \alpha \cdot \Psi(F^*, \xi^*)$. Let $\xi = A_{\mathbb{C}}(F) \in \mathbb{C}$ be the clustering corresponding to F. Thus, $\Psi(F, \xi) \leq \Psi(F, \xi^*)$ and so with probability at least $1/2$, $\Psi(F, \xi) \leq \alpha \cdot \Psi(F^*, \xi^*)$. Since F' gives the minimum cost clustering in the list, we have $\Psi(F', \xi') \leq \Psi(F, \xi)$. Therefore, with probability at least $1/2$, $\Psi(F', \xi') \leq \alpha \cdot \Psi(F^*, \xi^*)$.

Since, the algorithm runs a partition procedure for every center set in the list, the running time of this step is $|\mathcal{L}| \cdot T_A$. Picking a minimum cost clustering from the list takes $O(|\mathcal{L}|)$ time. Hence the overall running time is $O(T_B + |\mathcal{L}| \cdot T_A)$. ◀

Now suppose we are given a list $\mathcal{L}$ of size $g(k)$ (for some function g) and a partition algorithm for the problem with FPT running time. Then by Theorem 4, we get an FPT algorithm for the constrained k-service problem. Since for many of the constrained k-service problems there exists efficient partition algorithms, it makes sense to design an algorithm for the list k-service problem that outputs a list of size at most $g(k)$. We design such an algorithm in this paper. We also need to make sure that the partition algorithms for constrained problems that we saw in Table 1 exist and our plan of approaching the constrained problem using the list problem can be executed. Indeed, Ding and Xu [24] gave partition algorithms for a number of constrained problems. We make addition to their list which allows us to discuss new problems in this work. These additions and other discussions on approaching specific constrained problems using the list problem is discussed in the full version of the paper. What we note here is that the approximation guarantee for the list problem carries over to all the constrained problem in Table 1. We now look at our main results for the list k-service problem and its main implications for the constrained problems.

1.1 Our Results

We will show the following result for the list k-service problem.

▶ **Theorem 5** (Main Theorem). *Let $0 < \varepsilon \leq 1$ and $\ell \geq 1$. Let (L, C, k, d, ℓ) be any k-service instance and let $\xi = \{C_1, C_2, \ldots, C_k\}$ be any arbitrary clustering of the client set. There is an algorithm that, with probability at least $1/2$, outputs a list $\mathcal{L}$ of size $(k/\varepsilon)^{O(k\,\ell^2)}$, such that there is a k-center-set $S \in \mathcal{L}$ in the list such that $\Psi(S, \xi) \leq (3^\ell + \varepsilon) \cdot \Psi^*(\xi)$. Moreover, the running time of the algorithm is $O\left(n \cdot (k/\varepsilon)^{O(k\,\ell^2)}\right)$. For the special case when $C \subseteq L$, the algorithm gives a $(2^\ell + \varepsilon)$-approximation guarantee.*

Using the above Theorem together with Theorem 4, we obtain the following main results for the constrained k-means and k-median problems.

▶ **Corollary 6** (k-means). *For any constrained version of the metric k-means problem with a partition algorithm with FPT running time $g(k) \cdot n^{O(1)}$, there is a $(9+\varepsilon)$-approximation algorithm with an FPT running time $g(k) \cdot (k/\varepsilon)^{O(k)} \cdot n^{O(1)}$. For a special case when $C \subseteq L$, the algorithm gives a $(4+\varepsilon)$-approximation guarantee.*

▶ **Corollary 7** (k-median). *For any constrained version of the metric k-median problem with a partition algorithm with FPT running time $g(k) \cdot n^{O(1)}$, there is a $(3+\varepsilon)$-approximation algorithm with an FPT running time of $g(k) \cdot (k/\varepsilon)^{O(k)} \cdot n^{O(1)}$. For a special case when $C \subseteq L$, the algorithm gives a $(2+\varepsilon)$-approximation guarantee.*

Note that by Theorem 4, as long as the running time of the partition algorithm is $g(k) \cdot n^{O(1)}$, the total running time of the algorithm still stays FPT. All the problems in Table 1 either have an efficient partition algorithm (polynomial in n and k) or a partition algorithm with an FPT running time. We discuss these partition algorithms in the full version of the paper. It should be noted that other than the problems mentioned in Table 1, our algorithm works for any problem that fits the framework of the constrained k-service problem (i.e., Definition 2) and has a partition algorithm. This makes the approach extremely versatile since one may be able to solve more problems that may arise in the future.[3] The known results on constrained problems in Table 1 is summarised in Table 2. Note that for **all** these problems we obtain FPT time $(9+\varepsilon)$-approximation and $(3+\varepsilon)$-approximation for k-means and k-median respectively. For the special case when $C \subseteq L$ (a facility can be opened at any client location), we obtain FPT time $(4+\varepsilon)$-approximation and $(2+\varepsilon)$-approximation for k-means and k-median respectively. There are some subtle differences in the problems in Table 1 and Table 2. This is to be able to compare our results with known results. We will highlight these differences in the related work section.

Moreover, we can convert our algorithms to streaming algorithms using the technique of Goyal et al. [28]. We basically require a streaming version of our algorithm for the list k-service problem and a streaming partition algorithm for the constrained k-service problem. In Section 1.5, we will design a constant-pass log-space streaming algorithm for the list k-service problem. We already know streaming partition algorithms for the various constrained k-service problems [28]. This would give a streaming algorithm for all the problems given in Table 1 except for the l-diversity and chromatic k-service problems. Although *single*-pass streaming algorithms are considered much useful, it is interesting to know that there is a constant-pass streaming algorithm for many constrained versions of the k-service problem.

1.2 Related Work

A unified framework for constrained k-means/k-median problems was introduced by Ding and Xu [24]. Using this framework, they designed a PTAS (fixed k) for various constrained clustering problems. However, their study was limited to the Euclidean space where $C \subseteq L = \mathbb{R}^d$. Their results were obtained through an algorithm for the list version of the k-means problem (even though it was not formally defined in their work). The running time of this algorithm was $O(nd \cdot (\log n)^k \cdot 2^{poly(k/\varepsilon)})$ and the list size was $(\log n)^k \cdot 2^{poly(k/\varepsilon)}$. Bhattacharya et al. [9] formally defined and studied the list k-service problem. They obtained a faster

[3] We note that new ways of modelling fairness in clustering is giving rise to new clustering problems with fairness constraints and some of these new problems may fit into this framework.

Table 2 Known results for the constrained clustering problems. Note that for **all** the above problems we obtain FPT time $(3+\varepsilon)$-approximation and $(9+\varepsilon)$-approximation for k-median and k-means respectively. For the special case when $C \subseteq L$ (a facility can be opened at any client location), we obtain FPT time $(2+\varepsilon)$-approximation and $(4+\varepsilon)$-approximation for k-median and k-means respectively. Note that uniform case for problems 1 and 2 means that the lower/upper bound on the size of all clusters is the same.

#	**Problem**	**Metric k-median**	**Metric k-means**
1.	r-gather k-service (uniform case)	7.2-approx [23] (for $C = L$) (in FPT time)	86.9-approx [23] (for $C = L$) (in FPT time)
2.	r-Capacity k-service (uniform case)	$(3+\varepsilon)$-approx [19] (in FPT time)	$(9+\varepsilon)$-approx [19] (in FPT time)
3.	l-Diversity k-service	-	-
4.	Chromatic k-service	-	-
5.	Fault tolerant k-service	93-approx. [31]	-
6.	OWA k-service	93-approx. [12]	-
7.	Semi-supervised k-service	-	-
8.	Uncertain k-service (assigned version)	$(6.35+\varepsilon)$-approx. [20] (for $C \subseteq L$)	$(74+\varepsilon)$-approx. [20] (for $C \subseteq L$)
9.	Outlier k-service	$(7+\varepsilon)$-approx. [37]	$(53+\varepsilon)$-approx. [37]

For the Euclidean k-means and k-median (where $C \subseteq L = \mathbb{R}^d$), all the constrained problems have an FPT time $(1+\varepsilon)$ approximation algorithm [24, 9].

algorithm for the list problem with running time to $O(nd \cdot (k/\varepsilon)^{O(\log(k/\varepsilon))})$ and list size to $(k/\varepsilon)^{O(\log(k/\varepsilon))}$ for the constrained k-means/k-median problem. Recently, Goyal et al. [28] obtained useful generalisations of the results of Bhattacharya et al. [9] and used this to design logspace (assuming k and ε are constants) streaming algorithms for various constrained versions of the problem. In this paper, we study the constrained k-means/median problems in general metric spaces while treating L and C as separate sets. More importantly, we design an algorithm that gives a better approximation guarantee than the previously known algorithms by taking advantage of FPT running time. Moreover, for many problems, it is the first algorithm that achieves a constant-approximation in FPT running time. Please see Table 2 for the known results on the problem. Due to space restrictions, we have a detailed discussion on these problems in the full version of the paper.

In the introduction, we would specifically like to discuss the result of Addad et al. [19] for the capacitated k-service problem. Their definition of the capacitated k-service problem is different from the one mentioned in Table 1 that we are considering. Following is their definition of the capacitated k-service problem.

▶ **Definition 8** (Addad et al. [19]). *Given an instance $\mathcal{I} = (L, C, k, d, \ell)$ of the k-service problem and a capacity function $r : L \to \mathbb{Z}_+$, find a set $F \subseteq L$ of k facilities such that the assignment cost $\sum_{j \in C} \min_{i \in F} d^\ell(j, i)$ is minimized, and no more than r_i clients are assigned to a facility $i \in L$.*

Note that in the above definition, there is a capacity associated with every facility location in L whereas in our definition, capacities are associated with the k clusters. This means that a facility can service arbitrary number of clients as long as the cluster sizes are bounded. This is not allowed as per the problem definition of Addad et al. [19]. However, for the uniform capacities the problem definitions are equivalent and the results become comparable. We match the approximation guarantees obtained Addad et al. [19] for the uniform case even though using very different techniques.

As we mentioned earlier, the unconstrained metric k-median problem is hard to approximate within a factor strictly smaller than $(1+2/e)$, and the metric k-means problem is hard to approximate within a factor strictly smaller than $(1+8/e)$. Surprisingly this lower bound persists even if we allow an FPT running time [17, 42]. However, this FPT lower bound is based on a complexity theoretic conjecture known as Gap-ETH [13]. The problem also has a matching upper bound algorithm with an FPT running time [17]. So, the unconstrained k-means and k-median problems in the metric setting is fairly well understood. On the other hand, our understanding of most constrained versions of the problem is still far from complete. We believe that our work is an important step in understanding constrained problems in general metric spaces.

1.3 Our Techniques

In this section, we discuss our sampling based algorithm for list k-service problem. First, let us define a few notations and identities that we will use often in our discussions. We define the unconstrained k-service cost of a set S with respect to a center set F as: $\Phi(F,S) := \sum_{x\in S} \min_{f\in F} d^\ell(f,x)$. For a singleton set $\{f\}$, we denote $\Phi(\{f\},S)$ by $\Phi(f,S)$. We denote the optimal (unconstrained) k-service cost of an instance (L,C,k,d,ℓ) by $OPT(L,C)$.

As described earlier, an FPT algorithm for the list k-service problem gives an FPT algorithm for a constrained version of the k-service problem that has an efficient or FPT-time partition algorithm. Given that we are allowed FPT running time for the list problem, it may be tempting to think of the following strategy: Use a bi-criteria approximation algorithm for the unconstrained version of the k-median problem to obtain $poly(k/\varepsilon)$ centers S and then use the partition algorithm on all k-sized subsets of S to pick the best one. Unfortunately, this strategy does not give a constant factor approximation. We discuss the details in the full version of the paper.

In this work, we give a sampling based algorithm that is similar to the algorithm of Goyal et al. [28] that was specifically designed for the Euclidean setting. However, working in a metric space instead of Euclidean space poses challenges as some of the main tools used for analysis in the Euclidean setting cannot be used in metric spaces. We carefully devise and prove new sampling lemmas that makes the high-level analysis of Goyal et al. [28] go through. Our algorithm is based on D^ℓ-sampling. Given a point set F, D^ℓ-sampling a point from the client set C w.r.t. center set F means sampling using the distribution where the sampling probability of a client $x \in C$ is $\frac{\Phi(F,\{x\})}{\Phi(F,C)} = \frac{\min_{f\in F} d^\ell(f,x)}{\sum_{y\in C}\min_{f\in F} d^\ell(f,y)}$. In case F is empty, then D^ℓ-sampling is the same as uniform sampling. Please see Algorithm 1 for the list k-service problem.

Let us discuss some of the main ideas of the algorithm and its analysis. First, note that as per the algorithm description, the list size is $2^k \cdot \binom{(\eta+1)k^2}{k}$ which is $(k/\varepsilon)^{O(k\ell^2)}$ for the parameters given. This is because in step (9), the algorithm considers all possible k sized subsets of (multi)set T of size $(\eta+1)k^2$. We now discuss the approximation guarantee. Note that in the first step, we obtain a center-set $F \subseteq C$ which is an α-approximation for the

Algorithm 1 Algorithm for the list k-service problem.

```
List-k-service (L, C, k, d, ℓ, ε)
    Inputs: k-service instance (L, C, k, d, ℓ) and accuracy ε
    Output: A list 𝓛, each element in 𝓛 being a k-center set
    Constants: β = 4^{ℓ−1} · ( ℓ^ℓ · 3^{ℓ²+4ℓ+3} / ε^{ℓ+1} + 1 ); γ = ℓ^ℓ · 3^{ℓ²+5ℓ+1} / ε^ℓ;
    η = α β γ k · 3^{ℓ+2} / ε²
(1)  Run any α-approximation algorithm with α = poly(k) for the unconstrained
     k-service instance (C, C, k, d, ℓ) and let F be the obtained center-set.
       (k-means++ [6] is one such algorithm.)
(2)  𝓛 ← ∅
(3)  Repeat 2^k times:
(4)      Sample a multi-set M of ηk points from C using D^ℓ-sampling w.r.t.
          center set F
(5)      M ← M ∪ F
(6)      T ← ∅
(7)      For every point x in M:
(8)          T ← T ∪ {k points in L that are closest to x}
(9)      For all subsets S of T of size k:
(10)         𝓛 ← 𝓛 ∪ {S}
(11) return(𝓛)
```

unconstrained k-service instance (C, C, k, d, ℓ). Any α that is polynomial in k suffices for our analysis. That is, $\Phi(F, C) \leq \alpha \cdot OPT(C, C)$. One such algorithm is the k-means++ algorithm [6] that gives an $O(4^\ell \cdot \log k)$-approximation guarantee and a running time $O(nk)$. Now, let us see how the center-set F can help us. Let us focus on any cluster C_i of a target clustering $\xi = \{C_1, \ldots, C_k\}$. We note that the closest facility to a uniformly sampled client from any client set C_i provides a constant approximation to the optimal 1-median/means cost for C_i in expectation. This is formalized in the next lemma. This lemma (or a similar version) has been used in multiple other works in analysing sampling based algorithms (for example, see Lemma 3.1 in [6]). This lemma is restated and formally proven in the full version of the paper.

▶ **Lemma 9.** *Let $S \subseteq C$ be any subset of clients and let f^* be any center in L. If we uniformly sample a point x in S and open a facility at the closest location in L, then the following identity holds:*

$$\mathbb{E}[\Phi(t(x), S)] \leq 3^\ell \cdot \Phi(f^*, S), \text{ where } t(x) \text{ is the closest facility location from } x.$$

Unfortunately, we cannot uniformly sample from C_i directly since C_i is not known to us. Given this, our main objective should be to use F to try to uniformly sample from C_i so that we could achieve a constant approximation for C_i. Let us do a case analysis based on the distance of points in C_i from the nearest point in F. Consider the following two possibilities: The first possibility is that the points in C_i are close to F. If this is the case, we can uniformly sample a point from F instead of C_i. This would incur some extra cost. However, the cost is small and can be bounded. To cover this first possibility, the algorithm adds the entire set F to the set of sampled points M (see line (5) of the algorithm). The second possibility

is that the points in C_i are far-away from F. In this case, we can D^ℓ-sample the points from C. Since the points in C_i are far away, the sampled set would contain a good portion of points from C_i and the points will be *almost* uniformly distributed. We will show that almost uniform sampling is sufficient to apply Lemma 9 on C_i. However, we would have to sample a large number of points to boost the success probability. This requirement is taken care of by line (4) of the algorithm. Note that we may need to use a hybrid approach for analysis since the real case may be a combination of the first and second possibility. Most of the ingenuity of this work lies in formulating and proving appropriate sampling lemmas to make this hybrid analysis work.

To apply Lemma 9, we need to fulfill one more condition. We need the closest facility location from a sampled point. This requirement is handled by lines (7) and (8) of the algorithm. However, note that the algorithm picks k-closest facility locations instead of just one facility location. We will show that this step is crucial to obtain a *hard-assignment* solution for the problem. Finally, the algorithm adds all the potential center sets to a list $\mathcal{L}$ (see line (9) and (10) of the algorithm). The algorithm repeats this procedure 2^k times to boost the success probability (see line (3) of the algorithm). We will show the following result from which our main theorem (Theorem 5) trivially follows.

▶ **Theorem 10.** *Let $0 < \varepsilon \leq 1$ and $\ell \geq 1$. Let (L, C, k, d, ℓ) be any k-service instance and let $\xi = \{C_1, C_2, \ldots, C_k\}$ be any arbitrary clustering of the client set. The algorithm* `List-k-service`*$(L, C, k, d, \ell, \varepsilon)$, with probability at least $1/2$, outputs a list $\mathcal{L}$ of size $(k/\varepsilon)^{O(k\,\ell^2)}$, such that there is a k center set $S \in \mathcal{L}$ in the list such that $\Psi(S, \xi) \leq (3^\ell + \varepsilon) \cdot \Psi^*(\xi)$. Moreover, the running time of the algorithm is $O\left(n \cdot (k/\varepsilon)^{O(k\,\ell^2)}\right)$. For the special case of $C \subseteq L$, the approximation guarantee is $(2^\ell + \varepsilon)$.*

The details of the analysis is given in the full version of the paper. Here, we give the high-level outline of the proof. Let $\xi = \{C_1, C_2, \ldots, C_k\}$ be the (unknown) target clustering and $F^* = \{f_1^*, f_2^*, \ldots, f_k^*\}$ be the corresponding optimal center set. Suppose C_i is assigned to f_i^*, and $\Delta(C_i)$ denote its corresponding cost, i.e., $\Delta(C_i) = \Phi(f_i^*, C_i)$. Let us classify the clusters into two categories: W and H.

$$W \coloneqq \{C_i \mid \Phi(F, C_i) \leq \frac{\varepsilon}{\alpha\,\gamma\,k} \cdot \Phi(F, C),\ \text{for } 1 \leq i \leq k\}$$

$$H \coloneqq \{C_i \mid \Phi(F, C_i) > \frac{\varepsilon}{\alpha\,\gamma\,k} \cdot \Phi(F, C),\ \text{for } 1 \leq i \leq k\}$$

In other words, W contains the *low-cost* clusters and H contains the *high-cost* clusters with respect to F. Now, let us look at the set M obtained by lines (4) and (5) of the algorithm. M contains some D^ℓ-sampled points from C and the center set F. We show that M has the following property.

Property-I: For any cluster $C_i \in \{C_1, C_2, \ldots, C_k\}$, with probability at least $1/2$, there is a point s_i in M such that the following holds:

$$\Phi(t(s_i), C_i) \leq \begin{cases} \left(3^\ell + \frac{\varepsilon}{2}\right) \cdot \Delta(C_i) + \frac{\varepsilon}{2^{\ell+1}\,k} \cdot OPT(C, C), & \text{if } C_i \in W \\ \left(3^\ell + \frac{\varepsilon}{2}\right) \cdot \Delta(C_i), & \text{if } C_i \in H \end{cases}$$

where $t(s_i)$ denotes any facility location that is closer to s_i than f_i^*, i.e., $d(s_i, t(s_i)) \leq d(s_i, f_i^*)$.

First, let us see how this property gives the desired result. By a well known fact, we have $OPT(C, C) \leq 2^{\ell} \cdot OPT(L, C)$. Moreover, the optimal cost $OPT(L, C)$ of the unconstrained k-service instance is always less than the constrained k-service cost $\sum_{i=1}^{k} \Delta(C_i)$. Therefore, Property-I implies that $T_s \coloneqq \{t(s_1), t(s_2), \ldots, t(s_k)\}$ is a $\left(3^{\ell} + \varepsilon\right)$-approximation for ξ, with probability at least $\frac{1}{2^k}$. [4] Now, note that the facility locations that are closest to s_i satisfy the definition of $t(s_i)$. Moreover, the algorithm adds one such facility location to set T (see line (8) of the algorithm). Thus there is a center-set T_s in the list that gives $(3^{\ell}+\varepsilon)$-approximation for ξ. To boost the success probability to $1/2$, the algorithm repeats the procedure 2^k times (see line (3) of the algorithm). Based on these arguments, it looks like we got the desired result. However, there is one issue that we need to take care of. Remember, we are looking for a hard assignment for the problem, and the set T_s could be a soft center-set, since the closest facility locations might be same for s_i's. In other words, $t(s_i)$ could be same as $t(s_j)$ for some $i \neq j$. At the end of this section we will show that there is indeed a hard center-set in the list $\mathcal{L}$, that gives the required approximation for the problem. For now let us try to argue Property-I for M and the target clusters. First consider the case of low-cost clusters as follows.

Case 1: $\boldsymbol{\Phi(F, C_i) \leq \frac{\varepsilon}{\alpha \gamma k} \cdot \Phi(F, C)}$

For a point $x \in \mathcal{X}$, let $c(x)$ denote the closest location in F. Based on this definition, consider a multi-set $M_i \coloneqq \{c(x) \mid x \in C_i\}$. Since C_i has a low cost with respect to F, the points in C_i are close to from points from F. Consider uniformly sampling a point from M_i. In the next lemma, we show that a uniformly sampled point from M_i is a good enough center for C_i. We give the proof in the full version of the paper.

▶ **Lemma 11.** *Let p be a point sampled uniformly at random from M_i. Then the following bound holds:*

$$\mathbb{E}[\Phi(t(p), C_i)] \leq \left(3^{\ell} + \frac{\varepsilon}{2}\right) \cdot \Delta(C_i) + \frac{\varepsilon}{2^{\ell+1} k} \cdot OPT(C, C).$$

Since the above lemma estimates the average cost corresponding to a sampled point, there has to be a point p in M_i such that $\Phi(t(p), C_i) \leq \left(3^{\ell} + \frac{\varepsilon}{2}\right) \cdot \Delta(C_i) + \frac{\varepsilon}{2^{\ell+1} k} \cdot OPT(C, C)$. Since M_i is only composed of the points from F and we keep the entire set F in M (see line (5) of the algorithm), therefore Property-I is satisfied for every cluster $C_i \in W$. Let us now prove Property I for the high cost clusters.

Case 2: $\boldsymbol{\Phi(F, C_i) > \frac{\varepsilon}{\alpha \gamma k} \cdot \Phi(F, C)}$

Since the cost of the cluster is high, some points of C_i are far away from the center set F. We partition C_i into two sets: C_i^n and C_i^f, as follows.

$$C_i^n \coloneqq \{x \mid d^{\ell}(c(x), x) \leq R^{\ell}, \text{ for } x \in C_i\}, \quad \text{where } R^{\ell} = \frac{1}{\beta} \cdot \frac{\Phi(F, C_i)}{|C_i|}$$

$$C_i^f \coloneqq \{x \mid d^{\ell}(c(x), x) > R^{\ell}, \text{ for } x \in C_i\}, \quad \text{where } R^{\ell} = \frac{1}{\beta} \cdot \frac{\Phi(F, C_i)}{|C_i|}$$

[4] Note that the probabilities can be multiplied since M can be partitioned into k groups and we actually show that the good point s_i for C_i is either in F or is in any group with probability at least $1/2$.

In other words, C_i^n represents the set of points that are *near* to the center set F and C_i^f represents the set of points that are *far* from the center set F. Recall that our prime objective is to obtain a uniform sample from C_i, so that we can apply lemma 9. To achieve that we consider sampling from C_i^n and C_i^f separately. The idea is as follow. To sample a point from C_i^f we use the D^ℓ-sampling technique and show that it gives an almost uniform sample from C_i^f. For C_i^n, we will use F as its proxy, and sample a point from F instead. However, doing so would incur an extra cost. Since we are using F as a proxy for C_i^n, we define a multi-set $M_i^n \coloneqq \{c(x) \mid x \in C_i^n\}$. Let us define another multi-set $M_i \coloneqq C_i^f \cup M_i^n$. In the following lemma we show that there is a point in M_i that is a good center for C_i. The lemma is similar to lemma 11 of the low-cost clusters. The formal proof is given in the full version of the paper.

▶ **Lemma 12.** *Let p be a point sampled uniformly at random from M_i. Then the following bound holds:*

$$\mathbb{E}[\Phi(t(p), C_i)] \leq \left(3^\ell + \frac{\varepsilon}{4}\right) \cdot \Delta(C_i)$$

The above lemma gives a bound on the expectation. To show that M has a good center for high-cost cluster C_i with high probability, we need to make sure that adequate samples are obtained in line (4) of the algorithm. The choice of parameters β, γ, and η is based on this probability analysis and is given in the full version of the paper.

Having argued that Property-I is satisfied for every cluster in W and H, we can finally claim that $T_s = \{t(s_1), t(s_2), \ldots, t(s_k)\}$ is a $\left(3^\ell + \varepsilon\right)$-approximation for ξ with probability at least $\frac{1}{2^k}$. However, as described earlier, T_s could be a soft center-set since $t(s_i)$ can be same as $t(s_j)$ for some $i \neq j$. To obtain a hard center-set, we make use of line (8) of the algorithm. In line (8), the algorithm pulls out the k closest points from L instead of just one. Note that it is not necessary to open a facility at a closest location in L. Rather, we can open a facility at any location f in L, that is at least as close to s_i as f_i^*, i.e., $d(s_i, f) \leq d(s_i, f_i^*)$.

Let $T(s_i)$ denote a set of k closest facility location for s_i. We show that there is a hard center-set $T_h \subset \cup_i T(s_i)$, such that $T_h \coloneqq \{f_1, \ldots, f_k\}$ and $d(s_i, f_i) \leq d(s_i, f_i^*)$ for every $1 \leq i \leq k$. We define T_h using the following simple subroutine:

```
1 FindFacilities
2   - T_h ← ∅
3   - For i ∈ {1, ..., k}:
4       - if (f_i^* ∈ T(s_i)) T_h ← T_h ∪ {f_i^*}
5       - else
6           - Let f ∈ T(s_i) be any facility such that f is not in T_h
7           - T_h ← T_h ∪ {f}
```

▶ **Lemma 13.** *$T_h = \{f_1, f_2, \ldots, f_k\}$ contains exactly k different facilities such that for every $1 \leq i \leq k$, we have $d(s_i, f_i) \leq d(s_i, f_i^*)$.*

Proof. First, let us show that all facilities in T_s are different. Since, f_i^* is different for different clusters, the if statement adds facilities in T_h that are different. In else part, we only add a facility to T_h that is not present in T_h. Thus the else statement also adds facilities in T_h that are different. Now, let us prove the second property, i.e., $d(s_i, f_i) \leq d(s_i, f_i^*)$ for every $1 \leq i \leq k$. The property is trivially true for the facilities added in the if statement. Now, for the facilities added in the second step we know that $T(s_i)$ does not contain f_i^*. Since, $T(s_i)$

is a set of k-closest facility locations, we can say that for any facility location f in $T(s_i)$, $d(s_i, f) \leq d(s_i, f_i^*)$. Thus any facility added in the else statement has $d(s_i, f) \leq d(s_i, f_i^*)$. This completes the proof. ◀

Thus $T_h \in \mathcal{L}$ is a hard center-set, which gives the $(3^\ell + \varepsilon)$-approximation for the problem. This completes the analysis of the algorithm.

Now, suppose we are given the flexibility to open a facility at a client location. In other words, suppose it is given that $C \subseteq L$. For this case, we can directly open the facilities at the locations $\{s_1, s_2, \ldots, s_k\}$ instead of $t(s_i)$'s, and we would not need lines (7) and (8) of the algorithm. Further, we can show that lemma 11 and 12 would give $(2^\ell + \varepsilon)$-approximation for this special case. However, please note that $\{s_1, s_2, \ldots, s_k\}$ is still a soft center-set. To obtain a hard center-set we do need to consider the k-closest facility locations for a point s_i. In that case, lemma 11 and 12 would not provide $(2^\ell + \varepsilon)$-approximation. Therefore, we need to make some changes in the analysis of lemma 11 and 12 to get back $(2^\ell + \varepsilon)$-approximation. We discuss these details in the full version of the paper.

1.4 A Matching Lower Bound on approximation

We gave sampling based algorithms and showed an approximation guarantee of $(3^\ell + \varepsilon)$ (and $(2^\ell + \varepsilon)$ for the special case $C \subseteq L$). In this subsection, we show that our analysis of the approximation factor is tight. More specifically, we will show that our algorithm does not provide better than $(3^\ell - \delta')$ approximation guarantee for arbitrarily small $\delta' > 0$ (and $2^\ell - \delta'$ for the case $C \subseteq L$). To show this, we create a *bad instance* for the problem in the following manner. We create the instance using an undirected weighted graph where $C \cup L$ is the vertex set of the graph and the shortest weighted path between two vertices defines the distance metric. The set C is partitioned into the subsets $C_1, C_2, \ldots, C_k$, and L is partitioned into the subsets $L_1, L_2, \ldots, L_k$. The subgraphs over $C_1 \cup L_1, C_2 \cup L_2, \ldots,$ and $C_k \cup L_k$ are all identical to each other. Let us describe the subgraph over vertex set $C_i \cup L_i$ in general. In this subgraph, all the clients are connected to a common facility location f_i^* with an edge of unit weight. Also, every client is connected to a distinct set of k facility locations with an edge of weight $(1 - \delta)$. We denote this set by $T(x)$ for a client $x \in C_i$. Figure 1 shows the complete description of this subgraph. Lastly, all pairs of subgraphs $C_i \cup L_i$ and $C_j \cup L_j$ are connected with an edge (f_i^*, f_j^*) of weight $\Delta \gg |C|$. This completes the construction of the bad instance.

Let us define a target clustering on the instance. Consider the unconstrained k-service problem. It is easy to see that $\xi = \{C_1, C_2, \ldots, C_k\}$ is an optimal clustering for this instance. The optimal cost of a cluster C_i is $\Phi(f_i^*, C_i) = |C_i|$, and the optimal cost of the entire instance is $OPT = \sum_i |C_i| = |C|$.

Now, we will show that any list $\mathcal{L}$ produced by the algorithm `List-k-service` does not contain any center-set that can provide better than $(3^\ell - \delta')$-approximation for ξ. To show this, let us examine every center-set in the list $\mathcal{L}$ produced by `List-k-service`. Note that the set T obtained in line (8) of the algorithm does not contain any optimal facility location f_i^* because f_i^* does not belong to $T(x)$. Therefore, no center set in the list contains any of the optimal facility locations $\{f_1^*, ..., f_k^*\}$. Let us evaluate the clustering cost corresponding to every center set in the list. Let $F = \{f_1, f_2, \ldots, f_k\}$ be a center-set in the list. We have two possibilities for the facilities in F. The first possibility is that, there are at least two facilities in F, that belongs to the same subgraph $C_i \cup L_i$. In this case, the cost of the target clustering is $\Psi(F, \xi) > \Delta \gg OPT$. So in this case, F gives an unbounded clustering cost. Let us consider the second possibility that all facilities in F belong to different subgraphs.

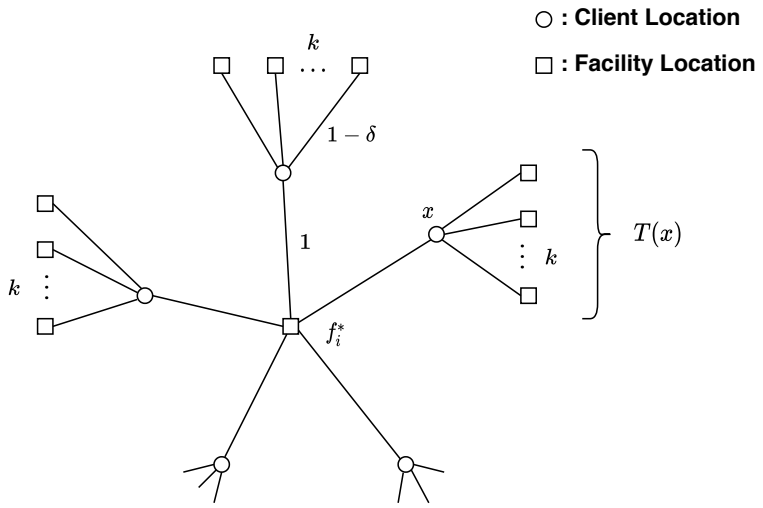


Figure 1 An undirected weighted subgraph on $C_i \cup L_i$.

Without loss of generality, we can assume that $f_i \in L_i$. Since f_i can not be the optimal facility location, we can further assume that $f_i \in T(x)$ for some $x \in C_i$. The cost of a cluster in this case is $\Phi(f_i, C_i) = (3-\delta)^\ell(|C_i|-1) + (1-\delta)^\ell > (3-\delta)^\ell(|C_i|-1)$. Hence, the overall cost of the instance is $\Psi(F,\xi) > (3-\delta)^\ell \cdot (|C|-k) \geq (3-\delta)^\ell \cdot |C| - 3^\ell\, k \geq (3^\ell - \delta') \cdot |C|$, for $\delta' = 3^{\ell-1} \cdot \ell\, \delta + \frac{3^\ell k}{|C|}$. Therefore, we can say that list does not contain any center set that can provide better than $(3^\ell - \delta')$ approximation guarantee for ξ.

▶ **Theorem 14.** *For any $0 < \delta' \leq 1$, there are instances of the k-service problem for which the algorithm* `List-k-service`$(L, C, k, d, \ell, \varepsilon)$ *does not provide better than $(3^\ell - \delta')$ approximation guarantee.*

Now, let us examine the same bad instance when we have the flexibility to open a facility at a client location. In this case, we have a third possibility that $F = \{f_1, f_2, \ldots, f_k\}$ such that f_i is some client location in C_i. The cost of a cluster in this case is $\Phi(f_i, C_i) = 2^\ell \cdot (|C_i| - 1)$ and the overall cost the instance is $\Psi(F,\xi) = 2^\ell \cdot |C| - 2^\ell \cdot k = (2^\ell - \delta') \cdot |C|$, for $\delta' = 2^\ell \cdot k/|C|$. So for the special case $C \subseteq L$, we obtain the following theorem.

▶ **Theorem 15.** *For any $0 < \delta' \leq 1$, there are instances of the k-service problem (with $C \subseteq L$), for which the algorithm* `List-k-service`$(L, C, k, d, \ell, \varepsilon)$ *does not provide better than $(2^\ell - \delta')$ approximation guarantee.*

1.5 Streaming Algorithms

In this subsection, we discuss how to obtain a constant-pass streaming algorithm using the ideas of Goyal et al. [28]. Our offline algorithm has two main components, namely: the list k-service algorithm and partition algorithm. The list k-service procedure is common to all constrained versions of the problem. However, the partition algorithm differs for different constrained versions. First, let us convert `List-k-service`(L, C, k, d, ℓ) algorithm to a streaming algorithm.

Algorithm Streaming algorithm.

1. In the first pass, we run a streaming α-approximation algorithm for the instance (C, C, k, d, ℓ). For this, we can use the streaming algorithm of Braverman et al. [10]. The algorithm gives a constant-approximation with the space complexity of $O(k \log n)$.
2. In the second pass, we perform the D^ℓ-sampling step using the *reservoir sampling* technique [46].
3. In the third pass, we find the k-closest facility locations for every point in M.

This gives us the following result.

▶ **Theorem 16.** *There is a 3-pass streaming algorithm for the list k-service problem, with the running time of $O(n \cdot f(k, \varepsilon))$ and space complexity of $f(k, \varepsilon) \cdot \log n$, where $f(k, \varepsilon) = (k/\varepsilon)^{O(k\ell^2)}$.*

Now, let us discuss the partition algorithms in streaming setting. For the l-diversity and chromatic k-service problems, it is known that there is no deterministic log-space streaming algorithm [28]. For the remaining constrained problems, there are streaming partition algorithms that are discussed in [28] (for the Euclidean setting) and the full version of the paper. Note that all of the streaming partitioning algorithms do not give an optimal partitioning but only a partitioning that is close to the optimal. Each algorithm makes at most 3-pass over the data-set and takes logarithmic space complexity. The partition algorithm, together with the list k-service algorithm, gives the following main results.

▶ **Theorem 17.** *For the following constrained k-service problems there is a 6-pass streaming algorithm that gives a $(3^\ell + \varepsilon)$-approximation guarantee: (1) r-gather k-service problem, (2) r-capacity k-service problem, (3) Fault-tolerant k-service problem, (4) Semi-supervised k-service problem, (5) Uncertain k-service problem (assigned case). The algorithm has the space complexity of $O(f(k, \varepsilon, \ell) \cdot \log n)$ and the running time of $O(f(k, \varepsilon, \ell) \cdot n^{O(1)})$, where $f(k, \varepsilon, \ell) = (k/\varepsilon)^{O(k\ell^2)}$. Further, the algorithm gives $(2^\ell + \varepsilon)$-approximation guarantee when $C \subseteq L$.*

▶ **Theorem 18.** *For the outlier k-service problem there is a 5-pass streaming algorithm that gives a $(3^\ell + \varepsilon)$-approximation guarantee. The algorithm has space complexity of $O(f(k, m, \varepsilon, \ell) \cdot \log n)$ and running time of $f(k, m, \varepsilon, \ell) \cdot n^{O(1)}$, where $f(k, m, \varepsilon, \ell) = ((k + m)/\varepsilon)^{O(k\ell^2)}$. Further, the algorithm gives $(2^\ell + \varepsilon)$-approximation guarantee when $C \subseteq L$.*

2 Conclusion and Open Problems

In this paper, we worked within the unified framework of Ding and Xu [24] to obtain simple sampling based algorithms for a range of constrained k-median/means problems in general metric spaces. Surprisingly, even working within this high-level framework, we obtained better (or matched) approximation guarantees of known results that were designed specifically for the constrained problem. On one hand, this shows the versatility of the unified approach along with the sampling method. On the other hand, it encourages us to try to design algorithms with better approximation guarantees for these constrained problems. Our matching approximation lower bound for the sampling algorithm suggests that further improvement may not be possible through sampling based ideas. On the lower bound side, it may be useful to obtain results similar to that for the unconstrained setting where approximation lower bounds of $(1 + 2/e)$ and $(1 + 8/e)$ are known for k-median and k-means respectively [17]. Another direction is to find other constrained problems that can fit into the unified framework and can benefit from the results in this work.

References

1 Marek Adamczyk, Jaroslaw Byrka, Jan Marcinkowski, Syed M. Meesum, and Michal Wlodarczyk. Constant-factor FPT approximation for capacitated k-median. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2019.1.

2 Gagan Aggarwal, Rina Panigrahy, Tomás Feder, Dilys Thomas, Krishnaram Kenthapadi, Samir Khuller, and An Zhu. Achieving anonymity via clustering. *ACM Trans. Algorithms*, 6(3), July 2010. doi:10.1145/1798596.1798602.

3 S. Ahmadian, A. Norouzi-Fard, O. Svensson, and J. Ward. Better guarantees for k-means and Euclidean k-median by primal-dual algorithms. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS 2017)*, pages 61–72, October 2017. doi:10.1109/FOCS.2017.15.

4 Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of Euclidean sum-of-squares clustering. *Mach. Learn.*, 75(2):245–248, May 2009. doi:10.1007/s10994-009-5103-0.

5 Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. *SIGMOD Rec.*, 28(2):49–60, June 1999. doi:10.1145/304181.304187.

6 David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2007, pages 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics.

7 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004. doi:10.1137/S0097539702416402.

8 Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. The Hardness of Approximation of Euclidean k-Means. In Lars Arge and János Pach, editors, *31st International Symposium on Computational Geometry (SoCG 2015)*, volume 34 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 754–767, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.SOCG.2015.754.

9 Anup Bhattacharya, Ragesh Jaiswal, and Amit Kumar. Faster algorithms for the constrained k-means problem. *Theor. Comp. Sys.*, 62(1):93–115, January 2018. doi:10.1007/s00224-017-9820-7.

10 Vladimir Braverman, Adam Meyerson, Rafail Ostrovsky, Alan Roytman, Michael Shindler, and Brian Tagiku. Streaming k-means on well-clusterable data. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2011, page 26–40, USA, 2011. Society for Industrial and Applied Mathematics.

11 Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Trans. Algorithms*, 13(2), March 2017. doi:10.1145/2981561.

12 Jaroslaw Byrka, Piotr Skowron, and Krzysztof Sornat. Proportional Approval Voting, Harmonic k-median, and Negative Association. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2018.26.

13 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-eth to fpt-inapproximability: Clique, dominating set, and more. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS 2017)*, pages 743–754, 2017.

14 Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002. doi:10.1006/jcss.2002.1882.

15 Ke Chen. On coresets for k-median and k-means clustering in metric and Euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009. doi:10.1137/070699007.

16 Vincent Cohen-Addad and Karthik C.S. Inapproximability of clustering in lp metrics. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS 2019)*, pages 519–539, 2019.

17 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT Approximations for k-Median and k-Means. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.42.

18 Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in Euclidean and minor-free metrics. *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS 2016)*, 00:353–364, 2016. doi:doi.ieeecomputersociety.org/10.1109/FOCS.2016.46.

19 Vincent Cohen-Addad and Jason Li. On the Fixed-Parameter Tractability of Capacitated Clustering. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 41:1–41:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.41.

20 Graham Cormode and Andrew McGregor. Approximation algorithms for clustering uncertain data. In *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '08, page 191–200, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1376916.1376944.

21 Marek Cygan, MohammadTaghi Hajiaghayi, and Samir Khuller. LP rounding for k-centers with non-uniform hard capacities. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 273–282, 2012.

22 Sanjoy Dasgupta. The hardness of k-means clustering. Technical Report CS2008-0916, Department of Computer Science and Engineering, University of California San Diego, 2008.

23 Hu Ding. Faster balanced clusterings in high dimension. *Theoretical Computer Science*, 2020. doi:10.1016/j.tcs.2020.07.022.

24 Hu Ding and Jinhui Xu. A unified framework for clustering constrained data without locality property. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2015, page 1471–1490, USA, 2015. Society for Industrial and Applied Mathematics.

25 Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD 1996, page 226–231. AAAI Press, 1996.

26 Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k-means clustering based on weak coresets. In *Proceedings of the twenty-third annual symposium on Computational geometry*, SCG 2007, pages 11–18, New York, NY, USA, 2007. ACM. doi:10.1145/1247069.1247072.

27 Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Local search yields a PTAS for k-means in doubling metrics. *SIAM Journal on Computing*, 48(2):452–480, 2019. doi:10.1137/17M1127181.

28 Dishant Goyal, Ragesh Jaiswal, and Amit Kumar. Streaming PTAS for constrained k-means, 2019. arXiv:1909.07511.

29 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31(1):228–248, 1999. doi:10.1006/jagm.1998.0993.

30 Anupam Gupta and Kanat Tangwongsan. Simpler analyses of local search algorithms for facility location. *CoRR*, abs/0809.2554, 2008. arXiv:0809.2554.

31 Mohammadtaghi Hajiaghayi, Wei Hu, Jian Li, Shi Li, and Barna Saha. A constant factor approximation algorithm for fault-tolerant k-median. *ACM Trans. Algorithms*, 12(3), April 2016. doi:10.1145/2854153.

32 Erez Hartuv and Ron Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(4):175–181, 2000. doi:10.1016/S0020-0190(00)00142-3.

33 Anil K. Jain, M Narasimha Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, September 1999. doi:10.1145/331499.331504.

34 Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC 2002, pages 731–740, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/509907.510012.

35 Ragesh Jaiswal, Amit Kumar, and Sandeep Sen. A simple D^2-sampling based PTAS for k-means and other clustering problems. *Algorithmica*, 70(1):22–46, 2014. doi:10.1007/s00453-013-9833-9.

36 Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of the Eighteenth Annual Symposium on Computational Geometry*, SCG 2002, page 10–18, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/513400.513402.

37 Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. Constant approximation for k-median and k-means with outliers via iterative rounding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, page 646–659, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188882.

38 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2):5:1–5:32, February 2010. doi:10.1145/1667053.1667054.

39 Shi Li. Approximating capacitated k-median with $(1+\varepsilon)k$ open facilities. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, page 786–796, USA, 2016. Society for Industrial and Applied Mathematics.

40 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC 2013, page 901–910, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2488608.2488723.

41 Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is NP-hard. *Theoretical Computer Science*, 442:13–21, 2012. Special Issue on the Workshop on Algorithms and Computation (WALCOM 2009). doi:10.1016/j.tcs.2010.05.034.

42 Pasin Manurangsi. Tight running time lower bounds for strong inapproximability of maximum k-coverage, unique set cover and related problems (via t-wise agreement testing theorem). In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2020, page 62?81, USA, 2020. Society for Industrial and Applied Mathematics.

43 Pitu B. Mirchandani and Richard L. Francis. *Discrete Location Theory*. Wiley, 1990.

44 Latanya Sweeney. k-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, October 2002. doi:10.1142/S0218488502001648.

45 Andrea Vattani. The hardness of k-means clustering in the plane. Technical report, Department of Computer Science and Engineering, University of California San Diego, 2009.

46 J S Vitter. Random sampling with a reservoir. *ACM Trans. Math. Software*, 11(1):37–57, 1985.

47 Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2, August 2015. doi:10.1007/s40745-015-0040-1.

Fixed-Parameter Algorithms for Graph Constraint Logic

Tatsuhiko Hatanaka
Graduate School of Information Sciences, Tohoku University, Sendai, Japan
hatanaka@ecei.tohoku.ac.jp

Felix Hommelsheim
Fakultät für Mathematik, TU Dortmund University, Germany
felix.hommelsheim@math.tu-dortmund.de

Takehiro Ito
Graduate School of Information Sciences, Tohoku University, Sendai, Japan
takehiro@tohoku.ac.jp

Yusuke Kobayashi
Research Institute for Mathematical Sciences, Kyoto University, Japan
yusuke@kurims.kyoto-u.ac.jp

Moritz Mühlenthaler
Laboratoire G-SCOP, Grenoble INP, Université Grenoble Alpes, France
moritz.muhlenthaler@grenoble-inp.fr

Akira Suzuki
Graduate School of Information Sciences, Tohoku University, Sendai, Japan
a.suzuki@ecei.tohoku.ac.jp

Abstract

Non-deterministic constraint logic (NCL) is a simple model of computation based on orientations of a constraint graph with edge weights and vertex demands. NCL captures PSPACE and has been a useful tool for proving algorithmic hardness of many puzzles, games, and reconfiguration problems. In particular, its usefulness stems from the fact that it remains PSPACE-complete even under severe restrictions of the weights (e.g., only edge-weights one and two are needed) and the structure of the constraint graph (e.g., planar AND/OR graphs of bounded bandwidth). While such restrictions on the structure of constraint graphs do not seem to limit the expressiveness of NCL, the building blocks of the constraint graphs cannot be limited without losing expressiveness: We consider as parameters the number of weight-one edges and the number of weight-two edges of a constraint graph, as well as the number of AND or OR vertices of an AND/OR constraint graph. We show that NCL is fixed-parameter tractable (FPT) for any of these parameters. In particular, for NCL parameterized by the number of weight-one edges or the number of AND vertices, we obtain a linear kernel. It follows that, in a sense, NCL as introduced by Hearn and Demaine is defined in the most economical way for the purpose of capturing PSPACE.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph algorithms

Keywords and phrases Combinatorial Reconfiguration, Nondeterministic Constraint Logic, Fixed Parameter Tractability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.15

Related Version A full version of this paper is available at `https://arxiv.org/abs/2011.10385`.

Funding *Tatsuhiko Hatanaka*: Partially supported by JSPS KAKENHI Grant Number JP16J02175, Japan.
Takehiro Ito: Partially supported by JSPS KAKENHI Grant Numbers JP18H04091 and JP19K11814, Japan.

Yusuke Kobayashi: Partially supported by JSPS KAKENHI Grant Numbers 17K19960, 18H05291, and JP20K11692, Japan.
Akira Suzuki: Partially supported by JSPS KAKENHI Grant Numbers JP18H04091 and JP20K11666, Japan.

1 Introduction

Non-deterministic constraint logic (NCL) has been introduced by Hearn and Demaine [7] as a model of computation in order to show that many puzzles and games are complete in their natural complexity classes. For instance, they showed that the 1-player games Sokoban and Rush Hour are PSPACE-complete [7] and there are many follow-up results showing hardness of a large number of puzzles, games, and reconfiguration problems. An NCL constraint graph is a graph with edge-weights one and two and a *configuration* is given by an orientation of the constraint graph, such that the in-weight at each vertex is at least two. Two configurations are *adjacent* if they differ with respect to the orientation of a single edge. The question whether two given configurations are connected by a path, i.e., a sequence of adjacent configurations, is known to be PSPACE-complete, even if the constraint graph is a planar graph of maximum degree three (in fact, a planar AND/OR graph, to be defined shortly) [7]. Similar hardness results are known for the question whether it is possible to reverse a single given edge, or whether there is a transformation between two configurations, such that each edge is reversed at most once.

One of the main advantages of NCL, apart from its simplicity, is its hardness on constraint graphs with a severely restricted structure, which entails strong hardness results for other problems. In particular, NCL is PSPACE-complete on *AND/OR graphs*, which are cubic graphs, where each vertex is either incident to three weight-two edges ("OR vertex") or exactly one weight-two edge ("AND vertex"), see Figure 1. It remains PSPACE-complete if in addition we assume that the constraint graphs are planar [7] and have bounded bandwidth [15]. We investigate the possibility of obtaining a further strengthening by restricting the *composition* of the constraint graph. In particular we consider constraint graphs with a bounded number of weight-one or weight-two edges, and AND/OR graphs with a bounded number of AND or OR vertices. Our main result is that NCL parameterized by any of the four quantities admits an FPT algorithm. That is, for the purpose of capturing PSPACE, the definition of NCL given by Hearn and Demaine is as economical as possible. We furthermore hope that based on our results, NCL may become of interest for investigating the parameterized complexity of puzzles, games, and reconfiguration problems.

In the following we adhere to the historical convention that an edge of weight one (resp., weight two) of a constraint graph is called *red* (resp., *blue*). We refer to the question whether a given configuration of a constraint graph is reachable from another given configuration as

(a) AND vertex. **(b)** OR vertex.

Figure 1 The two types of vertices that occur in AND/OR constraint graphs. Edges must be oriented such that the in-weight at each vertex is at least two. By convention, weight-one edges are red and weight-two edges are blue.

Table 1 Parameterized Complexity of NCL. For entries marked with † we obtain a linear kernel.

Parameter(s)	C2C	C2E
treewidth and max. degree [15]	PSPACE-c	PSPACE-c
transformation length [15]	W[1]-hard	W[1]-hard
transformation length and max. degree [15]	FPT	FPT
# of AND vertices (AND/OR graphs)	FPT† (Cor. 4)	FPT† (Cor. 7)
# of OR vertices (AND/OR graphs)	FPT (Thm. 1)	FPT (Thm. 1)
# of red edges	FPT† (Thm. 3)	FPT† (Cor. 7)
# of blue edges	FPT (Thm. 8)	FPT (Cor. 18)

configuration-to-configuration (*C2C*). Furthermore, by *configuration-to-edge* (C2E) we refer to the question whether, we can reach from a given configuration another one such that a given edge is reversed.

Our Contribution

We consider four natural parameterizations of NCL and show that the corresponding parameterized problems admit FPT algorithms. In particular we consider as parameters

1. the number of AND vertices of an AND/OR graph,
2. the number of OR vertices of an AND/OR graph,
3. the number of red edges of a constraint graph, and
4. the number of blue edges of a constraint graph.

Note that none of these parameterizations trivially leads to an XP algorithm that just enumerates all orientations for the constant number of red/blue edges according to the parameter. For an overview of the parameterized complexity results on NCL, including our results, please refer to Table 1.

NCL is known to be PSPACE-complete on AND/OR constraint graphs, which are undirected edge-weighted graphs where each vertex is either and AND vertex or an OR vertex as shown in Figure 1. We show that C2C and C2E parameterized by the number of AND vertices or the number of OR vertices admits an FPT algorithm. The algorithm first performs a preprocessing step followed by a reduction to the problem Binary Constraint Satisfiability Reconfiguration (BCSR for short). Hatanaka et al. have shown that BCSR can be solved in time $O^*(d^{O(p)})$, where d and p are the maximum size of a domain and the number of non-Boolean variables, respectively [5].

On general constraint graphs we obtain a linear kernel for C2C parameterized by the number of red edges. For this purpose we introduce three reduction rules, which, when applied exhaustively, yield a kernel of linear size. To the best of our knowledge, this is the first polynomial kernel for a parameterization of NCL. The first rule states that each component containing at least two blue cycles can be replaced by a gadget of constant size for each red edge that is attached to the component. The second rule states that vertices incident to a blue edge only can be deleted, since the orientation of this edge is the same for every orientation. The third rule is inverse to subdividing a blue edge: any vertex incident to precisely two blue edges can be deleted and replaced by a single blue edge connecting its former neighbors. Note that the number of red edges in an AND/OR graph is precisely the number of AND vertices in an AND/OR graph. Hence, a linear kernel for NCL parameterized by the number of red edges implies a linear kernel for NCL parameterized by the number of AND vertices of an AND/OR graph. Furthermore, we show that slightly modified reduction rules can be applied in order to obtain a linear kernel for C2E.

Finally, we consider C2C and C2E parameterized by the number k of blue edges and show that it admits an FPT algorithm. Our key idea is to partition the set of feasible orientations of the constraint graph into $2^{O(k)}$ classes, such that in each class, all blue edges are oriented in the same way and the red edges have the same indegree sequence. Denote the set of these classes by $\mathcal{F}$, and define a mapping ϕ from the set of orientations of the constraint graph to $\mathcal{F}$ (see Section 5.1 for the details). Then, in Section 5.2, we define an adjacency relation between elements in $\mathcal{F}$, which is consistent with the reachability of configurations of the constraint graph in some sense. In our algorithm, instead of the original reconfiguration problem, we first solve the reconfiguration problem in $\mathcal{F}$, which can be done in $2^{O(k)} \cdot \mathrm{poly}(|V|)$ time, where V is the set of vertices of the constraint graph. If it is impossible to reach the target configuration in $\mathcal{F}$, then we can conclude that it is also impossible with respect to the original constraint graph. Otherwise, we can reduce the original problem to the case that the blue edges agree in the initial and target configuration and the set of red edges in the initial configurations whose orientation differs from the target configuration consists of arc-disjoint dicycles (see Section 5.3). Finally, in Section 5.4, we test whether the direction of each dicycle can be reversed or not.

Related Work

A large number of puzzles, games, and reconfiguration problems have been shown to be hard using reductions from NCL and its variants. Examples include motion planning problems, where rectangular pieces have to be moved to certain final positions and sliding block puzzles such as Rush Hour [3, 7], Sokoban [7], Snowman [6] and other puzzle games such as Bloxors [16]. In the *bounded length* version of NCL, the orientation of each edge may be reversed at most once. This variant has been used to show NP-completeness of the games Klondike, Mahjong Solitaire and Nonogram [8]. Note that NCL gives a uniform view on games as computation and often allows for simpler proofs and strengthenings of known complexity results in this area. Furthermore, deciding proof equivalence in multiplicative linear logic has been shown to be PSPACE-complete by a reduction from NCL [9].

NCL is also very useful for showing hardness of reconfiguration problems. In a reconfiguration problem we are given two configurations and agree on some simple "move" that produces a new configuration from a given one. The question is whether we can reach the second configuration from the first by a sequence of moves. For surveys on reconfiguration problems, please refer to [12, 14]. For many reconfiguration problems, such as token sliding on graphs [7], a variant of independent set reconfiguration [11], as well as vertex cover reconfiguration [10], dominating set reconfiguration [4], reconfiguration of paths [2], and deciding Kempe-equivalence of 3-colorings [1], reductions from NCL establish PSPACE-hardness even on planar graphs of low maximum degree. Van der Zanden showed that there is some constant c, such that NCL is PSPACE-complete on planar subcubic graphs of bandwidth at most c [15]. Note that this property is often maintained in the reductions [1, 2, 4, 7, 10] and it implies that NCL remains hard on graphs of bounded treewidth.

Tractable special cases of NCL have received much less attention. Concerning parameterized complexity, NCL remains PSPACE-complete when parameterized by treewidth and maximum degree of the constraint graph. On the other hand, NCL parameterized by the length of the transformation is W[1]-hard and it becomes FPT when parameterized by the length of the transformation and the maximum degree [15]. If additionally each edge may be reversed at most once in a transformation, NCL is FPT when parameterized by treewidth and the maximum degree, or by the length of the transformation [15].

Organization

The paper is organized as follows. In the next section we give some preliminaries about NCL and introduce notation used throughout the paper. Section 3 contains our FPT algorithm for NCL parameterized by the number of OR vertices. The linear kernel for NCL parameterized by the number of red edges, which also implies the result for AND vertices, can be found in Section 4. Finally, in Section 5 we give an FPT algorithm for NCL parameterized by the number of blue edges. Section 6 concludes the paper and gives some open problems.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph, which may have multiple edges and (self) loops. We denote by $V(G)$ (resp., $E(G)$) the set of vertices (resp., set of edges) G. Each edge in an undirected graph which joins two vertices x and y is represented as an unordered pair xy (or equivalently yx). On the other hand, each arc in a digraph which leaves x and enters y is written as an ordered pair (x, y). Let (V, E, w) be a constraint graph, that is, an undirected graph (V, E) with edge weights $w : E \to \{1, 2\}$. We denote by E^{red} and E^{blue} the sets of red (weight one) and blue (weight two) edges in E, respectively, and have that $E = E^{\mathrm{red}} \cup E^{\mathrm{blue}}$. We denote by $V_{\mathrm{AND}}(G)$ and $V_{\mathrm{OR}}(G)$ the sets of AND and OR vertices in a graph G, respectively; we sometimes drop G, and simply write V_{AND} and V_{OR} if it is clear from the context. A constraint graph is called *AND/OR graph* if each vertex is an AND or OR vertex; thus, an AND/OR graph is 3-regular.

An *orientation* A of E is a multi-set of arcs obtained by replacing each edge in E with a single arc having the same end vertices. We refer to G as the underlying graph of the digraph (V, A). For an orientation A of E, we always denote by A^{red} and A^{blue} the subsets of A corresponding to E^{red} and E^{blue}, respectively. For any arc subset $B \subseteq A$ and a vertex $v \in V$, let $\rho_B(v)$ denote the number of arcs in B that enter v. Then, ρ_B can be regarded as a vector in $\mathbb{Z}_{\geq 0}^{V}$, where $\mathbb{Z}_{\geq 0}$ is the set of all nonnegative integers. An orientation A of E is *feasible* if $\rho_{A^{\mathrm{red}}}(v) + 2 \cdot \rho_{A^{\mathrm{blue}}}(v) \geq 2$ for every $v \in V$; a feasible orientation is synonymously referred to as *configuration*.

For two orientations B and B' of an edge subset $F \subseteq E$, we write $B \leftrightarrow B'$ if $B = B'$ or there exists an arc $(x, y) \in B$ such that $B' = (B \setminus \{(x, y)\}) \cup \{(y, x)\}$. For notational convenience, we simply write $B' = B - (x, y) + (y, x)$ in the latter case. For an orientation B of F, *reversing* the direction of an edge $xy \in F$ is the operation which yields from B an orientation B' of F, such that $B' = B - (x, y) + (y, x)$ if $(x, y) \in B$ and $B - (y, x) + (x, y)$ otherwise. For two feasible orientations A and A' of E, a sequence $\langle A_0, A_1, \ldots, A_\ell \rangle$ of feasible orientations of E is called a *reconfiguration sequence* between A and A' if $A_0 = A$, $A_\ell = A'$, and $A_{i-1} \leftrightarrow A_i$ for all $i \in \{1, 2, \ldots, \ell\}$. We write $A \leftrightsquigarrow A'$ if there exists a reconfiguration sequence between A and A' (or $A \not\leftrightsquigarrow A'$ if not). Given a constraint graph G and two feasible orientations A_{ini} and A_{tar} of $E(G)$, the problem C2C asks whether $A_{\mathrm{ini}} \leftrightsquigarrow A_{\mathrm{tar}}$ or not. Similarly, given a constraint graph (G, w), a feasible orientation A_{ini} of $E(G)$, and an edge $e \in E(G)$, the problem C2E asks whether there is a feasible orientation A_{tar}, such that $A_{\mathrm{ini}} \leftrightsquigarrow A_{\mathrm{tar}}$ and the direction of e is different in A_{ini} and A_{tar}. We denote by a triple $(G, A_{\mathrm{ini}}, A_{\mathrm{tar}})$ an instance of C2C and by a triple $(G, A_{\mathrm{ini}}, vw)$ an instance of C2E.

3 NCL for AND/OR graphs

In this section, we consider NCL when restricted to AND/OR constraint graphs. Recall that NCL remains PSPACE-complete on AND/OR graphs [7]. We thus prove that C2C and C2E on AND/OR constraint graphs is fixed-parameter tractable when parameterized by the

number of OR vertices. An analogous result for C2C and C2E parameterized by the number of AND vertices follows from our FPT result for NCL parameterized by the number of red edges in the next section (see Theorem 3). Therefore, the main result here is the following theorem.

▶ **Theorem 1.** *C2C and C2E on AND/OR constraint graphs with n vertices parameterized by the number k of OR vertices admits a $2^{O(k)} \cdot \mathrm{poly}(n)$-time algorithm.*

In the reminder of this section, we give an overview of the proof of Theorem 1. Our strategy is to give an FPT-reduction from C2C on AND/OR constraint graphs to the BINARY CONSTRAINT SATISFIABILITY RECONFIGURATION problem (BCSR, for short) [5], which will be defined in Section 3.2. To do so, we first apply some preprocessing to a given instance of C2C on an AND/OR graph (in Section 3.1), and then give our FPT-reduction to BCSR (in Section 3.2). By similar arguments we obtain the result for C2E.

3.1 Preprocessing

The preprocessing subdivides each blue edge that is not a loop into two blue edges. It is not hard to see that a single subdivision yields an equivalent instance: Let uv be a blue edge of a constraint graph $\hat{G}$ and consider the constraint graph G obtained by subdividing uv into two blue edges uz and zv, where z is a new vertex we call *middle vertex*. Let G be the resulting constraint graph and observe that from any feasible orientation $\hat{A}$ of $\hat{G}$ we may obtain a feasible orientation A of G by letting $A = \hat{A} - (u,v) + (u,z) + (z,v)$ if $(u,v) \in \hat{A}$ and $A = \hat{A} - (v,u) + (v,z) + (z,u)$ otherwise.

Furthermore, in any feasible orientation of $\hat{G}$, we can transfer in-weight from, say, u to v by reversing the arc (v,u) iff the in-weight at u is at least four. Furthermore, due to the orientation of uv, the corresponding arc contributes to the in-weight of precisely one of u and v. Conversely, in an orientation of G, we can transfer in-weight from, say, u to v by reversing the directions of the arcs corresponding to uz and zv iff the in-weight at u is at least four. Furthermore, in any orientation of G, the arcs corresponding to uz and zv contribute in-weight to at most one of u and v. Hence, by subdividing a blue edge of $\hat{G}$ from an instance $(\hat{G}, \hat{A}_{\mathrm{ini}}, \hat{A}_{\mathrm{tar}})$ of C2C, we obtain an equivalent instance. Let $(G, A_{\mathrm{ini}}, A_{\mathrm{tar}})$ be the instance of C2C obtained from $(\hat{G}, \hat{A}_{\mathrm{ini}}, \hat{A}_{\mathrm{tar}})$ by subdividing each blue edge of $\hat{G}$ that is not a loop. By repetition of the above argument we obtain the following result.

▶ **Lemma 2.** $(G, A_{\mathrm{ini}}, A_{\mathrm{tar}})$ *is a* yes*-instance if and only if* $(\hat{G}, \hat{A}_{\mathrm{ini}}, \hat{A}_{\mathrm{tar}})$ *is.*

3.2 FPT-reduction to BCSR

In this subsection, we sketch our FPT-reduction to BCSR. We start by formally defining the problem BCSR. Let $H = (X, F)$ be an undirected graph. We call each vertex $x \in X$ a *variable*. Each $x \in X$ has a finite set $D(x)$, called a *domain of* x. A variable x is called a *Boolean variable* if $|D(x)| \leq 2$, and otherwise called a *non-Boolean variable*. Each edge $xy \in F$ has a subset $\mathcal{C}(xy) \subseteq D(x) \times D(y)$, called a (*binary*) *constraint of* xy. A mapping $\Gamma\colon X \to \bigcup_{x \in X} D(x)$ is a *solution* of H if $\Gamma(x) \in D(x)$ for every $x \in X$. In addition, a solution Γ of H is *proper* if $\Gamma(x)\Gamma(y) \in \mathcal{C}(xy)$ for every $xy \in F$. For two solutions Γ and Γ', we write $\Gamma \leftrightarrow \Gamma'$ if $|\{x \in X : \Gamma(x) \neq \Gamma'(x)\}| = 1$. Given an undirected graph H, a domain $D(x)$ for each $x \in X$, a constraint $\mathcal{C}(xy)$ for each $xy \in F$, and two proper solutions Γ_{ini} and Γ_{tar} of H, the BINARY CONSTRAINT SATISFIABILITY RECONFIGURATION problem (BCSR) asks whether there exists a sequence $\langle \Gamma_0, \Gamma_1, \ldots, \Gamma_\ell \rangle$ of proper solutions of H such that $\Gamma_0 = \Gamma_{\mathrm{ini}}$, $\Gamma_\ell = \Gamma_{\mathrm{tar}}$, and $\Gamma_{i-1} \leftrightarrow \Gamma_i$ for each $i \in \{1, 2, \ldots, \ell\}$. Let $(H, D, \mathcal{C}, \Gamma_{\mathrm{ini}}, \Gamma_{\mathrm{tar}})$ an instance of BCSR.

It is known that BCSR can be solved in time $O^*(d^{O(p)})$, where $d := \max_{x \in X} |D(x)|$ and p is the number of non-Boolean variables in X [5, Theorem 18]. To prove Theorem 1, given an instance $(\hat{G}, \hat{A}_{\text{ini}}, \hat{A}_{\text{tar}})$ of C2C on an AND/OR constraint graph with at most k AND/OR vertices, we first perform the preprocessing from Section 3.1 to obtain an instance $(G, A_{\text{ini}}, A_{\text{tar}})$ of C2C. Note that G is not an AND/OR graph, and V can be partitioned into $V_{\text{AND}}(G)$, $V_{\text{OR}}(G)$ and $V_{\text{MID}}(G)$, where $V_{\text{MID}}(G)$ (or simply V_{MID}) is the set of middle vertices in G. By Lemma 2, we have that $(G, A_{\text{ini}}, A_{\text{tar}})$ is a yes-instance if and only if $(\hat{G}, \hat{A}_{\text{ini}}, \hat{A}_{\text{tar}})$. Hence, to conclude the proof of Theorem 1, we provide an FPT-reduction from a preprocessed instance $(G, A_{\text{ini}}, A_{\text{tar}})$ of C2C with the parameter $|V_{\text{OR}}(G)| = |V_{\text{OR}}(\hat{G})| \leq k$ to an instance $(H, D, \mathcal{C}, \Gamma_{\text{ini}}, \Gamma_{\text{tar}})$ of BCSR such that both d and p are bounded by some computable functions depending only on k.

Due to the preprocessing, observe that the constraint graph G has no two parallel blue edges. In addition, no edge in G joins an AND vertex and an OR vertex, and hence we can partition E into two sets E_{AND} and E_{OR}, defined as follows: E_{AND} is the set of edges of G that are incident to an AND vertex; E_{OR} is the set of edges of G that are incident to an OR vertex. The high-level idea of the reduction to BCSR is the following. For each OR vertex v, we create an OR variable x_v. Observe that the in-weight requirement at v is violated only if each arc is pointing away from v. We forbid such orientations by giving each OR variable x_v a domain of size seven corresponding to the seven legal orientations of the incident edges of v.

The remaining in-weight requirements and consistency requirements are modelled by adding constraints, which also define the set of edges in H. For each edge e of G, we create a Boolean edge-variable x_e, whose domain represents the two possible orientations of an edge. The construction of domains above ensures that in-weight requirement is satisfied for each AND vertex. To ensure the same property for all other vertices, we add three types of constraints for middle vertices and AND vertices, to enforce the following constraints:

Type 1: Constraints for middle vertices.

Let v be a middle vertex between two vertices v_1 and v_2. Since both v_1v and vv_2 are blue edges, the in-weight requirement at v is satisfied if and only if v_1v or vv_2 points to v.

Type 2-1: Constraints for AND vertices having loops.

Let v be an AND vertex having a loop vv. So vv must be red and the remaining edge $vv_3 \in E_{\text{AND}}$ is blue where v_3 is a middle vertex. Then, the in-weight requirement at v is satisfied if and only if vv_3 is oriented towards v.

Type 2-2: Constraints for AND vertices without loops.

Let v be an AND vertex, and let vv_1, vv_2, vv_3 be three (distinct) edges incident to v such that vv_1 and vv_2 are red, and vv_3 is blue; it may hold that $v_1 = v_2$. Then, the in-degree requirement at v is satisfied if and only if i) vv_1 or vv_3 are oriented towards v and ii) vv_2 or vv_3 are oriented towards v.

By the construction of constraints above, we know that a solution Γ of H is proper if and only if the corresponding orientation A_Γ of E is feasible. Therefore, we can define proper solutions Γ_{ini} and Γ_{tar} of H which correspond to feasible orientations A_{ini} and A_{tar} of E, respectively. In this way, from a preprocessed instance $(G, A_{\text{ini}}, A_{\text{tar}})$ of C2C with the parameter $|V_{\text{OR}}(G)| \leq k$, we have constructed in polynomial time a corresponding equivalent instance $(H, D, \mathcal{C}, \Gamma_{\text{ini}}, \Gamma_{\text{tar}})$ of BCSR such that $d = \max_{x \in X} |D(x)| = 7$ and $p \leq |V_{\text{OR}}(G)| \leq k$.

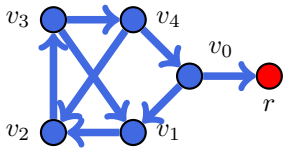


Figure 2 The gadget used in reduction rule 2.

4 NCL parameterized by the number of red edges

Our main result in this section is a linear kernel for C2C parameterized by the number of red edges of the constraint graph.

▶ **Theorem 3.** *There is a polynomial-time algorithm that, given an instance of C2C on a constraint graph with k red edges, outputs an equivalent instance of C2C of size $O(k)$.*

In particular, Theorem 3 implies that C2C parameterized by the number of red edges admits a $O^*(2^{O(k)})$-time algorithm. By observing that in any AND/OR constraint graph, the number of red edges is equal to the number of AND vertices, we immediately obtain the following result.

▶ **Corollary 4.** *C2C on AND/OR graphs parameterized by the number k' of AND vertices admits a kernel of size $O(k')$.*

It can be shown by similar arguments that there is also a linear kernel for C2E parameterized by the number of red edges of the constraint graph. In the remainder of this section, we prove Theorem 3. Let $I = (G, A_{\text{ini}}, A_{\text{tar}})$ be an instance of C2C, where G is any constraint graph with k red edges. We give four reduction rules, and show that applying them repeatedly preserves the answer. Furthermore, we show that applying them exhaustively yields an instance of size $O(k)$, where $k = |E^{\text{red}}|$. To conclude the proof, we show that the reduction can be applied in polynomial time.

We say that a vertex is *blue* if all its incident edges are blue. Otherwise, if at least one incident edge is red, we call the vertex *red*. A subset $V' \subseteq V$ is called a *blue component* if it is a connected component in the graph (V, E^{blue}). Note that a blue component may contain red vertices of G. The first reduction rule removes blue components of G that are directed cycles. Observe that no arc in such a component can be reversed. The second reduction rule removes blue components that contain at least two cycles and attaches to each red vertex v of the component a copy of the gadget shown in Figure 2. The gadget consists of a cycle on five new vertices $\{v_0, v_1, v_2, v_3, v_4\}$ with two chords $\{v_1, v_3\}$ and $\{v_2, v_4\}$. Additionally we add an edge joining v and v_0. All edges of the gadget have weight two. The third reduction rule removes blue vertices of degree one and the last rule removes the center vertex of a blue path on three vertices.

While modifying G we also modify A_{ini} and A_{tar} accordingly. That is, if we delete edges of G, these edges are also deleted in A_{ini} and A_{tar}. If we add a gadget to G, then the arcs in A_{ini} and A_{tar} have the same orientation on the gadget. Note that the number k of red vertices is not altered by an application of any of the rules. Here is a more formal description of the four rules:

▶ **Reduction rule 1.** *Let C be a component of G that is a blue chordless cycle. If the orientations A_{ini} and A_{tar} agree on C, then we remove C from the graph and adjust A_{ini} and A_{tar} accordingly. Otherwise we output a* no*-instance.*

▶ **Reduction rule 2.** *Let C be a blue component that contains at least two cycles. Then we remove from G every blue vertex in C and attach to each red vertex in C a copy of the gadget in Figure 2. Additionally we modify A_{ini} and A_{tar} accordingly such that both agree on each copy of the gadget.*

▶ **Reduction rule 3.** *If G has a blue vertex v of degree one, delete v and its incident edge from G and remove the corresponding arc(s) from A_{ini} and A_{tar}.*

▶ **Reduction rule 4.** *Suppose G has a blue vertex v of degree 2, such that the two neighbors u and w of v are non-adjacent in G. Then delete v and its incident edges from G and add the blue edge uw. Remove any arcs incident to v from A_{ini} and A_{tar}. Finally, add (u, w) to A_{ini} (resp. A_{tar}) if $(u, v) \in A_{\mathrm{ini}}$ (resp., A_{tar}) and (w, u) otherwise.*

We show that applying any of the four rules is *safe*, that is any application results in a yes-instance if and only if I is a yes-instance.

▶ **Proposition 5.** *Reduction rules 1–4 are safe for C2C.*

By applying a depth-first-search we can check if any of the rules can be applied. Thus we have the following.

▶ **Proposition 6.** *Reduction rules 1-4 can be applied exhaustively in time $O(|V| \cdot (|V| + |E|))$.*

Theorem 3 now follows by the previous propositions and a simple counting argument. Using similar arguments we show that there is also a linear kernel for C2E parameterized by the number k of red edges. The main difference is that in reduction rule 2 we only add the gadget to each red vertex that is part of a cycle or connected to two distinct cycles by two disjoint paths. Furthermore, if the edge e that we wish to reverse is part of of a component containing two cycles, we add a gadget to the tail of e.

▶ **Corollary 7.** *C2E parameterized by the number k of red edges admits a kernel of size $O(k)$. Furthermore, C2E on AND/OR graphs parameterized by the number k' of AND vertices admits a kernel of size $O(k')$.*

5 NCL parameterized by the number of blue edges

The objective of this section is to show that C2C parameterized by the number k of blue edges is fixed parameter tractable.

▶ **Theorem 8.** *C2C parameterized by the number k of blue edges can be solved in time $2^{O(k)} \cdot \mathrm{poly}(|V|)$.*

In the remainder of this section, we prove Theorem 8. Let $I = (G, A_{\mathrm{ini}}, A_{\mathrm{tar}})$ be an instance of C2C, where G is any constraint graph and denote by V and E the set of vertices and edges of G, respectively.

Let $\mathcal{A}$ denote the set of all feasible orientations of E. Our key idea is to classify the feasible orientations into $2^{O(k)}$ classes, where each class is determined by the orientation A^{blue} of E^{blue} and the indegree sequence of A^{red}. Denote the set of these classes by $\mathcal{F}$, and define a mapping ϕ from $\mathcal{A}$ to $\mathcal{F}$ (see Section 5.1 for details). Then, in Section 5.2, we define a reconfiguration relation $\overset{}{\underset{\mathcal{F}}{\leftrightsquigarrow}}$ between elements in $\mathcal{F}$, which is consistent with $\leftrightsquigarrow$ in some sense. In our algorithm, instead of the original reconfiguration problem in $\mathcal{A}$, we first solve the reconfiguration problem in $\mathcal{F}$, which can be done in $2^{O(k)} \cdot \mathrm{poly}(|V|)$ time. If it has no reconfiguration sequence, then we can conclude that there is no reconfiguration sequence

in the original problem. Otherwise, we can reduce the original problem to the case when $A^{\mathrm{blue}}_{\mathrm{ini}} = A^{\mathrm{blue}}_{\mathrm{tar}}$ and $A^{\mathrm{red}}_{\mathrm{ini}} \setminus A^{\mathrm{red}}_{\mathrm{tar}}$ consists of arc-disjoint dicycles (see Section 5.3). Finally, in Section 5.4, we test whether the direction of each dicycle can be reversed or not.

Before starting the main part of the proof of Theorem 8, we show the following lemma that plays an important role in our argument. Roughly, it says that we can change the orientation of E^{red} keeping a certain indegree constraint.

▶ **Lemma 9.** *Let $A^{\mathrm{red}}_{\mathrm{ini}}$ and $A^{\mathrm{red}}_{\mathrm{tar}}$ be orientations of E^{red}. Then, there exists a sequence $A^{\mathrm{red}}_0, A^{\mathrm{red}}_1, \ldots, A^{\mathrm{red}}_l$ of orientations of E^{red} such that $A^{\mathrm{red}}_0 = A^{\mathrm{red}}_{\mathrm{ini}}$, $\rho_{A^{\mathrm{red}}_l} = \rho_{A^{\mathrm{red}}_{\mathrm{tar}}}$, $A^{\mathrm{red}}_{i-1} \leftrightarrow A^{\mathrm{red}}_i$ for $i = 1, \ldots, l$, and $\rho_{A^{\mathrm{red}}_i}(v) \geq \min\{\rho_{A^{\mathrm{red}}_{\mathrm{ini}}}(v), \rho_{A^{\mathrm{red}}_{\mathrm{tar}}}(v)\}$ for any $v \in V$ and any $i \in \{0, 1, \ldots, l\}$.*

Proof. We prove the lemma by induction on $|A^{\mathrm{red}}_{\mathrm{ini}} \setminus A^{\mathrm{red}}_{\mathrm{tar}}|$. If $\rho_{A^{\mathrm{red}}_{\mathrm{ini}}} = \rho_{A^{\mathrm{red}}_{\mathrm{tar}}}$, then the claim is obvious, because the sequence consisting of only one orientation $A^{\mathrm{red}}_0 = A^{\mathrm{red}}_{\mathrm{ini}}$ satisfies the conditions. Thus, it suffices to consider the case when $\rho_{A^{\mathrm{red}}_{\mathrm{ini}}} \neq \rho_{A^{\mathrm{red}}_{\mathrm{tar}}}$. In this case, there exists a vertex $u \in V$ such that $\rho_{A^{\mathrm{red}}_{\mathrm{ini}}}(u) > \rho_{A^{\mathrm{red}}_{\mathrm{tar}}}(u)$, because $\sum_{v \in V} \rho_{A^{\mathrm{red}}_{\mathrm{ini}}}(v) = \sum_{v \in V} \rho_{A^{\mathrm{red}}_{\mathrm{tar}}}(v)$. Then, there exists an arc $a \in A^{\mathrm{red}}_{\mathrm{ini}} \setminus A^{\mathrm{red}}_{\mathrm{tar}}$ that enters u. Let A^{red}_1 be the orientation of E^{red} obtained from $A^{\mathrm{red}}_{\mathrm{ini}}$ by reversing the direction of a. Since $|A^{\mathrm{red}}_1 \setminus A^{\mathrm{red}}_{\mathrm{tar}}| < |A^{\mathrm{red}}_{\mathrm{ini}} \setminus A^{\mathrm{red}}_{\mathrm{tar}}|$, by induction hypothesis, there exists a sequence $A^{\mathrm{red}}_1, \ldots, A^{\mathrm{red}}_l$ of orientations of E^{red} such that $\rho_{A^{\mathrm{red}}_l} = \rho_{A^{\mathrm{red}}_{\mathrm{tar}}}$, $A^{\mathrm{red}}_{i-1} \leftrightarrow A^{\mathrm{red}}_i$ for $i = 2, \ldots, l$, and $\rho_{A^{\mathrm{red}}_i}(v) \geq \min\{\rho_{A^{\mathrm{red}}_1}(v), \rho_{A^{\mathrm{red}}_{\mathrm{tar}}}(v)\}$ for any $v \in V$ and any $i \in \{1, \ldots, l\}$. By letting $A^{\mathrm{red}}_0 = A^{\mathrm{red}}_{\mathrm{ini}}$, the sequence $A^{\mathrm{red}}_0, A^{\mathrm{red}}_1, \ldots, A^{\mathrm{red}}_l$ satisfies the conditions, because $A^{\mathrm{red}}_0 \leftrightarrow A^{\mathrm{red}}_1$, $\rho_{A^{\mathrm{red}}_1}(v) \geq \rho_{A^{\mathrm{red}}_{\mathrm{ini}}}(v)$ for each $v \in V \setminus \{u\}$, and $\min\{\rho_{A^{\mathrm{red}}_1}(u), \rho_{A^{\mathrm{red}}_{\mathrm{tar}}}(u)\} = \rho_{A^{\mathrm{red}}_{\mathrm{tar}}}(u) = \min\{\rho_{A^{\mathrm{red}}_{\mathrm{ini}}}(u), \rho_{A^{\mathrm{red}}_{\mathrm{tar}}}(u)\}$. ◀

The proof of Lemma 9 is constructive, and hence we can find such a sequence efficiently.

5.1 Classification of $\mathcal{A}$

In this subsection, we classify the feasible orientations into $2^{O(k)}$ classes. Let $X \subseteq V$ be the set of all vertices to which edges in E^{blue} are incident. Define $\mathcal{F}$ as the set of all pairs (A^{blue}, d) where A^{blue} is an orientation of E^{blue} and d is a vector in $\{0, 1, 2\}^X$ satisfying the following conditions:

(1) $2\rho_{A^{\mathrm{blue}}}(v) + d(v) \geq 2$ for any $v \in X$.

(2) There exists an orientation A^{red} of E^{red} such that for any $v \in V$,

$$\rho_{A^{\mathrm{red}}}(v) \begin{cases} = 0 & \text{if } v \in X \text{ and } d(v) = 0, \\ = 1 & \text{if } v \in X \text{ and } d(v) = 1, \text{ and} \\ \geq 2 & \text{otherwise.} \end{cases}$$

We note that $|\mathcal{F}| \leq 2^{|E^{\mathrm{blue}}|} \cdot 3^{|X|} = 2^{O(k)}$, because $|X| \leq 2|E^{\mathrm{blue}}|$. For a vector $d \in \{0, 1, 2\}^X$, we say that an orientation A^{red} of E^{red} *realizes* d if A^{red} satisfies the condition (2) above. We can easily see that if $(A^{\mathrm{blue}}, d) \in \mathcal{F}$ holds and A^{red} realizes d, then $A := A^{\mathrm{blue}} \cup A^{\mathrm{red}}$ is a feasible orientation of E. Conversely, if $A = A^{\mathrm{blue}} \cup A^{\mathrm{red}}$ is a feasible orientation of E (i.e., $A \in \mathcal{A}$), then the vector $d \in \{0, 1, 2\}^X$ defined by $d(v) = \min\{\rho_{A^{\mathrm{red}}}(v), 2\}$ for each $v \in X$ satisfies that $(A^{\mathrm{blue}}, d) \in \mathcal{F}$. This defines a mapping ϕ from $\mathcal{A}$ to $\mathcal{F}$.

We can also see that the membership problem of $\mathcal{F}$ can be decided in polynomial time.

▶ **Lemma 10.** *For an orientation A^{blue} of E^{blue} and a vector $d \in \{0, 1, 2\}^X$, we can test whether $(A^{\mathrm{blue}}, d) \in \mathcal{F}$ or not in polynomial time.*

Proof. We can easily check the condition (1). To check the condition (2), we construct a digraph $\hat{G} = (\hat{V}, \hat{A})$ and consider a network flow problem in it. Introduce a new vertex w_e for each $e \in E^{\mathrm{red}}$ and two new vertices s and t, and define $\hat{V} := V \cup \{w_e \mid e \in E^{\mathrm{red}}\} \cup \{s, t\}$. Define the arc set $\hat{A} := \hat{A}_1 \cup \hat{A}_2 \cup \hat{A}_3$ by

$$\hat{A}_1 := \{(s, w_e) \mid e \in E^{\mathrm{red}}\},$$
$$\hat{A}_2 := \{(w_e, v) \mid e \in E^{\mathrm{red}}, v \in V, e \text{ is incident to } v \text{ in } G\},$$
$$\hat{A}_3 := \{(v, t) \mid v \in V\}.$$

For each $a \in \hat{A}$, define the lower bound $l(a)$ and the upper bound $u(a)$ of the amount of flow through a as follows.

- For each $(s, w_e) \in \hat{A}_1$, define $l(s, w_e) := u(s, w_e) := 1$.
- For each $(w_e, v) \in \hat{A}_2$, define $l(w_e, v) := 0$ and $u(w_e, v) := 1$.
- For each $(v, t) \in \hat{A}_3$, define $l(v, t) := u(v, t) := d(v)$ if $v \in X$ and $d(v) \in \{0, 1\}$, and define $l(v, t) := 2$ and $u(v, t) := +\infty$ otherwise.

Then, the condition (2) holds if and only if $\hat{G}$ has an integral s-t flow satisfying the above constraint. This can be tested in polynomial time by a standard maximum flow algorithm (see e.g. [13, Corollary 11.3a]). ◀

5.2 Reconfiguration in $\mathcal{F}$

In this subsection, we consider a reconfiguration between elements in $\mathcal{F}$. For $(A_1^{\mathrm{blue}}, d_1)$, $(A_2^{\mathrm{blue}}, d_2) \in \mathcal{F}$, we denote $(A_1^{\mathrm{blue}}, d_1) \underset{\mathcal{F}}{\longleftrightarrow} (A_2^{\mathrm{blue}}, d_2)$ if

- $d_1 = d_2$ and $A_1^{\mathrm{blue}} \leftrightarrow A_2^{\mathrm{blue}}$, or
- $A_1^{\mathrm{blue}} = A_2^{\mathrm{blue}}$.

If there exists a sequence $(A_0^{\mathrm{blue}}, d_0), (A_1^{\mathrm{blue}}, d_1), \ldots, (A_l^{\mathrm{blue}}, d_l) \in \mathcal{F}$ such that $(A_{i-1}^{\mathrm{blue}}, d_{i-1}) \underset{\mathcal{F}}{\longleftrightarrow} (A_i^{\mathrm{blue}}, d_i)$ for $i = 1, \ldots, l$, then we denote $(A_0^{\mathrm{blue}}, d_0) \underset{\mathcal{F}}{\leftrightsquigarrow} (A_l^{\mathrm{blue}}, d_l)$. Then, we can easily see the following.

▶ **Lemma 11.** *Let $A_{\mathrm{ini}}, A_{\mathrm{tar}} \in \mathcal{A}$. If $A_{\mathrm{ini}} \leftrightsquigarrow A_{\mathrm{tar}}$, then $\phi(A_{\mathrm{ini}}) \underset{\mathcal{F}}{\leftrightsquigarrow} \phi(A_{\mathrm{tar}})$.*

Proof. If $A_{\mathrm{ini}} \leftrightarrow A_{\mathrm{tar}}$, then $\phi(A_{\mathrm{ini}}) \underset{\mathcal{F}}{\longleftrightarrow} \phi(A_{\mathrm{tar}})$ by definition. By using this relationship repeatedly, we obtain the claim. ◀

Although the opposite implication is not true, we show the following statement.

▶ **Lemma 12.** *Let $A_{\mathrm{ini}}, A_{\mathrm{tar}} \in \mathcal{A}$. If $\phi(A_{\mathrm{ini}}) \underset{\mathcal{F}}{\leftrightsquigarrow} \phi(A_{\mathrm{tar}})$, then there exists $A^{\circ}_{\mathrm{tar}} \in \mathcal{A}$ such that $\phi(A^{\circ}_{\mathrm{tar}}) = \phi(A_{\mathrm{tar}})$ and $A_{\mathrm{ini}} \leftrightsquigarrow A^{\circ}_{\mathrm{tar}}$.*

Proof. It suffices to consider the case when $\phi(A_{\mathrm{ini}}) \underset{\mathcal{F}}{\longleftrightarrow} \phi(A_{\mathrm{tar}})$. Denote $\phi(A_{\mathrm{ini}}) = (A_{\mathrm{ini}}^{\mathrm{blue}}, d_{\mathrm{ini}})$ and $\phi(A_{\mathrm{tar}}) = (A_{\mathrm{tar}}^{\mathrm{blue}}, d_{\mathrm{tar}})$. By definition, we have either $d_{\mathrm{ini}} = d_{\mathrm{tar}}$ and $A_{\mathrm{ini}}^{\mathrm{blue}} \leftrightarrow A_{\mathrm{tar}}^{\mathrm{blue}}$, or $A_{\mathrm{ini}}^{\mathrm{blue}} = A_{\mathrm{tar}}^{\mathrm{blue}}$.

If $d_{\mathrm{ini}} = d_{\mathrm{tar}}$ and $A_{\mathrm{ini}}^{\mathrm{blue}} \leftrightarrow A_{\mathrm{tar}}^{\mathrm{blue}}$, then $A_{\mathrm{ini}} \leftrightarrow A_{\mathrm{tar}}^{\mathrm{blue}} \cup A_{\mathrm{ini}}^{\mathrm{red}}$ and $\phi(A_{\mathrm{tar}}^{\mathrm{blue}} \cup A_{\mathrm{ini}}^{\mathrm{red}}) = (A_{\mathrm{tar}}^{\mathrm{blue}}, d_{\mathrm{ini}}) = (A_{\mathrm{tar}}^{\mathrm{blue}}, d_{\mathrm{tar}}) = \phi(A_{\mathrm{tar}})$, which means that $A^{\circ}_{\mathrm{tar}} := A_{\mathrm{tar}}^{\mathrm{blue}} \cup A_{\mathrm{ini}}^{\mathrm{red}}$ satisfies the conditions.

Otherwise, let $A^{\mathrm{blue}} := A_{\mathrm{ini}}^{\mathrm{blue}} = A_{\mathrm{tar}}^{\mathrm{blue}}$. By Lemma 9, we obtain a sequence $A_0^{\mathrm{red}}, A_1^{\mathrm{red}}, \ldots, A_l^{\mathrm{red}}$ of orientations of E^{red} such that $A_0^{\mathrm{red}} = A_{\mathrm{ini}}^{\mathrm{red}}$, $\rho_{A_l^{\mathrm{red}}} = \rho_{A_{\mathrm{tar}}^{\mathrm{red}}}$, $A_{i-1}^{\mathrm{red}} \leftrightarrow A_i^{\mathrm{red}}$ for $i = 1, \ldots, l$, and $\rho_{A_i^{\mathrm{red}}}(v) \geq \min\{\rho_{A_{\mathrm{ini}}^{\mathrm{red}}}(v), \rho_{A_{\mathrm{tar}}^{\mathrm{red}}}(v)\}$ for any $v \in V$ and any $i \in \{0, 1, \ldots, l\}$. Then, for any $i \in \{0, 1, \ldots, l\}$, we have

$$2\rho_{A^{\mathrm{blue}}}(v) + \rho_{A_i^{\mathrm{red}}}(v) \geq \min\{2\rho_{A^{\mathrm{blue}}}(v) + \rho_{A_{\mathrm{ini}}^{\mathrm{red}}}(v), 2\rho_{A^{\mathrm{blue}}}(v) + \rho_{A_{\mathrm{tar}}^{\mathrm{red}}}(v)\} \geq 2$$

for any $v \in V$, and hence $A^{\mathrm{blue}} \cup A_i^{\mathrm{red}}$ is feasible. Since $A^{\mathrm{blue}} \cup A_{i-1}^{\mathrm{red}} \leftrightarrow A^{\mathrm{blue}} \cup A_i^{\mathrm{red}}$ for $i = 1, \ldots, l$, we have

$$(A_{\mathrm{ini}} =) A^{\mathrm{blue}} \cup A_{\mathrm{ini}}^{\mathrm{red}} \leftrightsquigarrow A^{\mathrm{blue}} \cup A_l^{\mathrm{red}}.$$

Furthermore, since $\rho_{A_l^{\mathrm{red}}} = \rho_{A_{\mathrm{tar}}^{\mathrm{red}}}$, we have $\phi(A^{\mathrm{blue}} \cup A_l^{\mathrm{red}}) = \phi(A_{\mathrm{tar}})$. Therefore, $A_{\mathrm{tar}}^{\circ} := A^{\mathrm{blue}} \cup A_l^{\mathrm{red}}$ satisfies the conditions in the lemma. ◀

Note that we can construct A_{tar}° and a reconfiguration sequence in Lemma 12 efficiently.

5.3 Algorithm

Let $I = (G, A_{\mathrm{ini}}, A_{\mathrm{tar}})$ be an instance of C2C. We first compute $\phi(A_{\mathrm{ini}})$ and $\phi(A_{\mathrm{tar}})$, and test whether $\phi(A_{\mathrm{ini}}) \underset{\mathcal{F}}{\leftrightsquigarrow} \phi(A_{\mathrm{tar}})$ or not. If $\phi(A_{\mathrm{ini}}) \not\underset{\mathcal{F}}{\leftrightsquigarrow} \phi(A_{\mathrm{tar}})$, then we can immediately conclude that $A_{\mathrm{ini}} \not\leftrightsquigarrow A_{\mathrm{tar}}$ by Lemma 11.

Thus, in what follows, suppose that $\phi(A_{\mathrm{ini}}) \underset{\mathcal{F}}{\leftrightsquigarrow} \phi(A_{\mathrm{tar}})$. In this case, by applying Lemma 12, we can construct $A_{\mathrm{tar}}^{\circ} \in \mathcal{A}$ with $\phi(A_{\mathrm{tar}}^{\circ}) = \phi(A_{\mathrm{tar}})$ such that $A_{\mathrm{ini}} \leftrightsquigarrow A_{\mathrm{tar}}^{\circ}$. This shows that $A_{\mathrm{ini}} \leftrightsquigarrow A_{\mathrm{tar}}$ is equivalent to $A_{\mathrm{tar}}^{\circ} \leftrightsquigarrow A_{\mathrm{tar}}$, which means that we can regard A_{tar}° as a new initial configuration instead of A_{ini}. Thus, the problem is reduced to the case when $\phi(A_{\mathrm{ini}}) = \phi(A_{\mathrm{tar}})$. In particular, we have $A_{\mathrm{ini}}^{\mathrm{blue}} = A_{\mathrm{tar}}^{\mathrm{blue}}$.

Suppose that $A_{\mathrm{ini}}^{\mathrm{blue}} = A_{\mathrm{tar}}^{\mathrm{blue}} =: A^{\mathrm{blue}}$ and $\rho_{A_{\mathrm{ini}}^{\mathrm{red}}} \neq \rho_{A_{\mathrm{tar}}^{\mathrm{red}}}$. Then, by applying Lemma 9, we obtain an orientation A_l^{red} of E^{red} such that $\rho_{A_l^{\mathrm{red}}} = \rho_{A_{\mathrm{tar}}^{\mathrm{red}}}$ and $A_{\mathrm{ini}} \leftrightsquigarrow A^{\mathrm{blue}} \cup A_l^{\mathrm{red}}$. This shows that $A_{\mathrm{ini}} \leftrightsquigarrow A_{\mathrm{tar}}$ is equivalent to $A^{\mathrm{blue}} \cup A_l^{\mathrm{red}} \leftrightsquigarrow A_{\mathrm{tar}}$, which means that we can regard $A^{\mathrm{blue}} \cup A_l^{\mathrm{red}}$ as a new initial configuration instead of A_{ini}. Thus, the problem is reduced to the case when $\rho_{A_{\mathrm{ini}}^{\mathrm{red}}} = \rho_{A_{\mathrm{tar}}^{\mathrm{red}}}$. If $A_{\mathrm{ini}}^{\mathrm{red}} = A_{\mathrm{tar}}^{\mathrm{red}}$, we conclude that $A_{\mathrm{ini}} \leftrightsquigarrow A_{\mathrm{tar}}$. Otherwise, since $\rho_{A_{\mathrm{ini}}^{\mathrm{red}}} = \rho_{A_{\mathrm{tar}}^{\mathrm{red}}}$, the set $A_{\mathrm{ini}}^{\mathrm{red}} \setminus A_{\mathrm{tar}}^{\mathrm{red}}$ can be decomposed into arc-disjoint cycles.

Note that all of the above procedures can be executed in $2^{O(k)} \cdot \mathrm{poly}(|V|)$ time, since $|\mathcal{F}| = 2^{O(k)}$. In what follows, we give an algorithm for testing whether the direction of each cycle can be reversed or not. For this purpose, we show the following lemma.

▶ **Lemma 13.** *Let $A_{\mathrm{ini}} \in \mathcal{A}$ and let C be a dicycle with all the arcs in $A_{\mathrm{ini}}^{\mathrm{red}}$. Then, the followings are equivalent.*

- **(i)** *$A_{\mathrm{ini}} \leftrightsquigarrow (A_{\mathrm{ini}} \setminus C) \cup \overline{C}$, where $\overline{C}$ is the reverse dicycle of C.*
- **(ii)** *For any arc a in C, there exists an orientation $A \in \mathcal{A}$ such that $A_{\mathrm{ini}} \leftrightsquigarrow A$ and $a \notin A$.*
- **(iii)** *For any $u \in V(C)$, there exists an orientation $A \in \mathcal{A}$ such that $A_{\mathrm{ini}} \leftrightsquigarrow A$ and $2\rho_{A^{\mathrm{blue}}}(u) + \rho_{A^{\mathrm{red}}}(u) \geq 3$.*

Proof. We prove (i)⇒(ii), (ii)⇒(iii), and (iii)⇒(i), respectively.

(i)⇒(ii) If (i) holds, then $A := (A_{\mathrm{ini}} \setminus C) \cup \overline{C}$ satisfies the conditions in (ii), since it contains no arc in C.

(ii)⇒(iii) We prove the contraposition. Assume that (iii) does not hold, that is, there exists a vertex $u \in V(C)$ such that $2\rho_{A^{\mathrm{blue}}}(u) + \rho_{A^{\mathrm{red}}}(u) = 2$ for any $A \in \mathcal{A}$ with $A_{\mathrm{ini}} \leftrightsquigarrow A$. Let a be the arc in C that enters u. Since we cannot reverse the direction of a without violating the feasibility, a is contained in any orientation $A \in \mathcal{A}$ with $A_{\mathrm{ini}} \leftrightsquigarrow A$.

(iii)⇒(i) Suppose that (iii) holds. We take a sequence $A_0, A_1, \ldots, A_l$ of feasible orientations of E such that $A_0 = A_{\mathrm{ini}}$, A_i is obtained from A_{i-1} by reversing an arc $a_i \in A_{i-1}$ for $i \in \{1, 2, \ldots, l\}$, and there exists $u \in V(C)$ such that $2\rho_{A_l^{\mathrm{blue}}}(u) + \rho_{A_l^{\mathrm{red}}}(u) \geq 3$. By taking a minimal sequence with these conditions, we may assume that a_i is not contained in C for $i \in \{1, 2, \ldots, l\}$. Since $2\rho_{A_l^{\mathrm{blue}}}(u) + \rho_{A_l^{\mathrm{red}}}(u) \geq 3$, starting from A_l, we can change the

Algorithm 1 Algorithm for the Reconfiguration Problem.

Input : a graph $G = (V, E)$ and orientations $A_{\mathrm{ini}}, A_{\mathrm{tar}} \in \mathcal{A}$.
Output : "yes" if $A_{\mathrm{ini}} \leftrightsquigarrow A_{\mathrm{tar}}$, and "no" otherwise.

Compute $\mathcal{F}, \phi(A_{\mathrm{ini}})$, and $\phi(A_{\mathrm{tar}})$;
if $\phi(A_{\mathrm{ini}}) \not\underset{\mathcal{F}}{\leftrightsquigarrow} \phi(A_{\mathrm{tar}})$ **then return** "no" ;
if $\phi(A_{\mathrm{ini}}) \neq \phi(A_{\mathrm{tar}})$ *or* $\rho_{A^{\mathrm{red}}_{\mathrm{ini}}} \neq \rho_{A^{\mathrm{red}}_{\mathrm{tar}}}$ **then**
    Compute $A \in \mathcal{A}$ such that $\phi(A) = \phi(A_{\mathrm{tar}})$, $\rho_{A^{\mathrm{red}}} = \rho_{A^{\mathrm{red}}_{\mathrm{tar}}}$, and $A_{\mathrm{ini}} \leftrightsquigarrow A$;
    $A_{\mathrm{ini}} \leftarrow A$; `// See Section 5.2`
while $A^{\mathrm{red}}_{\mathrm{ini}} \setminus A^{\mathrm{red}}_{\mathrm{tar}}$ *contains a dicycle* C **do**
    Take $u \in V(C)$ and solve Problem A;
    if Problem A *has no feasible solution* **then**
        Return "no" ;
    else
        $A_{\mathrm{ini}} \leftarrow (A_{\mathrm{ini}} \setminus C) \cup \overline{C}$; `// See Section 5.3`
return "yes" ;

direction of each arc in C one by one without violating the feasibility, which shows that $A_l \leftrightsquigarrow (A_l \setminus C) \cup \overline{C}$. On the other hand, since $(A_i \setminus C) \cup \overline{C}$ is obtained from $(A_{i-1} \setminus C) \cup \overline{C}$ by reversing a_i for $i \in \{1, 2, \ldots, l\}$, we obtain $(A_{\mathrm{ini}} \setminus C) \cup \overline{C} \leftrightsquigarrow (A_l \setminus C) \cup \overline{C}$. Thus, it holds that $A_{\mathrm{ini}} \leftrightsquigarrow A_l \leftrightsquigarrow (A_l \setminus C) \cup \overline{C} \leftrightsquigarrow (A_{\mathrm{ini}} \setminus C) \cup \overline{C}$. ◀

Let C be a dicycle in $A^{\mathrm{red}}_{\mathrm{ini}} \setminus A^{\mathrm{red}}_{\mathrm{tar}}$. Fix a vertex $u \in V(C)$ and consider the following problem, for which an algorithm is presented later in Section 5.4.

Problem A
Input: A constraint graph G, an orientation $A_{\mathrm{ini}} \in \mathcal{A}$, and a vertex $u \in V(G)$.
Task: Find an orientation $A \in \mathcal{A}$ s.t. $2\rho_{A^{\mathrm{blue}}}(u) + \rho_{A^{\mathrm{red}}}(u) \geq 3$ and $A_{\mathrm{ini}} \leftrightsquigarrow A$ (if exists).

If Problem A has no solution, then condition (iii) in Lemma 13 does not hold. This shows that the condition (ii) in Lemma 13 does not hold, that is, there exists an arc a in C that is contained in any orientation $A \in \mathcal{A}$ with $A_{\mathrm{ini}} \leftrightsquigarrow A$. In this case, since $a \in A_{\mathrm{ini}} \setminus A_{\mathrm{tar}}$, we conclude that $A_{\mathrm{ini}} \not\leftrightsquigarrow A_{\mathrm{tar}}$.

Otherwise, Problem A has a solution, and hence the condition (iii) in Lemma 13 holds. Since it is equivalent to the condition (i) in Lemma 13, we have that $A_{\mathrm{ini}} \leftrightsquigarrow (A_{\mathrm{ini}} \setminus C) \cup \overline{C}$. Therefore, $A_{\mathrm{ini}} \leftrightsquigarrow A_{\mathrm{tar}}$ is equivalent to $(A_{\mathrm{ini}} \setminus C) \cup \overline{C} \leftrightsquigarrow A_{\mathrm{tar}}$, which means that we can regard $(A_{\mathrm{ini}} \setminus C) \cup \overline{C}$ as a new initial configuration instead of A_{ini}. Then, the problem is reduced to the case with smaller $|A_{\mathrm{ini}} \setminus A_{\mathrm{tar}}|$. By applying this procedure at most $O(|E|)$ times repeatedly, we can solve the original reconfiguration problem. The entire algorithm is shown in Algorithm 1.

5.4 Algorithm for Problem A

The remaining task is to give a polynomial time algorithm for Problem A. For this purpose, we use a similar argument to Section 5.2. Suppose we are given a graph $G = (V, E)$ and a vertex $u \in V$. Recall that $X \subseteq V$ is the set of all vertices to which edges in E^{blue} are incident. Define $\mathcal{F}_u$ as the set of all pairs (A^{blue}, d), where A^{blue} is an orientation of E^{blue} and d is a vector in $\{0, 1, 2, 3\}^{X \cup \{u\}}$ satisfying the following conditions:

(1) $2\rho_{A^{\mathrm{blue}}}(v) + d(v) \geq 2$ for any $v \in X \cup \{u\}$.
(2) There exists an orientation A^{red} of E^{red} such that for any $v \in V$,

$$\rho_{A^{\mathrm{red}}}(v) \begin{cases} = d(v) & \text{if } v \in X \cup \{u\} \text{ and } d(v) \in \{0,1,2\}, \\ \geq 3 & \text{if } v \in X \cup \{u\} \text{ and } d(v) = 3, \text{ and} \\ \geq 2 & \text{if } v \in V \setminus (X \cup \{u\}). \end{cases}$$

We note that $|\mathcal{F}_u| \leq 2^{|E^{\mathrm{blue}}|} \cdot 4^{|X \cup \{u\}|} = 2^{O(k)}$. We define $\underset{\mathcal{F}_u}{\longleftrightarrow}$, $\underset{\mathcal{F}_u}{\leftrightsquigarrow}$, and ϕ_u in the same way as $\underset{\mathcal{F}}{\longleftrightarrow}$, $\underset{\mathcal{F}}{\leftrightsquigarrow}$, and ϕ. We obtain the following lemmas in the same way as Lemmas 10, 11, and 12.

▶ **Lemma 14.** *For an orientation A^{blue} of E^{blue} and a vector $d \in \{0,1,2,3\}^X$, we can test whether $(A^{\mathrm{blue}}, d) \in \mathcal{F}_u$ or not in polynomial time.*

▶ **Lemma 15.** *Let $A_{\mathrm{ini}}, A_{\mathrm{tar}} \in \mathcal{A}$. If $A_{\mathrm{ini}} \leftrightsquigarrow A_{\mathrm{tar}}$, then $\phi_u(A_{\mathrm{ini}}) \underset{\mathcal{F}_u}{\leftrightsquigarrow} \phi_u(A_{\mathrm{tar}})$.*

▶ **Lemma 16.** *Let $A_{\mathrm{ini}}, A_{\mathrm{tar}} \in \mathcal{A}$. If $\phi_u(A_{\mathrm{ini}}) \underset{\mathcal{F}_u}{\leftrightsquigarrow} \phi_u(A_{\mathrm{tar}})$, then there exists $A^{\circ}_{\mathrm{tar}} \in \mathcal{A}$ with $\phi_u(A^{\circ}_{\mathrm{tar}}) = \phi_u(A_{\mathrm{tar}})$ such that $A_{\mathrm{ini}} \leftrightsquigarrow A^{\circ}_{\mathrm{tar}}$.*

▶ **Proposition 17.** Problem A *has a solution if and only if there exists a pair $(A^{\mathrm{blue}}, d) \in \mathcal{F}_u$ such that $2\rho_{A^{\mathrm{blue}}}(u) + d(u) \geq 3$ and $\phi_u(A_{\mathrm{ini}}) \underset{\mathcal{F}_u}{\leftrightsquigarrow} (A^{\mathrm{blue}}, d)$.*

Proof. If A is a solution of Problem A, then $\phi_u(A) = (A^{\mathrm{blue}}, d)$ satisfies the conditions by Lemma 15. Conversely, assume that there exists a pair $(A^{\mathrm{blue}}, d) \in \mathcal{F}_u$ such that $2\rho_{A^{\mathrm{blue}}}(u) + d(u) \geq 3$ and $\phi_u(A_{\mathrm{ini}}) \underset{\mathcal{F}_u}{\leftrightsquigarrow} (A^{\mathrm{blue}}, d)$. By Lemma 16, there exists an orientation $A \in \mathcal{A}$ with $\phi_u(A) = (A^{\mathrm{blue}}, d)$ such that $A_{\mathrm{ini}} \leftrightsquigarrow A$. Since $2\rho_{A^{\mathrm{blue}}}(u) + \rho_{A^{\mathrm{red}}}(u) \geq 2\rho_{A^{\mathrm{blue}}}(u) + d(u) \geq 3$, A is a solution of Problem A. ◀

By this proposition, in order to solve Problem A, it suffices to test whether there exists a pair (A^{blue}, d) such that $2\rho_{A^{\mathrm{blue}}}(u) + d(u) \geq 3$ and $\phi_u(A_{\mathrm{ini}}) \underset{\mathcal{F}_u}{\leftrightsquigarrow} (A^{\mathrm{blue}}, d)$. Since $|\mathcal{F}_u| = 2^{O(k)}$, it can be checked in $2^{O(k)} \cdot \mathrm{poly}(|V|)$ time. Note that the elements of $\mathcal{F}_u$ can be computed in $2^{O(k)} \cdot \mathrm{poly}(|V|)$ time by Lemma 14. Thus, Algorithm 1 solves the problem C2C in $2^{O(k)} \cdot \mathrm{poly}(|V|)$ time.

Using similar arguments as in Theorem 8 we can also solve the C2E version.

▶ **Corollary 18.** *C2E parameterized by the number k of blue edges can be solved in time $2^{O(k)} \cdot \mathrm{poly}(|V|)$.*

6 Conclusion

We investigated the parameterized complexity of NCL for four natural parameters related to the constraint graph: The number of AND/OR vertices of an AND/OR graph and the number of red/blue edges of a general constraint graph. We give FPT algorithms for the C2C and C2E version of NCL for each parameter and in particular a linear kernel for NCL parameterized by he number of red edges. An interesting question for future work is whether there is a polynomial kernel for NCL parameterized by the number of OR vertices or the number of blue edges.

References

1. Marthe Bonamy, Marc Heinrich, Takehiro Ito, Yusuke Kobayashi, Haruka Mizuta, Moritz Mühlenthaler, Akira Suzuki, and Kunihiro Wasa. Diameter of colorings under kempe changes. In *Computing and Combinatorics - 25th International Conference, COCOON 2019, Xi'an, China, July 29-31, 2019, Proceedings*, pages 52–64, 2019. doi:10.1007/978-3-030-26176-4_5.
2. Erik D Demaine, David Eppstein, Adam Hesterberg, Kshitij Jain, Anna Lubiw, Ryuhei Uehara, and Yushi Uno. Reconfiguring undirected paths. In *Workshop on Algorithms and Data Structures*, pages 353–365. Springer, 2019.
3. Gary William Flake and Eric B Baum. Rush hour is PSPACE-complete, or "why you should generously tip parking lot attendants". *Theoretical Computer Science*, 270(1-2):895–911, 2002.
4. Arash Haddadan, Takehiro Ito, Amer E Mouawad, Naomi Nishimura, Hirotaka Ono, Akira Suzuki, and Youcef Tebbal. The complexity of dominating set reconfiguration. *Theoretical Computer Science*, 651:37–49, 2016.
5. Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. Complexity of reconfiguration problems for constraint satisfaction. *CoRR*, abs/1812.10629, 2018. arXiv:1812.10629.
6. Weihua He, Ziwen Liu, and Chao Yang. Snowman is pspace-complete. *Theoretical Computer Science*, 677:31–40, 2017.
7. Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1-2):72–96, 2005. doi:10.1016/j.tcs.2005.05.008.
8. Hendrik Jan Hoogeboom, Walter A Kosters, Jan N van Rijn, and Jonathan K Vis. Acyclic constraint logic and games. *ICGA Journal*, 37(1):3–16, 2014.
9. Robin Houston and Willem Heijltjes. Proof equivalence in MLL is PSPACE-complete. *Logical Methods in Computer Science*, 12, 2016.
10. Takehiro Ito, Erik D. Demaine, Nicholas J.A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12–14):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.
11. Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical computer science*, 439:9–15, 2012.
12. Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.
13. Alexander Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer Berlin Heidelberg, 2003.
14. Jan van den Heuvel. The complexity of change. In Simon R Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics 2013*, volume 409. London Mathematical Society Lectures Note Series, 2013.
15. Tom C van der Zanden. Parameterized complexity of graph constraint logic. In *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
16. Tom C van der Zanden and Hans L Bodlaender. PSPACE-completeness of Bloxorz and of games with 2-buttons. In *International Conference on Algorithms and Complexity*, pages 403–415. Springer, 2015.

Approximation Algorithms for Steiner Tree Based on Star Contractions: A Unified View

Radek Hušek
Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic
husek@iuuk.mff.cuni.cz

Dušan Knop
Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Czech Republic
dusan.knop@fit.cvut.cz

Tomáš Masařík
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland
Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Czech Republic
http://tarken.krakonos.org/
masarik@kam.mff.cuni.cz

Abstract

In the Steiner Tree problem, we are given an edge-weighted undirected graph $G = (V, E)$ and a set of terminals $R \subseteq V$. The task is to find a connected subgraph of G containing R and minimizing the sum of weights of its edges. Steiner Tree is well known to be NP-complete and is undoubtedly one of the most studied problems in (applied) computer science.

We observe that many approximation algorithms for Steiner Tree follow a similar scheme (meta-algorithm) and perform (exhaustively) a similar routine which we call *star contraction*. Here, by a star contraction, we mean finding a star-like subgraph in (the metric closure of) the input graph minimizing the ratio of its weight to the number of contained terminals minus one; and contract. It is not hard to see that the well-known MST-approximation seeks the best star to contract among those containing two terminals only. Zelikovsky's approximation algorithm follows a similar workflow, finding the best star among those containing three terminals.

We perform an empirical study of star contractions with the relaxed condition on the number of terminals in each star contraction motivated by a recent result of Dvořák et al. [Parameterized Approximation Schemes for Steiner Trees with Small Number of Steiner Vertices, STACS 2018]. Furthermore, we propose two improvements of Zelikovsky's 11/6-approximation algorithm and we empirically confirm that the quality of the solution returned by any of these is better than the one returned by the former algorithm. However, such an improvement is exchanged for a slower running time (up to a multiplicative factor of the number of terminals).

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Steiner tree, approximation, star contractions, minimum spanning tree

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.16

Related Version A full version [19] of this paper is available at https://arxiv.org/abs/2002.03583.

Supplementary Material
https://github.com/JohnNobody-3af744f30980b7458372/star-contractions

Funding Students were supported by Charles University student grant SVV-2017-260452 and GAUK 1514217.
Dušan Knop: Supported by the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".
Tomáš Masařík: have received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme Grant Agreement 714704.

Acknowledgements We thank the authors of the Boost library [29], which we use in our code, for their work and effective implementation of many graph algorithms. We thank Tomáš Toufar for consultations in the early stages of preparation of the experiments in the paper as well as for the implementation of a part [20] which was partially used in our code. We thank the PACE challenge which was a good motivation and inspiration for the development of practical algorithms for the Steiner Tree problem. The results of this challenge motivated us to develop the experiments presented in this paper. We also thank anonymous referees for interesting feedback as well as for pointing us to several interesting directions for future improvements and related results. Part of the work was carried out while D. Knop and T. Masařík were at the University of Bergen.

1 Introduction

In the Steiner Tree problem an edge weighted graph $G = (V, E)$ is given together with the set of *terminal* vertices $R \subseteq V$; the non-terminal vertices are called *Steiner vertices*. The task is to find a connected subgraph of G containing all terminals and minimizing the sum of weights of its edges. Steiner Tree was among the first problems shown to be NP-complete [22] and is one of the most studied problems in computer science since then.

Steiner Tree	
Input:	A graph $G = (V, E)$, a set of terminals $R \subseteq V$, and a weight function $w\colon E \to \mathbb{N}$.
Solution:	A Steiner tree $F \subseteq G$ containing a path between any two terminals $s, t \in R$.

Steiner Tree is important not only as an interesting graph-theoretic problem, but it has many real-world applications e.g. in network design or VLSI design [21]. Thus, it is extensively studied by both theoreticians and practitioners. Many theoretical results are studying (approximation) algorithms for Steiner Tree; for an overview, see e.g. a survey [21]. We now discuss a few theoretical results important for our work.

Star Contraction and Approximation Algorithms. For an edge-weighted graph $G = (V, E)$ the *metric closure of* G, denoted $\mathrm{mc}(G)$, is the complete graph with the vertex set V with weight of an edge $\{u, v\}$ equal to the length of a shortest path between u and v in G. The most basic approximation algorithm for Steiner Tree is based on finding a minimum spanning tree (MST) in the metric closure of the input graph [25]. Improvements and variants of MST heuristics solving the Steiner tree problem were subsequently examined [9]. An MST heuristic was later improved by Zelikovsky [33] who used the finding of augmenting stars containing three terminals to improve the algorithm of Kou et al. [25] and was the first to beat the barrier of 2 for the approximation ratio. Here an augmenting star consists of a Steiner vertex and exactly three terminals such that if we contract the just defined star into a terminal and compute the weight of an MST, the weight of the thus obtained MST together with the weight of a contracted star is strictly smaller than the weight of the former MST. This approach was later improved by Borchers and Du [6]. Furthermore, the current best theoretical approximation algorithm of Byrka et al. [7] with approximation ratio $\ln(4) + \varepsilon$ is in fact based on star contractions as well.

We observe that the above-mentioned algorithms not only use star contractions as the main tool but, on top of this, most of these algorithms follow a very similar meta-algorithm – see Algorithm 1 and Example 1 below. There, we argue that the simplest algorithm we consider, the MST-approximation, can be described in the framework given in Algorithm 1. We justify the same for (our modification of) Zelikovsky's algorithm in Section 2, Example 2.

Algorithm 1 A unifying high-level framework of selected approximation algorithms for Steiner Tree. The `find_best_star()` function finds a star C with at most k terminals which among all such stars achieves the lowest value under the evaluation function `eval()`. The `contract()` function (usually) contracts the star C and assigns a partial (contracted) solution to S' (note that C and S' could possibly be different e.g. C can be in the metric closure of G').

Input: $G = (V, E)$, set of terminals R, parameters $k, \tau \in \mathbb{N}$, and functions `contract()`, `eval()`, `finish()`
Output: A Steiner tree
1 $G' \leftarrow G,\ R' \leftarrow R,\ S \leftarrow \emptyset$
2 **while** $|R'| > \tau$ **do**
3 $C \leftarrow$ `find_best_star(`G', R', k, `eval)`
4 **if** `eval(`C`)` $< \infty$ **then**
5 $G', R', S' \leftarrow$ `contract(`G', R', C`)`
6 $\mathcal{S} \leftarrow \mathcal{S} \cup \{S'\}$
7 **else break**
8 **return** `finish(`G, $\mathcal{S}$, R`)`

▶ **Example 1** (MST). Observe that in order to find a spanning tree of minimal weight it suffices to find the best star with two terminals, that is, the cheapest edge (between terminals) in the metric closure of the given graph. Furthermore, if we then contract such a path, we reduce the size of the terminal set by one. Thus, one can set the parameter $k = 2$ and $\tau = 1$ (as we perform star contractions exhaustively). The `eval()` function gives the length of the shortest path (i.e., the total weight of the proposed 2-star), the `contract()` function contracts, and the `finish()` function contracts the given collection of 2-stars.

Unbounded Size of the Best Star. A recent result of Dvořák et al. [15], which proposes a novel algorithm in the framework of parameterized approximations, also falls in the framework suggested in Algorithm 1. Surprisingly, their algorithm uses an unbounded value of the parameter k, the number of terminals in the best star. Their algorithm, given a parameter p and the desired approximation ratio $\varepsilon > 0$, runs in time $f(p, \varepsilon)\,\mathrm{poly}(|G|)$ and outputs a solution of cost at most $(1 + \varepsilon) \cdot \mathrm{OPT}(p)$, where $\mathrm{OPT}(p)$ is the value of an optimal solution that uses at most p Steiner vertices. Let us now discuss in more detail why the algorithm of Dvořák et al. follows the proposed framework; we discuss further technical details later (see Section 2). First we set the parameters $k = \infty$ and $\tau = c \cdot \frac{p^2}{\varepsilon^4}$ for a suitable constant c.

`eval()` Let C be a connected subgraph of G', let $w(C)$ be the total weight of edges in C, and let $R_C \subseteq R'$ be the set of terminals contained in C. The function `eval(`C`)` returns the value $\frac{w(C)}{|R_C|-1}$.

`find_best_star()` Since the parameter $k = \infty$, the function returns a connected subgraph C minimizing the value $\frac{w(C)}{|R_C|-1}$ among all connected subgraphs with at least two terminals.

`finish()` We first contract all subgraphs C obtained so far (i.e., we construct the graph G'). Then the algorithm of Fuchs et al. [16] is invoked on G'.

It is worth noting that the algorithm of Fuchs et al. computes an optimal solution in time $f(\tau) \cdot \mathrm{poly}(|G'|)$. Note that a similar running time has been achieved already by Dreyfus and Wagner [12]. We conclude that the best-star contraction is a popular and successful technique in the design of approximation algorithms for Steiner Tree. We refer to the work of Chimani and Woste 2011 [8] for an experimental comparison of contraction-based techniques known at that time. Recently, Beyer and Chimani 2019 [4] conducted another experimental study, where they compare approximation algorithms with approximation ratio

better than 2. The underlying technique within the compared algorithms (greedy as well as linear programming based approaches) is the contraction of *k-restricted full components*, components of at most k terminals where the set of leaves and terminals coincide, for some $k \geq 3$. For most of the considered algorithms, their strongest theoretical approximation bounds are only achieved for $k \to \infty$, but also, the running time is exponentially dependent on k, which makes it infeasible in practice.

Interestingly, in [8] conclude that the simplest and oldest algorithm with the weakest theoretical guarantee among the algorithms they considered has the best performance in practice. This was the already mentioned 11/6-approximation algorithm by Zelikovky [33] that we are considering in our study. They speculated in the conclusions that the reason might be that the larger values of k might not be feasible to compute in practice (A large k is also identified as the main practical obstacle in [4]). Instead, Chimani and Woste suggested that some clever algorithmic choices might help to overcome this issue in the future. In this light, we believe that our approach of best stars contractions might give such guidance. We are posing the following natural questions:

1. Do the star contractions behave well in practice? Specifically, is it possible to improve the total weight of a solution returned by well-studied heuristics (e.g. MST approximation) significantly when we first perform a few rounds of best star contractions?
2. Is it common to find large (nearly) best stars? That is, is there a significant fraction of all contracted stars containing more than e.g. 5 terminals?
3. Is there any point of Pareto optimality? For example, is it possible to reduce the number of terminals by 20% using only 10% of the total work? Ideally, while also improving the cost of the solution by 80%? Here 100% improvement is represented by performing best star contractions until only a single terminal is left?

Our Experiment. In what follows, we refer to Algorithm 1. We run the algorithm in steps – invocations of the while-loop – and each time we call the `contract()` function. We find a solution using all of the aforementioned methods. We collect the data and, e.g., for MST on public instances from PACE Challenge 2018 we aggregate statistics (see Figure 2). Furthermore, we measure the sizes of contracted stars and the performance of `find_best_star()`, since this is the most time-consuming step in the algorithm. It is worth pointing out that we implement some standard heuristics (see Section 2.1) which we use to preprocess the input, that is, all our data is collected on the already preprocessed instances.

Dataset. We evaluate the algorithm and present our results for the set of public instances of PACE Challenge 2018 in Section 3. According to the report from PACE Challenge 2018 [5], the set of instances in Track C consists of the hardest instances of Steinlib and from real-world telecommunication networks by Ivana Ljubic's group at the University of Vienna. It should be noted that similar sources were used for the DIMACS Challenge [1]. By that time, a majority of the selected instances cannot be solved within one hour by the state-of-the-art program and in several cases, the actual optimum was unknown. For more discussion about the chosen dataset, please consult the report from the PACE Challenge 2018 [5]. Furthermore, we evaluate our experiments on rectilinear instances from ORLib [3] in the full version of the paper.

Preliminaries. We give a brief recapitulation of graph theory terminology used in this work; for the basic notation, we refer the reader to monographs [26, 10]. All graphs are undirected without loops and multiple edges. If we argue about algorithmic complexity of a certain routine or the amount of memory needed in order to store some data for a graph G, by n we denote the number of vertices of G and by m we denote the number of edges of G. For

a graph $G = (V, E)$ and an edge $e = \{u, v\}$ if we *contract* e (denoted as G/e), we create a new graph with the vertex set $(V \setminus \{u, v\}) \cup \{z\}$, where z is a newly introduced vertex. The edge set of the resulting graph consists of all edges in E not incident to any end-vertex of e together with the newly introduced edges $\{w, z\}$ for every edge $\{w, u\}$ as well as for every edge $\{w, v\}$, where the weight of a newly created edge is the same as the weight of the edge $\{w, u\}$, $\{w, v\}$, respectively. Note that if the above operation is about to create multiple edges we simply keep the one with a lower weight. For a graph $G = (V, E)$ and a vertex v with exactly two neighbors in G by *suppressing* v we mean changing G into a new graph as follows. The new vertex set is $V \setminus \{v\}$ and we delete all edges incident to v. Finally, we insert edge $\{x, y\}$ if both $\{x, v\}, \{v, y\} \in E$ with weight $w(\{x, y\}) = w(\{x, v\}) + w(\{v, y\})$.

1.1 More Details on Past Implementation Challenges

Since Steiner Tree has many applications, it received attention among practitioners and in operations research. One particular example can be e.g. the specialized module SCIP-Jack [17] in the SCIP tool for solving (mixed) integer linear programs. In the 11$^{\text{th}}$ DIMACS Implementation Challenge [1] various variants of Steiner Tree formed the central topic of the challenge. Among others, e.g. the basic version Steiner Tree, geometrical versions (e.g., rectilinear instances), and prize-collecting variants were tackled. One can read in the description of the DIMACS Implementation Challenge:

> *DIMACS Implementation Challenges address questions of determining realistic algorithm performance where worst-case analysis is overly pessimistic and probabilistic models are too unrealistic: experimentation can provide guides to realistic algorithm performance where analysis fails.*

Last but not least, one track of the PACE Challenge 2018 [5, 2] was completely devoted to Steiner Tree with three specialized branches. In PACE Challenge 2018 there was an approximation branch and two exact branches – one with the additional promise of a small number of terminals and in the other, a tree-decomposition of the input graph of small tree-width was given.

2 Implementation Details and Heuristics

In this section, we describe our implementation and improvements of finding the Best Star as proposed in [15] (Section 2.2), the approximate algorithms used to finish the solution after the application of the Best Star Algorithm (Section 2.3), and also the heuristics we used to preprocess the instances (Section 2.1). Last but not least, we discuss a few simple yet in practice well-performing modification of Zelikovsky's algorithm (Section 2.3).

2.1 Heuristics

All heuristics we used are deterministic and ensure that the optimal value of instance before and after applying them is the same. Namely, we used the following well-known ones:

1. We contract all edges e with $w(e) = 0$ at the very beginning. Thus we assume in the rest that all weights are positive.
2. We remove all Steiner vertices of degree 1 and suppress Steiner vertices of degree 2.
3. We contract the edges incident to terminals of degree 1.
4. Contract an edge $e = \{s, t\}$ between two terminals if $w(e)$ is minimal among the edges incident to s or t.
5. *Shortest Path Test (SPT):* Delete an edge $e = \{u, v\}$ if $w(e)$ exceeds the length of the shortest path between u and v.
6. *Terminal Distance Test (TDT):* see below.

Algorithm 2 Pseudocode of the used preprocessing.

```
1  Function quick_heuristics(G):
2      run ← true
3      while run do
4          run ← false
5          run ← contract_zero_edges(G)
6          run ← run ∨ delete_degree_one_Steiner(G)
7          run ← run ∨ contract_the_only_edge_incident_to_term(G)
8          run ← run ∨ contract_the_cheapest_edge_between_two_terminals(G)
9  Function preprocessing(G):
10     quick_heuristics(G)
11     SPT(G)
12     quick_heuristics(G)
13     TDT(G)
14     quick_heuristics(G)
15     SPT(G)
16     quick_heuristics(G)
```

Heuristics (1) up to (4) are implemented using straightforward iteration over all edges or vertices of the graph, and if any of them succeeds, we again rerun all of these. This iteration blows up theoretical time complexity by a factor of n but in practice, only very few iterations yield an irreducible instance. In Algorithm 2 we encapsulate these into the `quick_heuristics()` function. For performance reasons, we split the SPT heuristic into two parts: first, it checks only paths consisting of two edges which is quite efficient ($O(n^2)$ because we store edges incident to every vertex in sorted order) and then we check paths of all lengths which requires to run Dijkstra's algorithm [11] from every vertex and is therefore much slower.

It is worth noting that both SPT and TDT can benefit from rerunning but due to their time complexity and usually lower number of improvements, we do not use these heuristics exhaustively. Instead, we run SPT, TDT, and SPT once more and execute `quick_heuristics()` at the beginning, between them, and at the end. See the `preprocessing()` function in Algorithm 2.

Terminal Distance Test. The TDT heuristic was introduced in [30]. The basic idea is the following: Let (W, W') be a partition of vertices such that both W and W' contain some terminal and each of them is connected, let be e and f the shortest and the second shortest edge of the cut induced by (W, W'). If there is a path connecting some terminal $t \in W \cap T$ and $t' \in W' \cap T$ which uses e and is no longer that f then there exists an optimal solution which uses edge e. Observe that while it is easy to verify the correctness of this heuristic, it does not give us directly an effective algorithm. Koch and Martin [23] point out that it is possible to implement TDT in time $O(|V|^3)$. Furthermore, they claim that the time complexity can be further improved to $O(|V|^2)$ as is described in the thesis [13]. However, we were unable to access the thesis.[1] Consequently, we describe our implementation in Algorithm 3 for the future reference.

[1] We thank to an anonymous referee for providing us with a reference [14] to an implementation of TDT heuristic running in time $O(|E| + |V| \log(|V|))$. It will be a part of the planned improvements in our experiments.

Our implementation is based on the data structure for dynamic edge 2-connectivity of Westbrook and Tarjan [32]. This structure has amortized time complexity $O(\alpha(m))$ per operation and supports edge addition and check whether two vertices belong to the same (2-connected) component. The time complexity of Algorithm 3 is $O(nm \log n)$ when Dijkstra's algorithm is implemented with d-regular heap.

Algorithm 3 Implementation of Terminal Distance Test.

```
1  Function test_edge(G, e, f, X):
2      u, v ← e
3      Remove e from G
4      t_1 ← terminal closest to u                    // Dijkstra's algorithm
5      t_2 ← terminal closest to v
6      if d(t_1, u) + w(e) + d(v, t_2) ≤ w(f) then
7          X ← X ∪ e
8      Add e back to G
9  Function TDT(G):
10     E' ← sort(E(G), w(a) < w(b))
11     C ← structure for 2-connectivity
12     X ← ∅
13     foreach e ∈ E' do
14         B ← add_edge(C, e)   // returns edges which were bridges before addition
           of e but no longer are
15         foreach b ∈ B do
16             test_edge(G, b, e, X)
17     Buy edges in X
```

2.2 Finding the Best Star

As already mentioned in Section 1, Algorithm 1 repeatedly finds and contracts a so called best star (according to a certain ratio) in G. Here, we first give the definition used by Dvořák et al. [15]. Later, we discuss a further additional practical extension of the former definition and describe our implementation in detail. Let G be a graph and R the set of terminals in G. For a vertex c and a set $R' \subseteq R$ we define a *star* (centered at c and a terminals set R') which we denote $\mathrm{st}(c, R')$. We define a *ratio of star* $\mathrm{st}(c, R')$ as

$$\mathrm{r}\left(\mathrm{st}(c, R')\right) = \frac{\sum_{t \in R'} \mathrm{dist}(c, t)}{|R'| - 1},$$

where $|R'| \geq 2$ and $\mathrm{dist}(c, t)$ is the weight of the edge $\{c, t\}$ in $\mathrm{mc}(G)$. The best ratio achievable for a star centered at c is $\mathrm{r}(c) = \min_{R' \subseteq R, |R'| \geq 2} \mathrm{r}(\mathrm{st}(c, R'))$ and a *best star centered at* c is any minimizer of the defining expression. The *best star in the graph* G is any star $\mathrm{st}(c, R')$ minimizing $\mathrm{r}(G) = \min_{c \in V} \mathrm{r}(c)$.

Clearly, the smaller the ratio is the better. On the other hand, in our experiments, there were many ties and thus we further extend this definition in the case there are more minimizers of the best ratio in G. We introduce a second measurement taking into account the number of terminals contained in a star. Intuitively, the more terminals it contains the better. Thus the *best star in the graph* G is any star $\mathrm{st}(c, R')$ with ratio $\min_{c \in V} \mathrm{r}(c)$ which maximizes $|R'|$.

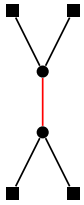


Figure 1 Simple graph containing four terminals (represented by squares) and two Steiner vertices (discs). Suppose all the edges have unit weight. When computing the weight of a star centered at any of the two Steiner vertices using the metric closure, we count the weight of the red edge twice. As a consequence, one gets two possible stars with ratio 2 – one containing only the two terminals connected directly to the assumed Steiner vertex and the other containing all four terminals. However, the second described star should better have a ratio of 5/3 (which is better than 2).

It is worth noting that if the best star with center c contains k terminals, then it contains k terminals that are closest to c in $\mathrm{mc}(G)$ [15, Lemma 6]. Even though the definition of the best star uses $\mathrm{mc}(G)$, in practice, we cannot afford to compute and store it due to its size (quadratic in n). Instead, we utilize Dijkstra's algorithm [11] to compute the best star centered in a given vertex. In total, we obtain running time $O(mn + n^2 \log n)$ per one round, that is, for one execution of the main loop in Algorithm 1. Furthermore, we use several heuristics to improve the running time significantly in practice. These heuristics employ memorization and early termination, among others; refer to Algorithm 4. Let $\mathrm{r}^{\mathrm{cur}}(G)$ denote the best so-far computed ratio, i.e., the best ratio among all already computed stars. By slightly overloading the notation when searching for the best star centered at c we let $\mathrm{r}^{\mathrm{cur}}(c)$ denote the ratio of the best so far computed star centered at c. We use this notation to describe practical heuristic improvements:

1. We stop the execution of Dijkstra's algorithm when the current distance from the source (center of the star) is strictly greater than $\mathrm{r}^{\mathrm{cur}}(c)$.
2. For every vertex $c \in V$ we store the best star centered at c in between the rounds.
3. We (re)compute the best star at c only if the stored one could have been affected by a star-contraction performed in the previous round. We can do this since a single star-contraction affects only a small (local) part of the graph.

Overestimating Star Weight. It is worth noting that the best star as described in [15] is a star in metric closure containing some number of terminals closest to its center (and minimizes the star ratio). Note that this clearly may overestimate the weight of such a star as well as its ratio (see Figure 1). Observe that if the best found star contains at most three terminals, then its weight is always estimated correctly. While this estimate is sufficient for the purpose of theoretical analysis, it may affect the overall behavior of the algorithm. We would like to point out that in our implementation of a star-contraction ($\mathrm{st}(c, R')$) we contract edges of an MST containing R' instead of contracting the star itself. Clearly, this modification can only decrease the cost of the single round. On the other hand, it is not clear how this "greedy" improvement affects the overall performance of the whole process. We propose a way to overcome the overcounting issue.

Improved Stars. Using Dijkstra's algorithm, we are recursively searching for a new terminal within the threshold distance and building the best star for each vertex. We start by setting the given vertex as the origin of Dijkstra's algorithm. Whenever we encounter a terminal

such that it forms a better star together with the so-far best star originating in the given vertex, we add it to the constructed star and start over while setting the whole star as the new origin for Dijkstra's algorithm.

Algorithm 4 A pseudocode for The Best Star function.

```
1  Function find_best_star_v(G,R,v,r):
2      weight ← 0                                          // sum of distances
3      nter ← 0                                            // current number of terminals
4      while (w,d) ← dijkstra_next(G,v) do
5          if nter ≥ 2 ∧ weight/(nter − 1) < d then return weight/(nter − 1)
6          if d > 2r then return lbound(d)
7          if w ∈ R then
8              weight ← weight + d
9              nter ← nter + 1
10 Function find_best_star(G,R):
11     r^cur(G) ← ∞
12     foreach v ∈ V do ratio[v] = ∞
13     foreach v ∈ V do
14         if invalid(v) ∨ (ratio[v] < r^cur(G) ∧ lbound(ratio[v])) then
15             ratio[v] ← find_best_star_v(G,R,v,r^cur(G))
16         if r^cur(G) > ratio[v] then
17             r^cur(G) ← ratio[v]
18             star_center ← v
19     return (star_center, r^cur(G))
```

2.3 Finishing the partial solution

The original algorithm of Dvořák et al. [15] performs star contractions until the number of terminals decreases under a threshold depending only on the desired approximation ratio ε and the number of Steiner vertices in (some) optimal solution. An exact algorithm is used to complete the solution when the number of terminals dropped below the threshold. While this is a very natural theoretical approach, it is not suitable for practical use for the following reasons:

1. Both discussed exact algorithms are based on the dynamic programming, which makes them quite slow in practice (mostly intractable for instances with more than 20 terminals).
2. The threshold depends on the number of Steiner vertices in (some) optimal solution and we, in general, have no good upper bound on it.

Instead, for the purposes of the evaluation, we choose the following algorithms which we run after every contraction:

- **MST:** The usual well-known minimum spanning tree approximation – we take a subgraph of the metric closure induced by terminals and find its minimum spanning tree. Note that the implementation does not compute whole metric closure but computes Voronoi regions of the terminals instead [28].. Then an auxiliary graph is constructed using terminals of the original graph as vertices and adding an edge for every edge $\{u, v\}$ crossing between Voronoi regions with length $d(u) + w(u, v) + d(v)$, where $w(\cdot, \cdot)$ is the length of an edge and $d(\cdot)$ is the distance to a closest terminal.

- **MST+:** We calculate MST and then improve its solution by taking its terminals and branching vertices (Steiner vertices with the degree at least 3 in the solution), marking them all as terminals and running MST on this modified instance. It is easy to see that solution of MST on this modified instance is never worse than the solution we began with because the original solution is a spanning tree of the modified instance. We repeat this while the solution is improving.
- **Zelikovsky:** Zelikovsky's algorithm [33], which augments the MST solution using stars with 3 terminals, was the first algorithm with a better approximation ratio than 2. The algorithm proceeds in rounds. Each round it looks at all 3-stars, selects the star s which maximizes the so-called "win" which is $mst(G) - mst(G/s) - d(s)$ (where $mst(G)$ is the weight of MST solution of instance G, G/s denotes G after contraction of terminals in s and $d(s)$ is the weight of star s), and if the win of the best star is strictly positive, it contracts it and starts another round. Otherwise, it stops, and returns MST on all terminals and selected star centers. (Computing the MST at the end is needed to ensure that the solution is really a tree.) The original version of the algorithm computes the best center for every triple of terminal and weight of such a star at the very beginning and runs in $O(n(m + n \log n + t^2) + t^4)$ time and requires $O(m + t^3)$ extra space.
- **Zelikovsky−:** This modification recomputes distances of triples in each round instead of precomputing them in advance. Unlike the usual Zelikovsky's algorithm where the triplets are only upper bounds, here we have optimal values in each round. This algorithm is slower ($O(nt(m + n \log n + t^2))$), but the memory requirement is $O(m)$ smaller.
- **Zelikovsky+:** This modification differs from Zelikovsky− only by the application of MST+ instead of MST at the end.

▶ **Example 2** (Zelikovsky-)**.** The Zelikovsky's algorithm also fits into the star-meta-algorithm framework described in the introduction: The parameters are $\tau = 2$ and $k = 3$, `eval()` function returns $-win$ (or ∞ for nonpositive win), the `contract()` just contracts the star and adds center of the star to S, and the `finish()` computes MST of $R \cup S$. Note that Zelikovsky− variant is more natural as it finds the best star each round, whereas the original Zelikovky's algorithm precomputes values of all stars with 3 terminals.

2.4 Comparison with state-of-the-art results

To put our study in the context, we compare its behavior with other programs competing in the PACE Challenge 2018. We describe the comparison system as was proposed for the PACE challenge 2018 [5]: A time-limit to output a solution was set to 30 minutes – we call this a run. Afterward, each run received points according to the fraction of the value of the returned solution to the best solution known. The results are aggregated over all instances, see Table 1. The implementation of best star contractions ended up at the 4th place in this comparison. It is important to note that the algorithm was enhanced with local search heuristics. We give a brief description of the particular implementation considered in the comparison (see [20] for the detailed description and the implementation).

After the initial heuristics described in Section 2.1 were exhausted, the program spends exactly 10 minutes performing the best star contraction algorithm. The rest of the available time was spent on the local search heuristics whose description follows.

Details on Local Search Heuristics. We implemented the two following randomized local search heuristic. Those heuristics run until the dedicated time was up. Then the best solution was returned. As the second heuristic is much more time consuming than the first one we run it only sparsely.

Table 1 An aggregated comparison of the overall performance of the algorithms participating in PACE Challenge 2018 Track C. We exchange the names of the teams with a summarizing name of the main method they used. For more implementation details of their algorithms, see the report [5] that also contains links to individual implementations and their comprehensive descriptions.
**Shortest Path Heuristic*: Pick one terminal as a root and repeat the following: Find the closest terminal to the root and contract the shortest path from this terminal to the root.

Algorithm	Score
Evolution Algorithm	99.91
MIP solver + Heuristics (SCIP-Jack [24])	99.89
Iterated Local Search	99.78
Best Star Contractions with Local Search [20]	**99.70**
Zelikovsky [33]	98.93
Simulated Annealing	98.27
Random Generation + Local Search	97.54
Shortest Path Heuristic*+ Local Search	97.15
Mehlhorn 2-approximation [27] + Watel and Weisser k-Approximation for the directed Steiner Tree Problem [31]	96.92
Shortest Path Heuristic*	94.57
Primal-Dual 2-approximation + Local Search	94.37
Contract Random 2-terminal Shortest Path + MST	82.61
Ant Colony Optimization	80.73

- **Local search using MST+-approximation.** We randomly select a couple of additional Steiner vertices to be added to the branching Steiner vertices of the current solution. Then we run the MST+ algorithm on them.
- **Local search using Dreyfus-Wagner partition.** Inspired by Dreyfus-Wagner FPT algorithm [12] we derive a heuristic that obtains the partition of the vertices given one of the promising solutions and then computes an optimal Steiner tree on this structure.

3 Outcomes of our Experiments

In this section, we perform our main tests that are carried out on the instances from the PACE Challenge 2018, Track C.[2][3] Aggregated data from the measurements are provided in Figure 2 and the corresponding data in the full version. There, star contractions are executed in rounds and in every round, all heuristics (from Section 2.3) are executed so

[2] It is worth noting that all the data, as well as the corresponding charts for all instances, are available in the repository containing our implementation code `https://github.com/JohnNobody-3af744f30980b7458372/star-contractions`.

[3] Due to time constraints the tests involving Zelikovski's algorithm were performed only on 123 out of 200 instances from the PACE Challenge 2018. The list of instances used in each experiment, as well as all the results and charts, are available in the above-mentioned git repository. The problem was that running Zelikovsky's algorithm after each best star contraction took too long on large instances. We provide figures where tests (excluding Zelikovsky's algorithm) were performed on almost all instances (excluding instances number 193, 196, 197, and 198 only) in the full version. Those four instances were still too large, even for the rest of the tests. We stress out that the outcomes of those tests are relatively similar to those on the limited number of instances.

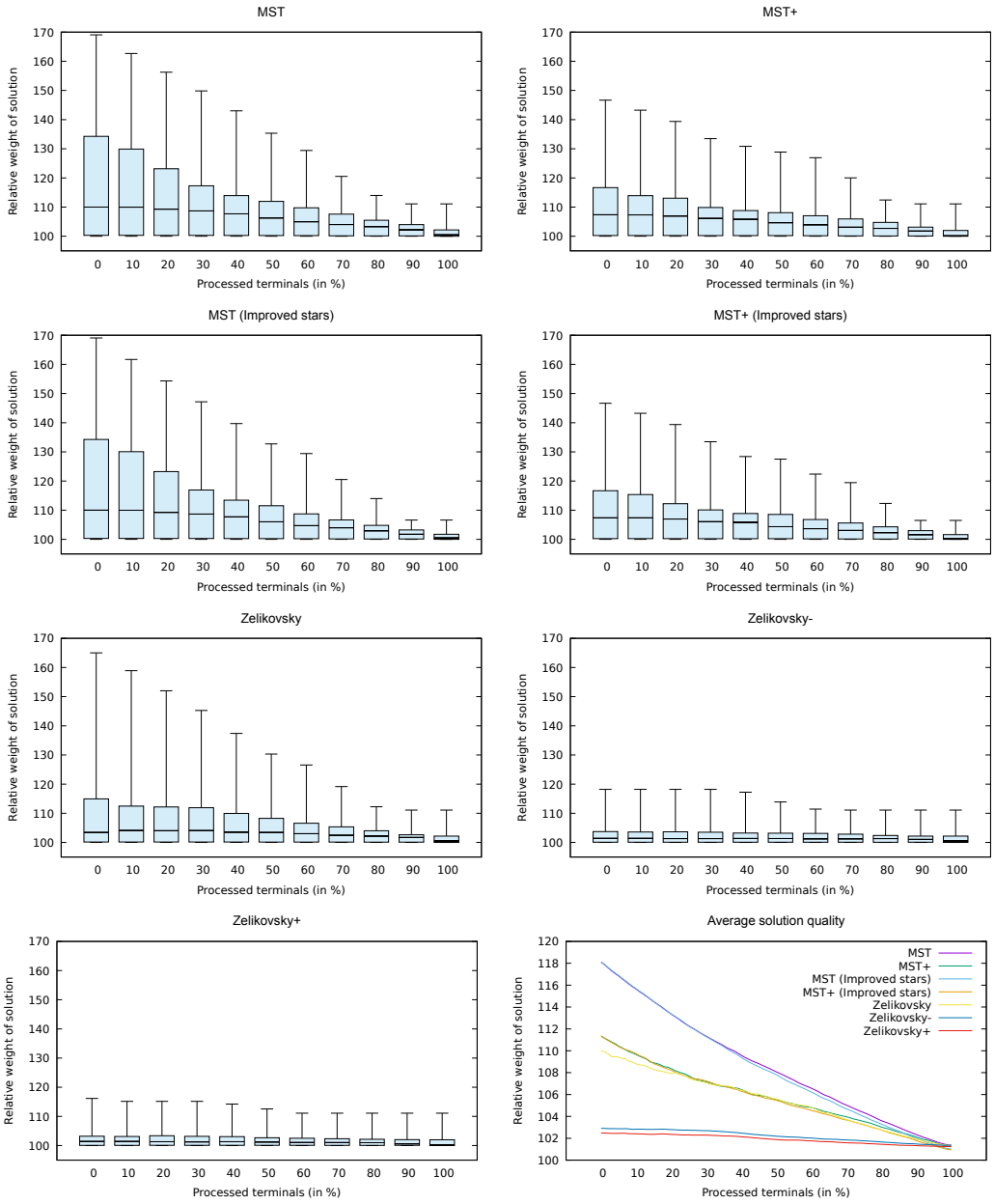


Figure 2 This chart shows the performance comparison of star contractions and MST heuristics on PACE Challenge 2018 instances. The x-axis represents the number of star contractions in percent before MST was computed. The zero value is MST heuristics after preprocessing only. Hundred denotes a result obtained by star contractions till one vertex remains in the graph. The y-axis represents the quality of the solution again in percentage where the hundred is the best solution we obtained during our experiments (not only in this comparison). As we pointed out, the best solution we are comparing to was derived using a local search algorithm, so optima are represented by more or less the current state-of-the-art results. The top line is the maximum in our dataset, the colored box represents data points from the first to the third quartile with the line in the middle denoting the median, and the line at the bottom is the minimum. The last plot shows arithmetic averages of the same data combined into a single plot for easier comparison.

Table 2 The total number of stars containing two to ten terminals contracted during the execution of the algorithm on all PACE Challenge 2018 instances.

# of terminals	2	3	4	5	6	7	8	9	10	> 10
# of basic stars	99965	23921	1721	683	246	135	197	126	149	1158
# of improved stars	91957	15346	4371	2070	969	539	410	263	218	1285

that the overall performance can be compared. Experiments are done separately for basic stars and only some of them are repeated for improved stars. An important thing to note is that the best solution for each input was derived out of the best of outcomes from all the performed experiments, including additional local search heuristics described in Subsection 2.4. Therefore, it represents more or less the current state-of-the-art. Also, "the worst" solution was obtained using the heuristics described in Section 2.1.

The basic outcome of our experiments is the chart for MST. It shows that the best star algorithm significantly improves the performance of an MST approximation. Here, the star contractions help to improve the quality of some solutions for more than 57%. The improvement is by more than 12% on average and it is worth pointing out that the number outliers is reduced significantly as well (see Figure 2) and thus we answered Question 1 positively.

We conclude that MST+ could replace MST for all purposes where a slow-down by a small multiplicative constant does not play a significant role. MST+ is not only better by definition, but also our experiments suggest that it outperforms MST quite significantly. In addition, it is still quite simple to code. Most importantly, Our measurements declare that it is approximately only 3 times slower than MST. This means it is (most of the time) negligible in practice since MST is computed within seconds on the current inputs. This is also supported by the fact that the computation of MST/MST+ takes only a fraction of the algorithms considered in this study. Moreover, star contraction combines very well with MST+ which improves not only the overall performance but, more importantly, a good solution is obtained much sooner. Besides, MST+ combines well with the local search algorithm presented in Subsection 2.4.

We answer Question 2 negatively. A vast majority of the contractions performed by Algorithm 1 on our instances are those of stars containing two or three terminals (see also Table 2 and Figure 3. It is worth noting that the best star containing only two terminals is found (on average) in more than 75% of all contractions performed during the execution of the algorithm.

As we have already observed, if during the algorithm's execution we only contract stars containing only two terminals, then the proposed algorithm returns a minimal spanning tree in the metric closure of the graph on terminals. Therefore, one should expect that if stars contracting more terminals are found and contracted during the execution, then the quality of the solution found should improve. Clearly, improved stars allow us to identify best stars containing more terminals; see Figure 3 and Table 2. We can see that the number of best stars containing more than 5 terminals increase from (roughly) 2% to 5%. Despite this improvement, overall performance is not much better than using the regular star contractions. However, the time consumption stays low (approximately three times as much, see Figure 4). In addition, improved stars improve the final solution quite significantly. They even cooperate well in combination with MST+. This is far the best method we studied when aiming at the smallest solution as it differs from the "best" solution by at most 6.49% and in the median by only 0.11%.

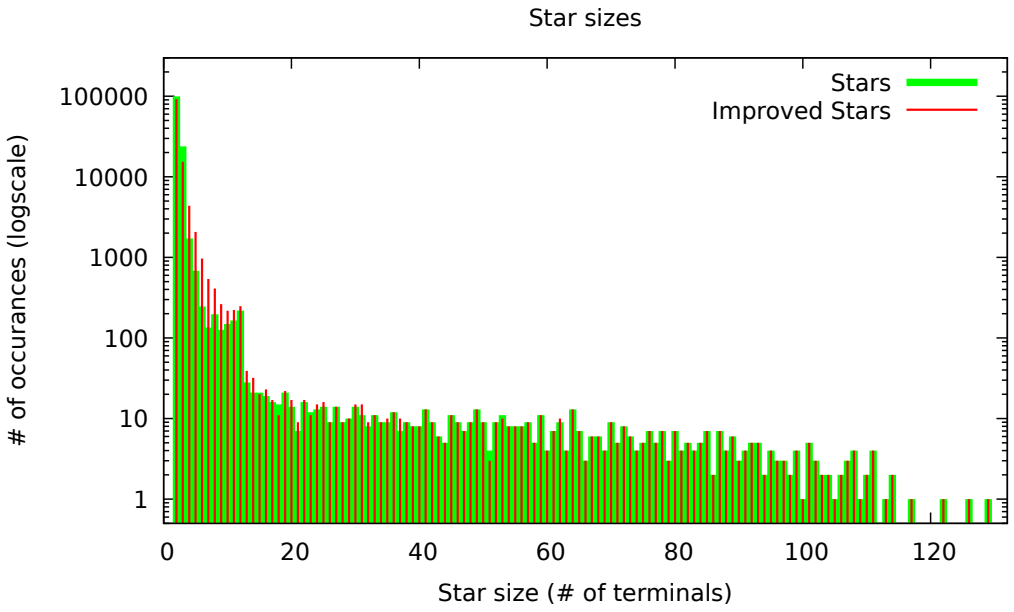


Figure 3 Star sizes on PACE Challenge 2018 instances with star contractions running until the end.

Despite our former beliefs that were supported by the results of Dvořák et al. [15], the answer to Question 3 seems to be also negative. Figure 5 indicates that there is no apparent threshold point for neither classical nor improved stars.

Driven by our experiments, we propose two variants of modifications of Zelikovsky's algorithm: Zelikovsky+, Zelikovsky-. These are based mainly on our insight that takes advantage of reformulating known algorithms in terms of star contractions. The key idea is to compute stars after each contraction as opposed to precomputing them. Of course, since we recompute a star to contract, the proposed variants are slower (roughly 3 times). However, they both achieve much better performance (i.e., the total weight of the solution found); see Figure 2. As it follows from the paragraph above, relaxed star contractions are not helping much since there are not so many large stars. A large improvement is achieved by combining it with the MST+ algorithm. Consult Figures 2 and 4 where one can compare MST+ with MST and our modification of Zelikovsky's algortihm. Surprisingly Star Contractions improve even the performance of the classical implementation of Zelikovsky's algorithm. However, this is easily outperformed by improved stars combined with the MST+ algorithm. On the other hand, our variants Zelikovsky− and Zelikovsky+ behaves reasonably well from the beginning and it seems that stars contractions have only a little to do with it. For example, Zelikovsky+ has a median that is only 1.04% worse even without any star contractions compared to 3.59% (for MST+). This good performance even without any star contractions is diminished by a slower running time which is comparable with many rounds of star contractions finished by MST+.

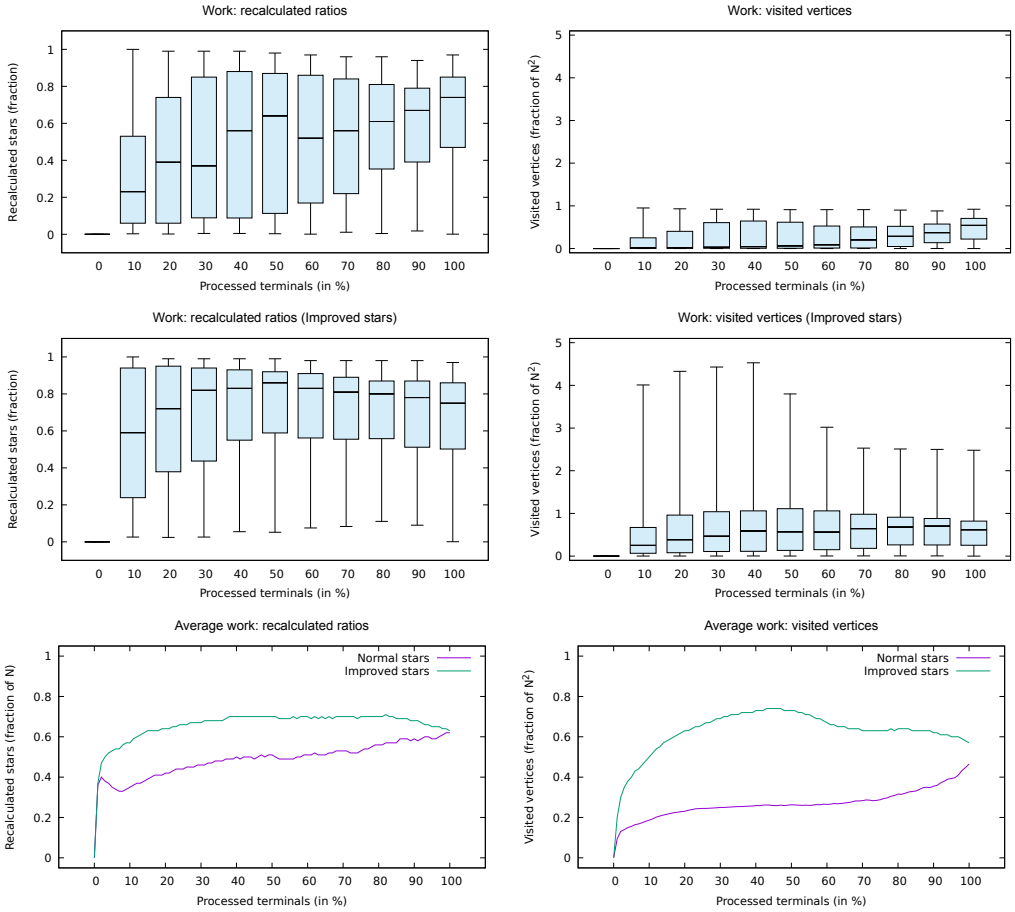


Figure 4 Work done on PACE Challenge 2018 instances.

4 Conclusions

In general, we have confirmed that contracting the best star improves the quality of the solution returned by the MST (MST+) algorithm. It seems that if we exhaustively apply contractions of best stars, we achieve a solution of slightly better quality than our modification of Zelikovsky's algorithm (i.e., Zelikovsky+ algorithm applied directly to the input). However, the running time of such approaches is comparable in practice. Unfortunately, unlike in classical Zelikovsky's algorithm, star contractions do not significantly help in our modifications. Importantly, MST+ heuristics should replace the classical MST since it outperforms it (by definition) without being much more complicated or time-consuming. Improved stars with MST+ do perform better when aiming for the best quality of the solution. They should be used whenever the slight increase in the running time is not important.

Future Work. Last but not least, our experiments suggest that our methods for lessening the time needed to compute the best star are useless when there are only a few terminals left in the graph since in such a case the computation is only local. Yet if this happens (according to Figure 4 this happens when about 30% of terminals are left), it is not possible to use the algorithm of Dreyfus and Wagner, since in such cases we still usually have more

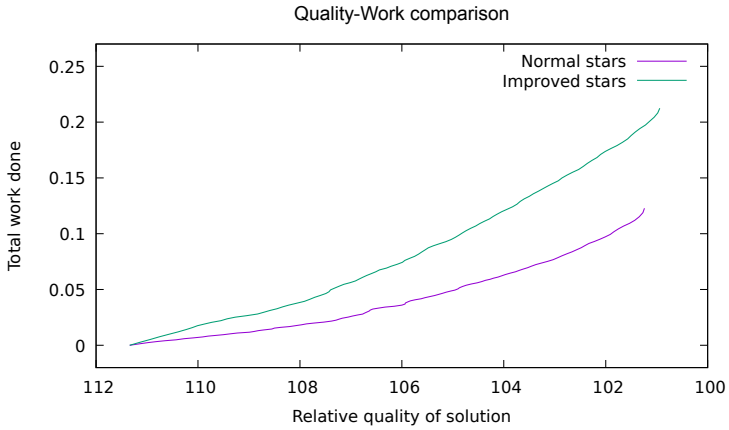


Figure 5 Work needed to get a solution of given quality using star contractions and MST+. (Y-axis is fraction of total maximum work done and work is measured as number of visited vertices during invocations of Dijkstra's algorithm, the work done by MST+ is negligible and hence ignored.)

than 30 terminals left—which is clearly intractable for larger instances. This brings us to the following question: Is it possible to use best stars to speed up (in both theory or practice) the algorithm of Dreyfus and Wagner while loosing only a bit in its precision? If yes, we hope that a suitable combination of the two algorithms can be used in practice. One can use recent improvements of Dreyfus and Wagner algorithm with the same worst-case running time but which behaves significantly well in practice on instances originated from VLSI design [18].

An interesting research direction is to augment the algorithm of Dvořák et al. [15] for Euclidean instances, since solutions to such instances should contain fewer Steiner vertices and thus the quality of the returned solution should increase. In a similar direction, we performed several basic tests for rectilinear instances from ORlib (see the full version). Interestingly, our approach works reasonably well on such specialized instances, significantly better than on general instances. However, we leave it for future work as the comparison with specialized heuristics for those instances is essential.

Yet another possibility is to, instead of contracting a subgraph with the best ratio, contract a subgraph with a slightly worse ratio which contains substantially many terminals. As our results suggest, the subgraph containing more terminals tends to improve the current as well as the final solution better. In a broader context, the algorithm of Dvořák et al. [15] cannot approximate Steiner Arborescence well, since even parameterized approximation is hard from parameterized complexity view [15]. Is this still true in practice?

References

1 11th DIMACS Implementation Challenge, 2011. URL: `http://dimacs11.zib.de/`.
2 PACE Challenge 2018, 2018. URL: `https://pacechallenge.wordpress.com/pace-2018/`.
3 John E. Beasley. Or-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990. `doi:10.1057/jors.1990.166`.
4 Stephan Beyer and Markus Chimani. Strong steiner tree approximations in practice. *ACM J. Exp. Algorithmics*, 24(1), January 2019. `doi:10.1145/3299903`.
5 Édouard Bonnet and Florian Sikora. The PACE 2018 Parameterized Algorithms and Computational Experiments Challenge: The Third Iteration. In Christophe Paul and Michał Pilipczuk, editors, *13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages

26:1–26:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.IPEC.2018.26.

6 Al Borchers and Ding-Zhu Du. Thek-steiner ratio in graphs. *SIAM Journal on Computing*, 26(3):857–869, 1997. doi:10.1137/S0097539795281086.

7 Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. Steiner Tree Approximation via Iterative Randomized Rounding. *Journal of the ACM*, 60(1):1–33, February 2013. doi:10.1145/2432622.2432628.

8 Markus Chimani and Matthias Woste. Contraction-based steiner tree approximations in practice. In *Algorithms and Computation*, pages 40–49. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-25591-5_6.

9 Marcus Poggi de Aragão and Renato F. Werneck. On the implementation of MST-based heuristics for the steiner problem in graphs. In *Algorithm Engineering and Experiments*, pages 1–15. Springer Berlin Heidelberg, 2002. doi:10.1007/3-540-45643-0_1.

10 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

11 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959. doi:10.1007/BF01386390.

12 Stuart E. Dreyfus and Robert A. Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971. doi:10.1002/net.3230010302.

13 Cees Duin. *Steiner's problem in graphs*. PhD thesis, University of Amsterdam, 1993.

14 Cees Duin. *Preprocessing the Steiner Problem in Graphs*, pages 175–233. Springer US, Boston, MA, 2000. doi:10.1007/978-1-4757-3171-2_10.

15 Pavel Dvořák, Andreas Emil Feldmann, Dušan Knop, Tomáš Masařík, Tomáš Toufar, and Pavel Veselý. Parameterized approximation schemes for steiner trees with small number of steiner vertices. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 26:1–26:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.26.

16 Bernhard Fuchs, Walter Kern, Daniel Mölle, Stefan Richter, Peter Rossmanith, and Xinhui Wang. Dynamic programming for minimum steiner trees. *Theory Comput. Syst.*, 41(3):493–500, 2007. doi:10.1007/s00224-007-1324-4.

17 Gerald Gamrath, Thorsten Koch, Stephen J. Maher, Daniel Rehfeldt, and Yuji Shinano. Scip-jack – a solver for STP and variants with parallelization extensions. *Math. Program. Comput.*, 9(2):231–296, 2017. doi:10.1007/s12532-016-0114-x.

18 Stefan Hougardy, Jannik Silvanus, and Jens Vygen. Dijkstra meets steiner: a fast exact goal-oriented steiner tree algorithm. *Mathematical Programming Computation*, 9(2):135–202, June 2017. doi:10.1007/s12532-016-0110-1.

19 Radek Hušek, Dušan Knop, and Tomáš Masařík. Approximation algorithms for steiner tree based on star contractions: A unified view, 2020. arXiv:2002.03583.

20 Radek Hušek, Tomáš Toufar, Dušan Knop, Tomáš Masařík, and Eduard Eiben. Steiner tree heuristics for PACE 2018 Challenge track C, 2018. URL: https://github.com/goderik01/PACE2018.

21 Frank K. Hwang, Dana S. Richards, and Pawel Winter. *The Steiner tree problem*, volume 53. Elsevier, 1992. doi:10.1016/s0167-5060(08)x7008-6.

22 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Plenum, 1972. doi:10.1007/978-1-4684-2001-2_9.

23 Thorsten Koch and Alexander Martin. Solving steiner tree problems in graphs to optimality. *Networks*, 32(3):207–232, 1998. doi:10.1002/(SICI)1097-0037(199810)32:3<207::AID-NET5>3.0.CO;2-O.

24 Thorsten Koch and Daniel Rehfeldt. SCIP-jack, 2018. URL: https://github.com/dRehfeldt/scipjack/.

25 Lawrence Kou, George Markowsky, and Leonard Berman. A fast algorithm for steiner trees. *Acta Informatica*, 15(2):141–145, June 1981. doi:10.1007/BF00288961.

26 Jiří Matoušek and Jaroslav Nešetřil. *Invitation to Discrete Mathematics*. Oxford University Press, Inc., New York, NY, USA, 1998.

27 Kurt Mehlhorn. A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters*, 27(3):125–128, 1988. doi:10.1016/0020-0190(88)90066-X.

28 Michael Ian Shamos and Dan Hoey. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science, Berkeley, California, USA, October 13-15, 1975*, pages 151–162, 1975. doi:10.1109/SFCS.1975.8.

29 Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

30 Eduardo Uchoa, Marcus Poggi de Aragão, and Celso C. Ribeiro. Preprocessing Steiner problems from VLSI layout. *Networks*, 40(1):38–50, 2002. doi:10.1002/net.10035.

31 Dimitri Watel and Marc-Antoine Weisser. A practical greedy approximation for the directed steiner tree problem. *Journal of Combinatorial Optimization*, 32(4):1327–1370, November 2016. doi:10.1007/s10878-016-0074-0.

32 Jeffery Westbrook and Robert E. Tarjan. Maintaining bridge-connected and biconnected components on-line. *Algorithmica*, 7(1):433–464, June 1992. doi:10.1007/BF01758773.

33 Alexander Z. Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9(5), 1993. doi:10.1007/BF01187035.

Fixed-Parameter Tractability of the Weighted Edge Clique Partition Problem

Andreas Emil Feldmann
Department of Applied Mathematics, Charles University, Prague, Czech Republic

Davis Issac
Hasso Plattner Institute, Potsdam, Germany

Ashutosh Rai
Department of Applied Mathematics, Charles University, Prague, Czech Republic

Abstract

We develop an FPT algorithm and a compression for the Weighted Edge Clique Partition (WECP) problem, where a graph with n vertices and integer edge weights is given together with an integer k, and the aim is to find k cliques, such that every edge appears in exactly as many cliques as its weight. The problem has been previously only studied in the unweighted version called Edge Clique Partition (ECP), where the edges need to be partitioned into k cliques. It was shown that ECP admits a kernel with k^2 vertices [Mujuni and Rosamond, 2008], but this kernel does not extend to WECP. The previously fastest algorithm known for ECP has a runtime of $2^{\mathcal{O}(k^2)}n^{O(1)}$ [Issac, 2019]. For WECP we develop a compression (to a slightly more general problem) with 4^k vertices, and an algorithm with runtime $2^{\mathcal{O}(k^{3/2}w^{1/2}\log(k/w))}n^{O(1)}$, where w is the maximum edge weight. The latter in particular improves the runtime for ECP to $2^{\mathcal{O}(k^{3/2}\log k)}n^{O(1)}$.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Edge Clique Partition, fixed-parameter tractability, kernelization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.17

Related Version A full version of the paper is available at `https://arxiv.org/abs/2002.07761`.

Funding All the three authors were supported by Center for Foundations of Modern Computer Science (Charles Univ. project UNCE/SCI/004).

1 Introduction

Problems that aim to cover a graph by a small number of cliques have a long history and have been studied extensively in the past (see e.g. [2, 3, 5, 10, 16, 18, 7, 8]). For these types of problems we are given a graph G and an integer k, and the tasks include to either cover or partition the edges or the vertices of G using at most k cliques or bicliques (i.e., complete bipartite graphs). Plenty of applications exist in both theory [22] and practice, e.g., in computational biology [1, 6], compiler optimization [21], language theory [11], and database tiling [9]. In this paper, we study the variant called the Edge Clique Partition (ECP) problem, defined as follows.

ECP (Edge Clique Partition)
Input: a graph G on n vertices, a positive integer k
Output: a partition of the edges of G into k cliques (if it exists, otherwise output NO)

ECP is known to be NP-hard even in K_4-free graphs and chordal graphs [16], and together with [14], the reductions of [16] imply APX-hardness. To circumvent these hardness results, we focus on *parameterized algorithms* (see [4] for the basics). More specifically, we focus on FPT algorithms for the natural parameter k, i.e., the number of cliques. Fleischer et al. [7] show that on planar graphs, ECP can be solved in $\mathcal{O}^*(2^{96\sqrt{k}})$ time[1]. They also

[1] The $\mathcal{O}^*$-notation hides polynomial factors in input size.

generalized the result to d-degenerate graphs, giving an algorithm with $\mathcal{O}^*(2^{dk})$ runtime, which has a linear exponent for bounded-degeneracy graphs. For K_4-free graphs, Mujuni and Rosamond [18] gave an algorithm with a runtime[2] of $\mathcal{O}^*((\frac{k+3}{2})^k) = \mathcal{O}^*(2^{\mathcal{O}(k \log k)})$, which was improved by Fleischer et al. [7] to $\mathcal{O}^*((\sqrt{k}/3)^k)$ and even $\mathcal{O}^*((64c)^k)$ for some large (unspecified) constant c. Hence, also for these graphs an exponent linear in k is possible, albeit with a very large base. On the other hand, the algorithm of Mujuni and Rosamond [18] for K_4-free graphs has been empirically shown [24] to be rather efficient, even though it "only" comes with a near-linear exponent of $\mathcal{O}(k \log k)$.

Mujuni and Rosamond [18] showed that ECP is FPT in k for general graphs, by giving a *kernel* (see [4] for definition) of size k^2. However, no algorithms with (near-)linear dependence on k in the exponent are known for ECP. The fastest algorithm so far is given by Issac [12, Theorem 3.10] and runs in $\mathcal{O}^*(2^{2k^2+k \log_2 k+k})$ time, i.e., the exponent is quadratic in k. This algorithm is an adaptation of an algorithm by Chandran et al. [3] for the Biclique Partition problem (where we want to partition the edges into k bicliques) in bipartite graphs. In contrast, the best runtime lower bound known for ECP only excludes a sub-linear dependence on k in the exponent: if n denotes the number of vertices of the input graph, there is no $2^{o(k)}n^{O(1)}$ time algorithm for ECP assuming the Exponential Time Hypothesis (ETH). This follows due to a $2^{o(n)}$ lower bound for 3-Dimensional Matching [13] under ETH, and a reduction from Exact 3-Cover (which is a generalization of 3-Dimensional Matching) to ECP by Ma et al. [16]. An obvious open problem arising here is to close the gap between the upper and lower bounds on the runtime for ECP. Our main contribution is to show that for general graphs the exponent of the runtime for ECP can be significantly lowered from $\mathcal{O}(k^2)$ to $\mathcal{O}(k^{3/2} \log k)$.

▶ **Theorem 1.** *ECP has an algorithm running in* $\mathcal{O}\left((2e\sqrt{k})^{k^{3/2}+k} \cdot k^2 2^{5k} + n^2 \log n\right)$ *time.*

In fact, our algorithm solves a more general problem that we call the *Weighted Edge Clique Partition* (WECP) problem defined as follows:

WECP (Weighted Edge Clique Partition)
Input: a graph G on n vertices, edge weights $w_e : E(G) \to \mathbb{N}$, and a positive integer k
Output: a multiset of at most k cliques such that each edge appears in exactly as many cliques as its weight (if it exists, otherwise output NO)

Note that WECP is equivalent to ECP on a multigraph, by taking the weights as the edge multiplicities, which however increases the encoding length.

WECP can be thought of as a *clustering* of vertices where the clusters are allowed to overlap and the weight of an edge denotes the number of clusters in which the endpoints appear together. Such clustering problems appear naturally in computational biology, e.g., in the inference of gene pathways from gene co-expression data [20], where the clusters correspond to pathways and vertices correspond to genes. Thus developing efficient algorithms for WECP is of practical relevance.

WECP has not been studied previously and the known FPT algorithms for ECP do not extend to WECP. In particular, the techniques from the k^2-kernel for ECP by Mujuni and Rosamond [18] does not extend to WECP. Also, a 3^k-kernel for the very similar Biclique Partition problem by Fleischer et al. [8] just uses twin-reduction rule but this does not work for WECP. We first show a compression (see preliminaries for definition) with 4^k vertices for WECP that can be computed in polynomial time. The compression is into an even more general (auxiliary) problem that we call the *Annotated Weighted Edge Clique Partition* (AWECP) problem, defined as follows.

[2] In [18] the runtime was mistakenly reported as $\mathcal{O}^*(k^{(k+3)/2})$, cf. [7].

AWECP (Annotated Weighted Edge Clique Partition)
Input: a graph G on n vertices, edge-weights $w_e : E(G) \to \mathbb{N}$, a special set of vertices $W \subseteq V(G)$, vertex weights $w_v : W \to \mathbb{N}$, and a positive integer k
Output: a multiset of at most k cliques such that each edge e appears in exactly as many cliques as its edge-weight, and each vertex in W appears in exactly as many cliques as its vertex-weight (if such k cliques exist, otherwise output NO)

Note that WECP is exactly the special case of AWECP when W is empty. We give a kernel for AWECP that implies the compression for WECP into AWECP.

▶ **Theorem 2.** *AWECP has a kernelization algorithm that runs in $\mathcal{O}(n^2 \log n)$ time and outputs a kernel having at most 4^k vertices and encoding length $\mathcal{O}(16^k \log k)$ bits.*

▶ **Corollary 3.** *WECP has a compression into an AWECP instance having at most 4^k vertices and encoding length $\mathcal{O}(16^k \log k)$ bits. The compression can be found in $\mathcal{O}(n^2 \log n)$ time.*

Then we proceed to give the first FPT algorithm for WECP, which also implies the improved algorithm for ECP.

▶ **Theorem 4.** *WECP with the edge weights upper bounded by some value w has an algorithm running in $\mathcal{O}\left((2e\sqrt{k/w})^{k^{3/2}w^{1/2}+k} \cdot k^2 2^{5k} + n^2 \log n\right)$ time.*

Note that Theorem 4 implies an FPT algorithm for WECP when parameterized by k as $w \leq k$ for any YES-instance. Also, Theorem 1 follows from Theorem 4 by setting $w = 1$.

1.1 Our techniques

Our approach is based on the work of Chandran et al. [3], who solve the Bipartite Biclique Partition problem using linear algebraic techniques: we express AWECP as a low-rank matrix decomposition problem. For this we allow matrices to have wildcard entries in the diagonal that will be denoted by $\star$. We define $\mathbb{Z}^\star := (\mathbb{Z}_{\geq 0} \cup \{\star\})$. For $x, y \in \mathbb{Z}^\star$, we write $x \stackrel{\star}{=} y$ if and only if either $x = y$, or at least one of x and y is $\star$. For two matrices X and Y in $(\mathbb{Z}^\star)^{m \times n}$, we write $X \stackrel{\star}{=} Y$ if and only if $X_{i,j} \stackrel{\star}{=} Y_{i,j}$ for all i, j. We say that a *binary matrix* B (not containing wildcards) is a *Binary Symmetric Decomposition (BSD)* of a matrix $A \in (\mathbb{Z}^\star)^{n \times n}$ if $BB^T \stackrel{\star}{=} A$. The matrix B is called a width-k BSD of A if it is a BSD of A and has at most k columns. We define the *Binary Symmetric Decomposition with Diagonal Wildcards* (BSD-DW) problem as follows

BSD-DW (Binary Symmetric Decomposition with Diagonal Wildcards)
Input: an integer non-negative symmetric matrix $A \in (\mathbb{Z}^\star)^{n \times n}$ such that the wildcards $\star$ appear only in the diagonal, and an integer k
Output: a *width-k BSD* of A (if it exists, otherwise output NO)

We prove (in Lemma 6) that AWECP and BSD-DW are equivalent. Moreover, each column of B (solution to BSD-DW) corresponds to a clique (in the solution to AWECP), i.e. the rows that have a 1 in the j-th column correspond to the vertices that are in the j-th clique. Due to this, we will index the rows and columns of A with vertices, the rows of B with vertices and the columns of B with integers from $[k]$, that correspond to the k cliques. Moreover, we will be fluently switching between the contexts of edge partition of graphs (AWECP), and matrix decomposition (BSD-DW).

In Section 2 we prove that there is a kernel for AWECP with 4^k vertices. For this, we define the notion of $\star$-twins where two vertices u and v are said to be $\star$-twins, if the rows A_u and A_v are equal under $\stackrel{\star}{=}$. We group the vertices into equivalence classes (that we call

blocks) of $\star$-twins. If a block has size more than 2^k, we show that they can be reduced and represented by one vertex. For this reduction rule, we need to specify how often the representative vertex needs to be covered by cliques. Thus, even if the input is an instance of WECP, the kernel we compute will be annotated, i.e., it will be an instance of AWECP. The 4^k bound on the kernel size follows then by giving a 2^k upper bound on the number of blocks for a YES instance. Since the edge weights and vertex weights for vertices in W cannot exceed k if there is a solution with at most k cliques, a kernel with at most 4^k vertices can be encoded using $\mathcal{O}(\binom{4^k}{2} \log k)$ bits, and so Theorem 2 follows.

To obtain Theorem 4, we first compute a kernel using Theorem 2 as the first step of the algorithm. Our algorithm will solve the more general AWECP problem. As in the algorithm of Chandran et al. [3] (where a different low-rank matrix decomposition problem is solved), the main idea of our algorithm is to guess a row basis for a width-k BSD B, and then fill the remaining rows of B one by one independent of each other. However we need to refine the techniques of Chandran et al. [3] in order to obtain our runtime improvement. In particular, there are two reasons why the algorithm in [3] has a quadratic dependence on k in the exponent: first, to guess a basis of rank k, they need to guess k binary vectors of length k each, which takes $\mathcal{O}(2^{k^2})$ time. But also, they need to guess the k row basis indices of B, for which there are $\binom{m}{k}$ possibilities if the matrix has m rows. Since for Bipartite Biclique Partition there is a kernel where $m \leq 2^k$ [8], this adds another factor of $\mathcal{O}(2^{k^2})$ to the runtime.

To circumvent these two runtime bottlenecks, in Section 3 we devise an algorithm that gets around guessing the row indices of the basis of the solution matrix B. Instead of guessing the whole basis, we add a row to the basis only when the current basis cannot *take care* of that row. While this makes our algorithm more involved than the one by Chandran et al. [3], it means that the only bottleneck left is guessing the basis entries. For BSD-DW we can show that a basis with only $k^{3/2}w^{1/2} + k$ ones exists, which follows from the well-studied Zarankiewicz problem [19]. This bound on the structure of the basis then implies Theorem 4.

Since the only bottleneck, which prevents our algorithm from having near-linear dependence on k in the exponent of the runtime, is the step that guesses the entries of the basis for the solution matrix B, a natural question is whether our upper bound of $k^{3/2}w^{1/2} + k$ of the number of ones is (asymptotically) tight. In Section 4 we show that this is indeed tight (at least for the unweighted case) by proving the following theorem:

▶ **Theorem 5.** *For every prime power N and $k = N^2 + N$, there is a matrix $A \in \{0,1\}^{(k+1)\times(k+1)}$ such that there is a width-k BSD for A and every row basis of every width-k BSD of A has $\Theta(k^{3/2})$ ones.*

While this does not give a runtime lower bound in general, it implies that in order to speed up our algorithm for ECP using a better enumeration of the potential basis matrices, one needs to use some property other than a bound on the number of ones. The tight instances are obtained via the well-known Finite Projective Planes.

1.2 Related results

We now survey some results for ECP and related problems, apart from those mentioned above. For ECP, it is also known that the problem is solvable in polynomial time on cubic graphs [7]. The problem of partitioning the vertices instead of the edges into k cliques is equivalent to k-coloring on the complement graph, which is well-known to be NP-hard even for $k = 3$. Similarly, when the vertices need to be partitioned into bicliques or covered by bicliques, Fleischer et al. [8] proved NP-hardness for any constant $k \geq 3$.

Covering the edges of a graph by cliques or bicliques turns out to be generally harder than *partitioning* the edges. For the Edge Clique Cover problem, a kernel with 2^k vertices was shown by Gramm et al. [10], which results in a double-exponential time FPT algorithm when solving the kernel by brute-force. Cygan et al. [5] showed that this is essentially best possible, as under ETH no $2^{2^{o(k)}}n^{O(1)}$ time algorithm exists for Edge Clique Cover and no kernel of size $2^{o(k)}$ exists unless $\mathsf{P} = \mathsf{NP}$. Similarly, for the Biclique Cover problem, where edges of a general graph need to be covered by bicliques, Fleischner et al. [8] gave a kernel with 3^k vertices, and for the Bipartite Biclique Cover problem they gave a kernel with 2^k vertices in each bipartition. These kernels naturally imply double-exponential time algorithms. Chandran et al. [3] proved that for Bipartite Biclique Cover, under ETH no $2^{2^{o(k)}}n^{O(1)}$ time algorithm exists, and unless $\mathsf{P} = \mathsf{NP}$ no kernel of size $2^{o(k)}$ exists.

Chalermsook et al. [2] showed that for the Biclique Cover problem, it is NP-hard to compute an $n^{1-\varepsilon}$-approximation for any $\varepsilon > 0$ [3]. Edge Clique Cover is hard to approximate within $n^{0.5-\varepsilon}$ due to a reduction by Kou et al. [15]. In contrast, a PTAS exists for Edge Clique Cover on planar graphs [1].

1.3 Preliminaries

A problem P_1 parameterized by k is said to admit a compression into problem P_2 if there is an algorithm that takes as input an instance I_1 of P_1, runs in time polynomial in the encoding length of I_1, and outputs an instance I_2 of P_2 that is equivalent to I_1 such that the encoding length of I_2 is at most $f(k)$ for some computable function $f : \mathbb{N} \to \mathbb{N}$. In particular, the size of I_2 depends only on the parameter k and not on the size of I_1.

For an $m \times n$ matrix A, we use $A_{i,j}$ to denote the entry of A at row i and column j. We use A_i to denote the row-vector given by the i-th row of A. For some $I \subseteq [m]$ and $J \subseteq [n]$, we use $A_{I,J}$ to denote the sub-matrix of A when restricted to rows with indices in I and columns with indices in J. Also, we use A_I to denote a sub-matrix of A when restricted to rows with indices in I. We call such a sub-matrix where only rows are restricted as row sub-matrix. A *row-basis* (or just *basis* for brevity) B of A is any row sub-matrix of A such that every row of A can be expressed as a linear combination of rows of B, and the rows of B are linearly independent with each other.

▶ **Lemma 6.** *Given an instance (G, w_e, W, w_v, k) of AWECP we can find an equivalent instance (A, k) of BSD-DW in $\mathcal{O}(|V(G)|^2)$ time. Similarly given an instance (A, k) of BSD-DW, we can find an equivalent instance of AWECP in $\mathcal{O}(n^2)$ time, where n is the number of rows (or columns) in A.*

Proof. Given an instance (G, w_e, W, w_v, k) of AWECP, we can construct an instance of (A, k) of BSD-DW as follows. Let $V(G) = \{1, \ldots, n\}$; take the non-diagonal entries of A as the corresponding entries of the *weighted adjacency matrix* of G, i.e., if there is an edge between two vertices u and v, the entry $A_{u,v}$ is equal to $w_e(uv)$ and if u and v do not have an edge between them then $A_{u,v} = 0$; for every vertex $v \in W$, take $A_{v,v}$ as the vertex weight of v; for every vertex $v \in V(G) \setminus W$, take $A_{v,v}$ as the wildcard $\star$. Note that the mapping is invertible, i.e., given a BSD-DW instance (A, k) we get an AWECP instance (G, w_e, W, w_v, k) as follows. Take $V(G) := \{1, 2, \ldots, n\}$ where n is the number of rows (and columns) of A. For distinct $u, v \in [n]$, if $A_{u,v}$ is non-zero, put an edge between u and v in G with weight $A_{u,v}$. For each

[3] The paper wrongly claims the same result also for Biclique Partition. The bug is acknowledged here: `https://sites.google.com/site/parinyachalermsook/research?authuser=0`.

$v \in [n]$ such that $A_{v,v}$ is not a wildcard, put v in W and set its vertex weight to $A_{v,v}$. It is clear that this mapping is a bijective mapping between AWECP and BSD-DW instances and can be calculated in both directions in $\mathcal{O}(n^2)$ time. It remains to prove that the instances are equivalent.

Now, we define a bijective mapping between candidate solutions of the two problems. Naturally, a candidate solution of AWECP is a multiset of k cliques and a candidate solution of BSD-DW is an $n \times k$ matrix. Consider a candidate solution $\mathcal{C} := \{C_1, C_2, \ldots C_k\}$ of an AWECP instance (G, w_e, W, w_v, k). We map it to a candidate solution $B \in \{0,1\}^{n \times k}$ of a BSD-DW instance (A, k) as follows. Take the row B_u as the characteristic vector of u in the k cliques, i.e., $B_{u,j} := 1$ if $u \in C_j$, and $B_{u,j} := 0$ otherwise. The inverse mapping then turns out to be as follows. Given a candidate solution $B \in \{0,1\}^{n \times k}$ of instance (A, k) construct k cliques where the j-th clique is $C_j := \{u \mid B_{u,j} = 1\}$. To see that C_j is indeed a clique, consider any two vertices $u, v \in C_j$: since $B_{u,j} = B_{v,j} = 1$, we know that $A_{u,v} = B_u B_v^T \geq 1$, which implies that there is an edge between u and v in G.

First, we prove that if $\mathcal{C}$ is a solution of AWECP(G, w_e, W, w_v, k), then B is a solution of BSD-DW(A, k). It is clear that B has only k columns by construction. So, it only remains to prove that for all pairs $u, v \in [n]$, $B_u B_v^T \stackrel{\star}{=} A_{u,v}$. First consider the case when u and v are distinct. Let J denote the set of all j such that both u and v appear together in C_j. Since $\mathcal{C}$ is a solution of AWECP(G, w_e, W, w_v, k), we have that $|J| = A_{u,v}$. By construction of B, we have that J is exactly the set of indices j where $B_{u,j} = B_{v,j} = 1$. Thus $B_u B_v^T = |J| = A_{u,v}$. Now consider the case when $u = v$. If $A_{u,u}$ is a $\star$ then clearly $B_u B_u^T \stackrel{\star}{=} \star = A_{u,u}$. So, suppose $A_{u,u} \neq \star$. This means $u \in W$ implying that u appears in exactly $A_{u,u}$ many cliques in $\mathcal{C}$. Thus $B_u B_u^T = A_{u,u}$.

We now prove the reverse direction, i.e., we prove that if B is a solution of BSD-DW(A, k), then $\mathcal{C}$ is a solution of AWECP(G, w_e, W, w_v, k). By construction, $\mathcal{C}$ has at most k cliques. Thus, it is sufficient to prove the following two statements: (1) every pair $u, v \in V(G)$ appears together in exactly $A_{u,v}$ many cliques in $\mathcal{C}$ (2) each vertex $v \in W$ appears in $A_{v,v}$ many cliques in $\mathcal{C}$. First we prove (1). We know $B_u B_v^T = A_{u,v}$. Since B is binary, this means that there are exactly $A_{u,v}$ many indices j such that $B_{u,j}$ and $B_{v,j}$ are both 1. Let J be the set of those indices. Observe that the set of cliques where both u and v appear together are exactly $\{C_j : j \in J\}$. Thus, the edge uv is in $|J| = A_{u,v}$ many cliques. Now we prove (2). Consider a vertex $v \in W$. We know $B_v B_v^T = A_{v,v}$. Since B is binary, this means that there are exactly $A_{v,v}$ many ones in B_v. Thus, the vertex v is in $A_{v,v}$ many cliques. ◀

2 Kernel

We will now give a kernel for AWECP and BSD-DW, thereby proving Theorem 2. Let (G, w_e, W, w_v, k) be an instance of AWECP and (A, k) be the corresponding instance of BSD-DW obtained by the transformation as in the proof of Lemma 6. We may move seamlessly between the graph and matrix terminologies as both problems are equivalent. Whenever we say a solution in this section, we mean the solution to the BSD-DW instance i.e., a width-k BSD of A. We say two distinct vertices u and v in G are **$\star$-twins** if they are adjacent and satisfy $A_u \stackrel{\star}{=} A_v$. We now prove the following easy property of $\star$-twins.

▶ **Lemma 7.** *For distinct vertices u, v and w in G, suppose u and v are $\star$-twins and v and w are $\star$-twins. Then:*

1. *u and w are $\star$-twins, and*
2. *all the entries of the submatrix $A_{\{u,v,w\},\{u,v,w\}}$ are the same except for wildcards.*

Proof. First, let us prove the second statement. Let $A_{u,v} = \alpha$. Then we know $A_{u,w} = \alpha$ as v and w are $\star$-twins. Then $A_{v,w} = \alpha$ as u and v are $\star$-twins. Thus all the non-diagonal elements of $A_{\{u,v,w\}\{u,v,w\}}$ are equal to α. If $A_{u,u} \neq \star$ then $A_{u,u} = A_{v,u} = \alpha$ as u and v are $\star$-twins. Similarly, if $A_{v,v} \neq \star$ then $A_{v,v} = A_{v,u} = \alpha$ as u and v are $\star$-twins. And, if $A_{w,w} \neq \star$ then $A_{w,w} = A_{v,w} = \alpha$ as v and w are $\star$-twins.

Now, for the first statement to hold, we only need to show that $A_{u,z} = A_{w,z}$ for all $z \notin \{u, v, w\}$. Indeed, $A_{u,z} = A_{v,z} = A_{w,z}$ where the first equality is because u and v are $\star$-twins and the second is because v and w are $\star$-twins. ◀

Thus we have that the relation *$\star$-twins* is transitive. It is also symmetric, as easily seen from the definition. Note that $\star$-twins are required to be adjacent, and thus the relation is not reflexive. But to make it reflexive, we simply define a vertex to be a $\star$-twin of itself. Thus, we can group the vertices into equivalence classes of $\star$-twins. We call each equivalence class a **block**. Note that there can be blocks containing only a single vertex. The following lemma is a direct consequence of Lemma 7.

▶ **Lemma 8.** *For a block D, the entries of the sub-matrix $A_{D,D}$ are all same except for wildcards.*

▶ **Fact 9.** *For values a, b and c, if $a \overset{\star}{=} b$ and $b \overset{\star}{=} c$, and $b \neq \star$ then $a \overset{\star}{=} c$.*

▶ **Lemma 10.** *Suppose we have a YES instance of AWECP without isolated vertices. Then there can be at most 2^k blocks.*

Proof. Let B be a width-k BSD of A. Note that B exists as we have a YES instance. In order to prove the lemma, it is sufficient to show that if u and v are in different blocks, then B_u and B_v are distinct, because then there can only be 2^k distinct rows of B, as there are only k columns in B and B is binary. Assume for the sake of contradiction that $B_u = B_v$ and u and v are in different blocks, i.e., they are not $\star$-twins. Let $b := B_u B^T = B_v B^T$. We have $A_u \overset{\star}{=} B_u B^T = b$ and $A_v \overset{\star}{=} B_v B^T = b$. This implies $A_u \overset{\star}{=} A_v$ using Fact 9, as the vector b contains no wildcards. Then, for u and v to be not $\star$-twins, it should be the case that u and v are not adjacent, i.e, $A_{u,v} = 0$. But then, $B_u B_v^T = 0$. Since $B_u = B_v$ by assumption, we have that $B_u = B_v = \mathbf{0}$ and hence $A_u = A_v = \mathbf{0}$. This means that u and v are isolated vertices, which is a contradiction. ◀

The above lemma shows the soundness of our first reduction rule that is as follows.

▶ **Reduction rule 1.** *If the number of blocks is more than 2^k, output that the instance is a NO instance.*

Next, we prove the following lemma about $\star$-twins that helps us to come up with a reduction rule that bounds the size of each block.

▶ **Lemma 11.** *Let $D := \{v_1, v_2, \ldots, v_t\}$ be a block of $\star$-twins. For a YES instance, there exists a solution B such that the rows $B_{v_1}, B_{v_2}, \ldots, B_{v_t}$ are either all pairwise distinct, or all same.*

Proof. It is sufficient to prove the following statement: if there is a solution B such that $B_{v_1} = B_{v_2}$, then there is also a solution C such that $C_{v_1} = C_{v_2} = \cdots = C_{v_t}$. So, assume that $B_{v_1} = B_{v_2}$. Let C be the matrix defined as $C_v := B_v$ for all $v \notin D$, and $C_v := B_{v_1} = B_{v_2}$ for all $v \in D$. We will prove that C is also a solution. For this, it is sufficient to prove that $C_u C_v^T = A_{u,v}$ for all $u, v \in V$ such that $A_{u,v} \neq \star$. If both u and v are not in D, then $C_u C_v^T = B_u B_v^T = A_{u,v}$. So, without loss of generality assume that $u \in D$. We distinguish the following cases.

1. If $v \in V \setminus D$, then $C_u C_v^T = B_{v_1} B_v^T = A_{v_1,v} = A_{u,v}$, where the last equality follows as v_1 and u are $\star$-twins.
2. If $v \in D \setminus \{u\}$, then $C_u C_v^T = B_{v_1} B_{v_2}^T = A_{v_1,v_2} = A_{u,v}$, where the last equality follows from Lemma 8.
3. If $v = u$: if $A_{u,u} = \star$ then there is nothing to prove, so assume $A_{u,u} \neq \star$. Then $A_{u,u} = A_{v_1,v_2}$ by Lemma 8. Hence we get $C_u C_u^T = B_{v_1} B_{v_2}^T = A_{v_1,v_2} = A_{u,u}$. ◀

Since there are only 2^k possible distinct rows for a solution B, Lemma 11 has the following consequence.

▶ **Lemma 12.** *Let $D := \{v_1, v_2, \ldots, v_t\}$ be a block of $\star$-twins such that $t > 2^k$. For a YES instance, there exists a solution B such that the rows $B_{v_1}, B_{v_2}, \ldots B_{v_t}$ are all same.*

The above lemma suggests that for a block D of size more than 2^k, we only need to keep one representative vertex for all the vertices in D. This leads us to our second reduction rule.

▶ **Reduction rule 2.** *Suppose there is a block D with more than 2^k vertices. Pick any two arbitrary vertices $u, v \in D$. We reduce our instance to an instance A' of AWECP (simultaneously to an instance G' of BSD-DW) as follows: let $G' := G \setminus (D \setminus \{v\})$; for every pair $(v_1, v_2) \neq (v, v)$ in $V(G') \times V(G')$, let $A'_{v_1,v_2} := A_{v_1,v_2}$; let $A'_{v,v} := A_{u,v}$.*

Once we have a solution B' to the reduced instance A' then we construct a solution B to the original instance A as follows: for all $x \in D$, let $B_x := B'_v$; for all $x \in V(G) \setminus D$, let $B_x := B'_x$.

Now, we prove that the above reduction rule is safe.

▶ **Lemma 13.** *Let A', G', B', B be as defined in Reduction rule 2.*
1. *If B' is a width-k BSD of A', then B is a width-k BSD of A.*
2. *Conversely, if A has a width-k BSD then so does A'.*

Proof. 1. It is clear that B has only k columns. So, it only remains to prove that B is a BSD of A, for which it is sufficient to prove that $B_{v_1} B_{v_2}^T \stackrel{\star}{=} A_{v_1,v_2}$ for all $v_1, v_2 \in V(G)$. For $v_1, v_2 \in V(G) \setminus D$, we have

$$B_{v_1} B_{v_2}^T = B'_{v_1} B'^T_{v_2} \stackrel{\star}{=} A'_{v_1,v_2} = A_{v_1,v_2}.$$

For $v_1 \in V(G) \setminus D$ and $v_2 \in D$, we have

$$B_{v_1} B_{v_2}^T = B'_{v_1} B'^T_{v} = A'_{v_1,v} = A_{v_1,v} = A_{v_1 v_2},$$

where the last equality follows as v and v_2 are $\star$-twins.
For $v_1, v_2 \in D$, we have

$$B_{v_1} B_{v_2}^T = B'_{v} B'^T_{v} = A'_{v,v} = A_{u,v} = A_{v_1,v_2},$$

where the last equality follows from Lemma 8.

2. By Lemma 12 we know that there exists a width-k BSD of A such that $B_{v_1} = B_{v_2}$ for all $v_1, v_2 \in D$. In particular $B_u = B_v$. Let B' be defined as $B'_x := B_x$ for all $x \in V(G')$. We show that B' is a width-k BSD of A'. Since B' has only k columns, it only remains to prove that B' is a BSD of A', which we do as follows. For $(v_1, v_2) \in (V(G') \times V(G')) \setminus (v, v)$, we have

$$B'_{v_1} B'^T_{v_2} = B_{v_1} B_{v_2}^T \stackrel{\star}{=} A_{v_1,v_2} = A'_{v_1,v_2},$$

and

$$B'_{v} B'^T_{v} = B_{v} B_{v}^T = B_{u} B_{v}^T = A_{u,v} = A'_{v,v}.$$ ◀

After the above rules are exhaustively applied, each block has size at most 2^k and the number of blocks is at most 2^k. Thus we have the required kernel of size 4^k.

The time required for computing the kernel can be shown to be $\mathcal{O}(n^2 \log n)$. This is because the blocks of $\star$-twins can be found by sorting the rows in lexicographic order. Since each comparison takes $\mathcal{O}(n)$ time the sorting can be done in $\mathcal{O}(n^2 \log n)$ time. Also, we need to compute the blocks only once as the reduction rules does not change the blocks.

Since the edge weights and vertex weights for vertices in W cannot exceed k if there is a solution with at most k cliques, a kernel with at most 4^k vertices can be encoded using $\mathcal{O}(\binom{4^k}{2} \log k)$ bits, and so Theorem 2 follows.

3 Algorithm

Here we give an algorithm for the BSD-DW problem. The algorithm also solves AWECP due to the equivalence from Lemma 6. In particular, it solves WECP thereby proving Theorem 4.

We now give a description of the algorithm. Pseudocode is given in Algorithm 1. Our input is a symmetric matrix $A \in (\mathbb{Z}_{\geq 0} \cup \{\star\})^{n \times n}$ where wildcards $\star$ appear only on the diagonal. First we guess a matrix $P \in \{0,1\}^{k \times k}$ such that for some $r \leq k$, $P_{[r],[k]}$ is a row basis of solution B. We show that for this, it is sufficient to enumerate $k \times k$ binary matrices that satisfy a specific property defined as follows. Let w be the largest integer entry of A. We call a matrix **w-limited** if the dot-product of each *distinct* pair of its rows is at most w. The following fact shows that we only need to enumerate w-limited matrices in $\{0,1\}^{k \times k}$ to guess P.

▶ **Fact 14.** *If B is a BSD of matrix A and w is the largest integer entry of A, then any submatrix of B (including B) is w-limited.*

Proof. Since B only has non-negative entries, if B is w-limited, then so are all the submatrices. Suppose the property does not hold for B. Then there exist two rows B_u and B_v such that $B_u B_v^T > w$. But $B_u B_v^T \stackrel{\star}{=} A_{uv}$ and hence $A_{uv} > w$ (note that A_{uv} is not $\star$ as it is not a diagonal-entry). Thus we have a contradiction. ◀

Note that guessing P is done in Loop 1 of Algorithm 1. We will later give a bound on the number of w-limited matrices in $\{0,1\}^{k \times k}$ during the runtime analysis in Section 3.2, thereby bounding the number of iterations of Loop 1.

We maintain partially filled matrices during the algorithm, i.e., we allow matrices to have *null rows* (this is different from wildcards). Think of the null rows as the rows that have not been filled yet. If each row of a matrix is either a binary row or a null row, we call it a *binary matrix with possibly null rows*. We denote by $\mathbb{B}^{n \times k}$, the set of all $n \times k$ binary matrices with possibly null rows.

We maintain a matrix $\tilde{B} \in \mathbb{B}^{n \times k}$ as a potential basis for our solution B. In Line 8, we call `CompleteBasis` that checks whether the current $\tilde{B}$ can be extended to a full solution B. Note that `CompleteBasis` does not try all possibilities to fill the remaining rows. It fills a row with the first binary vector that is compatible with the rows so far, where compatibility is defined as follows. For a matrix $B \in \mathbb{B}^{n \times k}$, we say that a vector $v \in \{0,1\}^k$ is **i-compatible** for B if $v^T v \stackrel{\star}{=} A_{i,i}$ and for all $j \neq i$ such that B_j is not a null row, $v^T B_j^T = A_{i,j}$. If `CompleteBasis` is able to fill all the rows with i-compatible binary vectors, then we are done and we return the resulting matrix (in Line 9). If not, we claim that the row for which we are not able to fill can be added to the basis (in Claim 16). So we add one more row to the basis by copying the next row from P (in Line 7). Thus we increase the number of non-null rows in the basis $\tilde{B}$ by one and repeat. Since the basis can be at most of size k, we need to repeat this at most k times.

Algorithm 1 Algorithm for BSD-DW.

Input : An $n \times n$ symmetric integer diagonal-wildcard matrix A
Output : If A has a width-k BSD then output a width-k BSD B of A; otherwise report that A has no width-k BSD

```
1  w ← largest integer weight in A
2  foreach w-limited P ∈ {0,1}^{k×k} do                          // Loop 1
3      Initialize B̃ to be an n × k matrix with all null rows
4      b ← 1
5      i ← 1
6      while b ≤ k and P_b is i-compatible with B̃ do             // Loop 2
7          B̃_i ← P_b
8          (B, i) ← CompleteBasis(A, B̃)
9          if i = n + 1 then output B and terminate the algorithm
10         b ← b + 1
11 output that A has no width-k BSD and terminate the algorithm

   Function CompleteBasis(A, B̃):
12     B ← B̃
13     for each null row i in B in increasing order do           // Loop 3
14         if there is a v ∈ {0,1}^k such that v is i-compatible with B then
15             B_i ← v
16         else return (B, i)
17     return (B, n + 1)
```

3.1 Correctness of the algorithm

The algorithm outputs either through Line 9 or through Line 11. In the former case, we prove the following claim.

▷ Claim 15. If output occurs through Line 9, then the matrix B that is output, is a width-k BSD of A.

Proof. If Line 9 is executed, then this means that the preceding `CompleteBasis` call on Line 8 returned $i = n + 1$. This implies that the return from `CompleteBasis` happened on Line 17. This in turn means that Loop 3 was exited after completing all iterations, implying that the matrix B did not have any null rows at the time of return. Thus $B \in \{0,1\}^{n \times k}$. The rows of B were each filled either in Line 7 (when it was $\tilde{B}$ before being passed to `CompleteBasis`) or in Line 15. In both places, we filled each row i with a vector that was i-compatible at the time of filling. From the definition of i-compatibility, it follows that $BB^T \overset{\star}{=} A$, and hence B is a width-k BSD of A. ◁

Consider a NO instance first. From Claim 15 it follows that the output does not occur through Line 9. Thus the output has to occur through Line 11 and hence we correctly output that A does not have a width-k BSD. So it only remains to prove the correctness when A is a YES instance, i.e., when A has a width-k BSD, which is the case we consider for the remainder of the proof. Let B^* be any fixed width-k BSD of A.

Observe that $\tilde{B}$ changes as follows during each iteration of Loop 1: it is initialized to all null rows and each time the algorithm encounters Line 7 a null row is replaced with a binary row vector. We say that a matrix B is **consistent** with B^* if $B_j = B^*_j$ for each j such that B_j is a non-null row.

▷ Claim 16. Consider a matrix $\tilde{B} \in \mathbb{B}^{n\times k}$ that is consistent with B^*. If `CompleteBasis`$(A, \tilde{B})$ returns $i \in [n]$ then B^*_i is linearly independent from the non-null rows of $\tilde{B}$.

Proof. For a matrix $M \in \mathbb{B}^{n\times k}$, we denote by $R(M)$ the set of indices of the non-null rows of M. Suppose for the sake of contradiction that `CompleteBasis`$(A, \tilde{B})$ returns $i \in [n]$ and B^*_i is linearly dependent on the non-null rows of $\tilde{B}$. Then, we have $B^*_i = \Sigma_{\ell\in R(\tilde{B})}\lambda_\ell \tilde{B}_\ell$ for some $\lambda_1, \lambda_2, \cdots, \lambda_\ell \in \mathbb{R}$. Since $\tilde{B}$ is consistent with B^*, we can write $B^*_i = \Sigma_{\ell\in R(\tilde{B})}\lambda_\ell B^*_\ell$.

As `CompleteBasis` returned i, we know that during that iteration of Loop 3 in which row i was considered, no vector $v \in \{0,1\}^k$ was i-compatible with B (here B is the matrix maintained by `CompleteBasis` that was initialized to $\tilde{B}$ on Line 12). In particular, $B^*_i \in \{0,1\}^k$ was not i-compatible with B. Therefore either there was some $j \in R(B)$ such that $B^*_i B^T_j \neq A_{i,j}$, or $B^*_i (B^*_i)^T \neq A_{i,i}$. The latter cannot be true as B^* is a width-k BSD of A. So there was a $j \in R(B)$ such that $B^*_i B^T_j \neq A_{i,j}$.

We branch into two cases: case 1 when $j \in R(\tilde{B})$ and case 2 when $j \in R(B) \setminus R(\tilde{B})$. In case 1, we have $B_j = \tilde{B}_j = B^*_j$ where the second equality is because $\tilde{B}$ and B^* are consistent. Thus $B^*_i B^T_j = B^*_i (B^*_j)^T = A_{i,j}$, giving a contradiction.

In case 2, B_j was added in Line 15 and hence B_j was j-compatible with B at this time, implying that $B_\ell B^T_j = A_{\ell,j}$ for all $\ell \in R(\tilde{B})$. Since $B_\ell = \tilde{B}_\ell = B^*_\ell$ for $\ell \in R(\tilde{B})$, we have that $B^*_\ell B^T_j = A_{\ell,j}$ for all $\ell \in R(\tilde{B})$. Then, we have a contradiction as follows:

$$\begin{aligned} B^*_i B^T_j &= \Sigma_{\ell\in R(\tilde{B})}\lambda_\ell B^*_\ell B^T_j \\ &= \Sigma_{\ell\in R(\tilde{B})}\lambda_\ell A_{\ell,j} \\ &= \Sigma_{\ell\in R(\tilde{B})}\lambda_\ell B^*_\ell (B^*_j)^T \\ &= B^*_i (B^*_j)^T \\ &= A_{i,j} \end{aligned}$$ ◁

For a matrix $X \in \{0,1\}^{k\times k}$, we say we are in iteration (X, t) of the algorithm if we are in the iteration of Loop 1 with $P = X$ and the iteration of Loop 2 with $b = t$. We use $\tilde{B}(X, t)$ to denote the value of $\tilde{B}$ after the execution of Line 7 during iteration (X, t).

▷ Claim 17. At any step of the algorithm, if $\tilde{B}$ is consistent with B^* then the non-null rows of $\tilde{B}$ are linearly independent.

Proof. Consider the first time this is violated during the algorithm. This has to be during the addition of a new non-null row at Line 7. Let (X, t) be the iteration in which this happens. Let p be the index of the row that was added. Observe that $\tilde{B}(X, t)$ has only one additional non-null row compared to $\tilde{B}(X, t-1)$. Also, this additional non-null row is equal to B^*_p as $\tilde{B}(X, t)$ is consistent with B^*. We know the rows of $\tilde{B}(X, t-1)$ are linearly independent as we assumed that the first violation of lemma happens in iteration (X, t). Also, during iteration $(X, t-1)$, i was returned with value p (as the insertion happens in Line 7 in iteration (X, t)). This implies that B^*_p is linearly independent from the non-null rows of $\tilde{B}(X, t-1)$ due to Claim 16. Hence the rows of $\tilde{B}(X, t)$ are linearly independent. ◁

▷ Claim 18. If the iteration (X, k) occurs during the algorithm for some $X \in \{0,1\}^{k\times k}$ such that $\tilde{B}(X, k)$ is consistent with B^* then the algorithm outputs through Line 9 in iteration (X, k).

Proof. Consider the i returned by `CompleteBasis`$(A, \tilde{B}(X,k))$. It is sufficient to prove that the condition $i = n+1$ in Line 9 is satisfied. Suppose otherwise. Then $i \in [n]$ and by Claim 16, B^*_i is linearly independent from the non-null rows of $\tilde{B}(X,k)$. But by Claim 17, we have that the non-null rows of $\tilde{B}(X,k)$ are linearly independent and hence span the whole space, thus giving a contradiction. ◁

▷ Claim 19. Assume that the output of the algorithm does not occur through Line 9. If for some $Y \in \{0,1\}^{k\times k}$ and $t \leq k-1$, iteration (Y,t) occurs and $\tilde{B}(Y,t)$ is consistent with B^*, then there exists some $Z \in \{0,1\}^{k\times k}$ such that iteration $(Z,t+1)$ occurs and $\tilde{B}(Z,t+1)$ is consistent with B^*.

Proof. Since $\tilde{B}(Y,t)$ is consistent with B^*, we know that $Y_{[t]}$ is a sub-matrix of B^*. As the condition in Line 9 is false, we know that an $i \in [n]$ was returned in Line 8 in iteration (Y,t). It is clear from the algorithm that i is a null-row in $\tilde{B}(Y,t)$. Let $Z \in \{0,1\}^{k\times k}$ be such that $Z_{[t]} := Y_{[t]}$, $Z_{t+1} := B^*_i$, and $Z_q := \mathbf{0}$ for all $q \geq t+1$. Observe that $Z_{[t+1]}$ is a submatrix of B^* and hence is w-limited by Fact 14. Since adding zeroes does not destroy w-limitedness, we have that Z is a w-limited $n \times k$ matrix. Thus there is some iteration of Loop 1 with $P = Z$. In this iteration the algorithm behaves similarly to the iteration with $P = Y$ for the first t iterations of Loop 2 as the algorithm has seen only the first t rows of P up to then. Thus $\tilde{B}(Z,t) = \tilde{B}(Y,t)$ and i is returned by Line 8 in iteration (Z,t). Now in Line 7 of iteration $(Z,t+1)$, $\tilde{B}_i$ is assigned Z_{t+1}. Note that $Z_{t+1} = B^*_i$ is indeed i-compatible with $\tilde{B}(Z,t)$ (as $\tilde{B}(Z,t) = \tilde{B}(Y,t)$ and $\tilde{B}(Y,t)$ is consistent with B^*) and that $t+1 \leq k$. Hence the loop condition of Loop 2 is true in iteration $(Z,t+1)$. Thus, we have $(\tilde{B}(Z,t+1))_i = Z_{t+1} = B^*_i$ and for all $j \neq i$, we have $(\tilde{B}(Z,t+1))_j = (\tilde{B}(Y,t))_j$. Since $\tilde{B}(Y,t)$ is consistent with B^*, it follows that $\tilde{B}(Z,t+1)$ is consistent with B^*. ◁

Let t be the largest number for which there exists a $P \in \{0,1\}^{k\times k}$ such that iteration (P,t) happens and $\tilde{B}(P,t)$ is consistent with B^*. Due to Claim 19, we know that $t = k$. Then the algorithm outputs through Line 9 according to Claim 18. Thus the algorithm outputs a correct solution B due to Claim 15.

3.2 Runtime analysis

First, let us bound the number of iterations of Loop 1. For this it is sufficient to bound the number of w-limited matrices in $\{0,1\}^{k\times k}$.

▶ **Lemma 20.** *The number of binary w-limited $k\times k$ matrices is at most $(2e\sqrt{k/w})^{k^{3/2}w^{1/2}+k}$.*

Proof. Note that no w-limited matrix can have a $2\times(w+1)$-sub-matrix having all ones. The number of ones in such a matrix is a special case of the well-studied Zarankiewicz problem and is known [19] to be at most $k^{3/2}w^{1/2}+k$. Hence it follows that the number of binary w-limited $k\times k$ matrices is at most $2^{k^{3/2}w^{1/2}+k} \cdot \binom{k^2}{k^{3/2}w^{1/2}+k}$ by choosing the positions of the at most $k^{3/2}w^{1/2}+k$ potential ones in the matrix and then choosing which of them are actually ones. The bound follows easily by using that $\binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$. ◀

Next, let us analyze the runtime of the function `CompleteBasis`. Loop 3 has at most n iterations. In Line 14, we need to check at most 2^k vectors $v \in \{0,1\}^k$. The checking for i-compatibility of each vector takes $\mathcal{O}(nk)$ time. Hence `CompleteBasis` takes $\mathcal{O}(k2^kn^2)$ time.

Now, we are ready to calculate the total run time. Due to Lemma 20, Loop 1 has at most $(2e\sqrt{k/w})^{k^{3/2}w^{1/2}+k}$ iterations. Line 3 takes $\mathcal{O}(nk)$ time. Loop 2 has at most k iterations. Line 7 takes at most $\mathcal{O}(k)$ time. The call to `CompleteBasis` in Line 8 takes at

most $\mathcal{O}(k2^k n^2)$ time as we already calculated. Any other step takes only constant time. Thus the total running time is bounded by $\mathcal{O}\left(\left((2e\sqrt{k/w})^{k^{3/2}w^{1/2}+k}\right)(nk + k(k + k2^k n^2))\right) = \mathcal{O}\left((2e\sqrt{k/w})^{k^{3/2}w^{1/2}+k} \cdot k^2 2^k n^2\right)$. We may run our algorithm on the kernel provided by Theorem 2, which means we may set $n = 4^k$ in the above expression. Thus the total running time is $\mathcal{O}\left((2e\sqrt{k/w})^{k^{3/2}w^{1/2}+k} \cdot k^2 2^{5k} + n^2 \log n\right)$. This proves Theorem 4.

4 Lower bound for number of ones in the basis matrix

In this section we construct binary matrices for which there is a width-k BSD and every basis of every width-k BSD has $\Omega(k^{3/2})$ ones, thereby proving Theorem 5. We obtain such instances via Finite Projective Planes (FPPs), which are defined as a set system $\mathcal{S}$ over a universe U of elements such that:

1. for each $e, e' \in U$ there is exactly one $S \in \mathcal{S}$ containing both of them,
2. for each $S, S' \in \mathcal{S}$ there is exactly one $e \in U$ such that $e \in S \cap S'$, and
3. there is a set of 4 elements in U such that no three of them are in any $S \in \mathcal{S}$.

It is known [17] that for any FPP, both the number of elements and the number of sets are equal to $N^2 + N + 1$ for some $N \geq 2$. Here N is called the *order* of the FPP. It also follows that for an FPP of order N, each set has exactly $N + 1$ elements and each element is contained in exactly $N + 1$ sets. It is also known that FPPs of order N exist for every prime power N [17]. Given an FPP of order N, in the following we will denote the characteristic incidence matrix of elements and sets by $F \in \{0, 1\}^{(N^2+N+1)\times(N^2+N+1)}$, where rows are elements and columns are sets.

We now give a reduction from FPPs to ECP. For this, consider a vertex set V with $N^2 + N + 1$ vertices. Let I be a subset of $N + 1$ vertices in V. Let G_N be the graph defined as the clique over V minus the clique over I, i.e., every pair of vertices in V is adjacent except when both are from I. In other words, if $X := V \setminus I$, then G_N is a split graph with X as the clique and I as the independent set, where all the adjacencies are present between X and I. In Lemmas 21 and 23, we show that G_N has a small ECP if and only if an FPP of order N exists.

▶ **Lemma 21.** *If a finite projective plane $\mathcal{S}$ of order N exists, then G_N has a clique partition $\mathcal{C}$ into $|\mathcal{C}| \leq N^2 + N$ cliques.*

Proof. Let $\mathcal{S}$ be an FPP of order N over a universe U, and fix one of its sets $S \in \mathcal{S}$. We identify this set with the independent set of G_N, i.e., $S = I$. After fixing the elements of S, all other elements in $U \setminus S$ are arbitrarily identified with the other vertices in X. We claim that the remaining sets in $\mathcal{S} \setminus \{S\}$ form a clique partition, i.e., if $C_{S'} = \{uv \in E(G_N) \mid u, v \in S'\}$ then the set $\mathcal{C} = \{C_{S'} \mid S' \in \mathcal{S} \setminus \{S\}\}$ partitions the edge set of G_N into cliques. From Property 1 of an FPP, for any edge uv (i.e., at least one of u and v is in X) there is exactly one set $S' \in \mathcal{S} \setminus \{S\}$ such that $u, v \in S'$. This means that the subgraphs in $\mathcal{C}$ partition the edge set. Furthermore, by Property 2 no $S' \in \mathcal{S} \setminus \{S\}$ intersects in more than one vertex with the independent set I. Thus every subgraph of $\mathcal{C}$ is a clique. Moreover, any FPP of order N has exactly $N^2 + N + 1$ sets, and so there are $N^2 + N$ cliques in $\mathcal{C}$. ◀

▶ **Lemma 22.** *If $\mathcal{C}$ is a set of cliques that partition the edges of G_N and $|\mathcal{C}| \leq N^2 + N$, then for each $C \in \mathcal{C}$, $|V(C)| = N + 1$.*

Proof. First let us prove that $|V(C)| \leq N+1$. Suppose for the sake of contradiction that $|V(C)| \geq N+2$. Note that C contains at most one vertex from I, as a clique and independent set can intersect on at most one vertex. Let $C' := V(C) \setminus I$ and $I' := I \setminus V(C)$. Clearly $|C'| \geq N+1$ and $|I'| \geq N$ (recall that $|I| = N+1$). Note that every edge in $C' \times I'$ has to be covered by a distinct clique in $\mathcal{C} \setminus \{C\}$: any two edges that have different endpoints in I cannot be in the same clique, since there is no edge between these endpoints, while any two edges with different endpoints in C cannot be in the same clique, since the only edge between these endpoints is already covered by C. But there are $|C'||I'| \geq N^2+N$ such edges whereas there are only N^2+N-1 cliques in $\mathcal{C} \setminus \{C\}$. Thus we have a contradiction.

Hence we established $|V(C)| \leq N+1$. Now suppose for the sake of contradiction $|V(C)| < N+1$. Using the fact that every clique of $\mathcal{C}$ has size at most $N+1$, the total number of edges covered by $\mathcal{C}$ is strictly less than $|\mathcal{C}|\binom{N+1}{2} \leq (N^2+N)\binom{N+1}{2} = N^2(N+1)^2/2$. However, since $|I| = N+1$ and consequently $|X| = N^2$, the total number of edges of G_N is $\binom{N^2}{2} + N^2 \cdot (N+1) = N^2(N+1)^2/2$. Thus, we have a contradiction. ◀

▶ **Lemma 23.** *Let $N \geq 2$. If $\mathcal{C}$ is a set of cliques that partition the edges of G_N such that $|\mathcal{C}| \leq N^2+N$, then $\mathcal{S} = \{V(C) \mid C \in \mathcal{C}\} \cup \{I\}$ is an FPP of order N over V. Moreover, the incidence matrix F of $\mathcal{S}$ with the column for I removed from it, is the BSD of the adjacency matrix of G_N that corresponds to $\mathcal{C}$.*

Proof. We will prove that $\mathcal{S} = \{V(C) \mid C \in \mathcal{C}\} \cup \{I\}$ satisfies the three properties in the definition of an FPP, which then has order N by Lemma 22 above. Property 1 follows easily from the definition of an edge clique partition: for each pair of adjacent vertices there is exactly one clique covering their edge, while any pair of non-adjacent vertices only appear in I.

Let us now prove Property 2. For any $S, S' \in \mathcal{S}$, it follows easily from the definition of an edge clique partition that $|S \cap S'| \leq 1$ (otherwise some edge is contained in two cliques). Also, for any $S \in \mathcal{S}$, it is true that $|S \cap I| \leq 1$ (otherwise some clique would contain a non-edge). Assume there are $S, S' \in \mathcal{S}$ with $S \cap S' = \emptyset$. By Lemma 22, we have $|S| = |S'| = N+1$, and so all the $(N+1)^2$ edges of $S \times S'$ have to be covered by distinct cliques (otherwise some clique would contain an edge already covered by one of the cliques induced by S or S'). But we do not have so many cliques as $|\mathcal{C}| \leq N^2+N$. Thus we have $|S \cap S'| = 1$ for any $S, S' \in \mathcal{S}$, and so Property 2 is satisfied.

Let us now prove Property 3. Consider any arbitrary clique $C \in \mathcal{C}$. Pick two vertices from $V(C) \setminus I$ and two vertices from $I \setminus V(C)$. Note that $|V(C) \setminus I| = |I \setminus V(C)| \geq N+1-1 = N \geq 2$, and hence two vertices can be picked from the sets. It is easy to see that out of these four vertices at most two are in any set in $\mathcal{S}$.

It is easy to see that the incidence matrix F of $\mathcal{S}$ minus the column for I is the BSD of the adjacency matrix of G_N that corresponds to the clique partition $\mathcal{C}$. ◀

By using Lemmas 21 and 23 and the fact that the element-set incidence matrix of an FPP has full rank [23], we prove Theorem 5, thereby giving the required lower bound on the number of ones in the basis matrix.

▶ **Fact 24.** *The element-set incidence matrix of any FPP has full rank [23].*

Proof of Theorem 5. Let N be a prime power and $k := N^2+N$. We will show that the adjacency matrix A of G_N has a width-k BSD and every basis of every width-k BSD of A has $\Theta(k^{3/2})$ ones. Note that A is a $(k+1) \times (k+1)$ binary matrix as stated in the theorem.

Since N is prime, there is an FPP of order N [17]. Then by Lemma 21, there is an edge clique partition of G_N with at most $k = N^2 + N$ cliques. Thus, the adjacency matrix A of G_N has a width-k BSD, by using the equivalence in Lemma 6.

Now, consider any width-k BSD B of A and $\tilde{B}$ be any basis of B. Then, by Lemma 6, there is an edge clique partition of G_N with at most k cliques. By Lemma 23, $\mathcal{S} = \{V(C) \mid C \in \mathcal{C}\} \cup \{I\}$ is an FPP of order N. Let F be the element-set incidence matrix of $\mathcal{S}$. By Lemma 23, B is equal to F minus the column in F corresponding to I. By Fact 24, F has full rank, i.e. it has rank $N^2 + N + 1 = k + 1$. This implies B has rank k, and hence has at least k columns. Since B is a width-k BSD, this means it has exactly k columns, and hence is a $(k+1) \times k$ matrix. Since B has rank k, we have that $\tilde{B}$ has k rows and k columns. Thus, $\tilde{B}$ is B minus some row of B. Since each column of B corresponds to a clique of $\mathcal{C}$ containing $N + 1$ vertices by Lemma 22, we have that B has $k(N + 1)$ ones. Hence the number of ones in $\tilde{B}$ is at least $k(N + 1) - k = \Theta(k\sqrt{k})$. ◀

5 Conclusion and Open Problems

We showed that AWECP admits a kernel with 4^k vertices, and an algorithm with a runtime of $2^{O(k^{3/2}w^{1/2}\log(k/w))}n^{O(1)}$, which implies that ECP can be solved in $2^{O(k^{3/2}\log k)}n^{O(1)}$ time. We think the following are the most interesting related open questions.

- Close the gap further between the upper and lower bounds on the running time for ECP that are currently $2^{\mathcal{O}(k^{3/2}\log k)}n^{\mathcal{O}(1)}$ and $2^{\Omega(k)}n^{\mathcal{O}(1)}$ respectively.
- Does WECP admit a polynomial-sized kernel like ECP?
- Can we show a tightness of analysis of our algorithm for WECP as we showed for ECP in Section 4, i.e., can we construct positive integer matrices with largest weight w that has a width-k BSD and every basis of every width-k BSD have $\Omega(k^{3/2}w^{1/2})$ ones?
- The algorithm of Chandran et al. [3] for Bipartite Biclique Partition with runtime $2^{O(k^2)}n^{O(1)}$ is also based on guessing the basis of a binary decomposition $A = BC$, and is currently the fastest FPT algorithm for the problem. If we can show that in any solution at least one of B and C has a row basis (column basis in case of C) with at most $g(k)$ ones, then we get a running time $2^{O(g(k)\log k)}n^{O(1)}$ using a similar algorithm as we gave for ECP. What is the minimum value of $g(k)$ possible?

References

1. Mathieu Blanchette, Ethan Kim, and Adrian Vetta. Clique cover on sparse networks. In *Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 93–102, 2012.
2. Parinya Chalermsook, Sandy Heydrich, Eugenia Holm, and Andreas Karrenbauer. Nearly tight approximability results for minimum biclique cover and partition. In *European Symposium on Algorithms*, pages 235–246. Springer, 2014.
3. L Sunil Chandran, Davis Issac, and Andreas Karrenbauer. On the parameterized complexity of biclique cover and partition. In *11th International Symposium on Parameterized and Exact Computation*, pages 1–13. Schloss Dagstuhl, 2017.
4. Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4 (8). Springer, 2015.
5. Marek Cygan, Marcin Pilipczuk, and Michał Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM Journal on Computing*, 45(1):67–83, 2016.

6 Andres Figueroa, James Borneman, and Tao Jiang. Clustering binary fingerprint vectors with missing values for dna array data analysis. *Journal of Computational biology*, 11(5):887–901, 2004.

7 Rudolf Fleischer and Xiaotian Wu. Edge Clique Partition of K4-Free and Planar Graphs. In *International Conference on Computational Geometry, Graphs and Applications*, pages 84–95. Springer, 2010.

8 Herbert Fleischner, Egbert Mujuni, Daniël Paulusma, and Stefan Szeider. Covering graphs with few complete bipartite subgraphs. *Theoretical Computer Science*, 410(21-23):2045–2053, 2009.

9 Floris Geerts, Bart Goethals, and Taneli Mielikäinen. Tiling databases. In *International conference on discovery science*, pages 278–289. Springer, 2004.

10 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Data reduction, exact, and heuristic algorithms for clique cover. In *2006 Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 86–94. SIAM, 2006.

11 Hermann Gruber and Markus Holzer. Inapproximability of nondeterministic state and transition complexity assuming p≠ np. In *International Conference on Developments in Language Theory*, pages 205–216. Springer, 2007.

12 Davis Issac. *On some covering, partition and connectivity problems in graphs*. PhD thesis, Saarland University, Saarbrücken, Germany, 2019. doi:10.22028/D291-29620.

13 Klaus Jansen, Felix Land, and Kati Land. Bounding the running time of algorithms for scheduling and packing problems. *SIAM Journal on Discrete Mathematics*, 30(1):343–366, 2016.

14 Viggo Kann. Maximum bounded 3-dimensional matching is max snp-complete. *Information Processing Letters*, 37(1):27–35, 1991.

15 L. T. Kou, L. J. Stockmeyer, and C. K. Wong. Covering edges by cliques with regard to keyword conflicts and intersection graphs. *Commun. ACM*, 21(2):135–139, 1978. doi:10.1145/359340.359346.

16 SH Ma, WD Wallis, and JL Wu. The complexity of the clique partition number problem. *Congr. Numer*, 67:59–66, 1988.

17 J Matoušek and J Nešetřil. *Invitation to Discrete Mathematics*. Oxford University Press, 2009.

18 Egbert Mujuni and Frances Rosamond. Parameterized complexity of the clique partition problem. In *Proceedings of the fourteenth symposium on Computing: the Australasian theory-Volume 77*, pages 75–78. Australian Computer Society, Inc., 2008.

19 Vladimir Nikiforov. A contribution to the zarankiewicz problem. *Linear algebra and its applications*, 432(6):1405–1411, 2010.

20 Blair Sullivan (University of Utah). Personal communication.

21 Subramanian Rajagopalan, Manish Vachharajani, and Sharad Malik. Handling irregular ilp within conventional vliw schedulers using artificial resource constraints. In *Proceedings of the 2000 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 157–164, 2000.

22 Fred S Roberts. Applications of edge coverings by cliques. *Discrete applied mathematics*, 10(1):93–109, 1985.

23 Howard Sachar. The f_p span of the incidence matrix of a finite projective plane. *Geometriae Dedicata*, 8(4):407–415, 1979.

24 Xiao-tian Wu, Yu-Hao Lin, and R Fleischer. Research of fixed parameter algorithm for clique partition problem. *Computer Engineering*, 37(11):92–93, 2011.

Parameterized Complexity of Deletion to Scattered Graph Classes

Ashwin Jacob
The Institute of Mathematical Sciences, HBNI, Chennai, India
ajacob@imsc.res.in

Diptapriyo Majumdar
Royal Holloway, University of London, UK
diptapriyo.majumdar@rhul.ac.uk

Venkatesh Raman
The Institute of Mathematical Sciences, HBNI, Chennai, India
vraman@imsc.res.in

Abstract

Graph-modification problems, where we add/delete a small number of vertices/edges to make the given graph to belong to a simpler graph class, is a well-studied optimization problem in all algorithmic paradigms including classical, approximation and parameterized complexity. Specifically, graph-deletion problems, where one needs to delete at most k vertices to place it in a given non-trivial hereditary (closed under induced subgraphs) graph class, captures several well-studied problems including Vertex Cover, Feedback Vertex Set, Odd Cycle Transveral, Cluster Vertex Deletion, and Perfect Deletion. Investigation into these problems in parameterized complexity has given rise to powerful tools and techniques. While a precise characterization of the graph classes for which the problem is *fixed-parameter tractable* (FPT) is elusive, it has long been known that if the graph class is characterized by a *finite* set of forbidden graphs, then the problem is FPT.

In this paper, we initiate a study of a natural variation of the problem of deletion to *scattered graph classes* where we need to delete at most k vertices so that in the resulting graph, each connected component belongs to one of a constant number of graph classes. A simple hitting set based approach is no longer feasible even if each of the graph classes is characterized by finite forbidden sets. As our main result, we show that this problem (in the case where each graph class has a finite forbidden set) is fixed-parameter tractable by a $O^*(2^{k^{\mathcal{O}(1)}})$[1] algorithm, using a combination of the well-known techniques in parameterized complexity – *iterative compression* and *important separators*. Our approach follows closely that of a related problem in the context of satisfiability [Ganian, Ramanujan, Szeider, TAlg 2017], where one wants to find a small backdoor set so that the resulting CSP (constraint satisfaction problem) instance belongs to one of several *easy* instances of satisfiability. While we follow the main idea from this work, there are some challenges for our problem which we needed to overcome.

When there are two graph classes with finite forbidden sets to get to, and if one of the forbidden sets has a path, then we show that the problem has a (better) singly exponential algorithm and a polynomial sized kernel. We also design an efficient FPT algorithm for a special case when one of the graph classes has an infinite forbidden set. Specifically, we give a $O^*(4^k)$ algorithm to determine whether k vertices can be deleted from a given graph so that in the resulting graph, each connected component is a tree (the sparsest connected graph) or a clique (the densest connected graph).

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Parameterized Complexity, Scattered Graph Classes, Important Separators

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.18

Acknowledgements The authors thank M.S.Ramanujan for many helpful discussions.

[1] The O^* notation ignores polynomial factors.

1 Introduction

Graph modification problems, where we want to modify a given graph by adding/deleting vertices/edges to obtain a *simpler* graph are well-studied problems in algorithmic graph theory. Starting from the classical work of Lewis and Yannakakis [12] (see also [20]) who showed the problem NP-complete for the resulting simpler graph belonging to any non-trivial (the graph property is true for infinitely many graphs and false for infinitely many graphs) hereditary graph class (closed under induced subgraphs), the complexity of the problem has been studied in several algorithmic paradigms including approximation and parameterized complexity. Specifically, deleting at most k vertices to a fixed hereditary graph class is an active area of research in parameterized complexity over the last several years yielding several powerful tools and techniques. Examples of such problems include Vertex Cover, Cluster Vertex Deletion, Feedback Vertex Set and Chordal Vertex Deletion.

It is well-known that any hereditary graph class can be described by a forbidden set of graphs, finite or infinite, that contains all minimal forbidden graphs in the class. In parameterized complexity, it is known that the deletion problem is fixed-parameter tractable (FPT) as long as the resulting hereditary graph class has a finite forbidden set [3]. This is shown by an easy reduction to the Bounded Hitting Set problem. This includes, for example, deletion to obtain a split graph or a cograph. We also know FPT algorithms for specific graph classes defined by infinite forbidden sets like feedback vertex set and odd cycle transversal [6]. While the precise characterization of the class of graphs for which the deletion problem is FPT is elusive, there are graph classes for which the problem is W-hard [10, 13].

Recently, some stronger versions have also been studied, where the problem is to delete at most k vertices to get a graph such that every connected component of the resulting graph is at most ℓ edges away from being a graph in a graph class $\mathcal{F}$ (see [16–18]). Some examples of $\mathcal{F}$ that have been studied in this stronger version include *forest, pseudo-forest or bipartite.*

Our results: In this paper, we address the complexity of a very natural variation of the graph deletion problem, where in the resulting graph, each connected component belongs to one of finitely many graph classes. For example, we may want the connected components of the resulting graph to be a clique or a biclique (a complete bipartite graph). It is known that cliques forbid exactly P_3s, the induced paths of length 2, and bicliques forbid P_4 and triangles. So if we just want every connected component to be a clique or every connected component to be a biclique, then one can find appropriate constant sized subgraphs in the given graph and branch on them (as one would in a hitting set instance). However, if we want each connected component to be a clique or a biclique, such a simple approach by branching over P_3, P_4, or K_3 would not work. Notice that triangles are allowed to be present in clique components and P_3s are allowed to be present in biclique components. It is not even clear that there will be a finite forbidden set for this resulting graph class.

Our main result of the paper shows this deletion problem, when there are a constant number of graph classes each characterized by a finite forbidden set for the resulting graph is fixed-parameter tractable using the well-known techniques in parameterized complexity – iterative compression and important separators. Specifically the problem we show to be fixed-parameter tractable is the following.

$(\Pi_1, \Pi_2, \ldots, \Pi_d)$ Vertex Deletion **Parameter:** k
Input: An undirected graph $G = (V, E)$, an integer k, and d graph classes $\Pi_1, \ldots, \Pi_d$ described by finite forbidden sets $\mathcal{F}_1, \ldots, \mathcal{F}_d$ respectively.
Question: Is there a subset $Z \subseteq V(G), |Z| \leq k$ such that every connected component of $G - Z$ is in at least one of the graph classes $\Pi_1, \ldots, \Pi_d$?

Several natural graph classes forbid induced paths of certain lengths. We can develop a much better algorithm in that case. More specifically, when there are only two graph classes for the connected components of the resulting graph to be placed, and if one of them has a path as a forbidden subgraph, then we show that the problem has a polynomial kernel, an efficient $O^*(c^k)$ algorithm and a c-approximation algorithm where c depends on the maximum size of the graphs in the forbidden set. Finally we also look at the problem for the specific case when one of the resulting graph classes has an infinite forbidden set. Specifically, we look at the problem of deleting a small number of vertices so that each connected component of the resulting graph is a tree (the sparsest connected graph) or a clique (the densest connected graph), and give an $O^*(4^k)$ algorithm.

Previous Work. While there has been a lot of work on graph deletion and modification problems, one work that comes close to ours is the work by Ganian, Ramanujan and Szeider [9] where they consider the parameterized complexity of finding strong backdoors to a scattered class of CSP instances. In fact, in their conclusion, they remark that

> "graph modification problems and in particular the study of efficiently computable modulators to various graph classes has been an integral part of parameterized complexity and has led to the development of several powerful tools and techniques. We believe that the study of modulators to "scattered graph classes" could prove equally fruitful and, as our techniques are mostly graph based, our results as well as techniques could provide a useful starting point towards future research in this direction".

Our work is a starting point in addressing the parameterized complexity of the problem they suggest.

Our Techniques. We now give a brief summary of the main FPT algorithm described in Theorem 3.3. We reduce the problem $(\Pi_1, \Pi_2, \ldots, \Pi_d)$ Vertex Deletion to Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$ Vertex Deletion Compression (Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC for short) using the standard technique of iterative compression. In Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC, we can assume that a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator W of the graph of size $k - i$ is also given to us as input for some $i \leq k$ (i is the number of vertices from the modulator we have guessed to be in the solution) and the aim is to check if there is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator of the graph of size $k - i - 1$ disjoint from W. This is formally described in Subsection 3.1.

In Subsection 3.2, we give an FPT algorithm for Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC in the special case when the solution that we are looking for leaves W in a single component. The algorithm uses the technique of important separators [15].

Finally in Subsection 3.3, we handle general instances of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. We focus on instances where the solution separates W. We guess $W_1 \subset W$ as the part of W that occurs in some single connected component after deleting the solution. Since, the solution separates W, we know that it contains a $W_1 - (W \setminus W_1)$ separator X. The algorithm uses the technique of important separator sequences [14]. Informally, an important separator sequence partitions the graph into slices with small boundaries which allows us to look for solutions local to the slices. The algorithm guesses the integer ℓ which is the size of the part of the solution present in the graph containing W_1 after removing X. The algorithm then constructs the important separator sequence corresponding to ℓ and finds the separator P furthest from W_1 in the sequence such that there is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator of size ℓ in the graph containing W_1 after removing P. If separator X either intersects P or dominates

the other, then a recursive smaller instance is easily constructable. In the case when the two separators are incomparable, the algorithm identifies a set of vertices Y that is reachable from W_1 after deleting P. The algorithm then constructs a graph gadget of $k^{\mathcal{O}(1)}$ vertices whose appropriate attachment to the boundary P of the graph $G[Y]$ gives a graph G' which preserves the part of the solution of G present in $G[Y]$. Since this part of the solution is strictly smaller in size, the algorithm can find the solution for G by recursively finding the solution in G'.

While the main techniques for a good part follow from the paper by Ganian, Ramanujan and Szeider [9], the problem has its own challenges. The paper by Ganian, Ramanujan and Szeider worked on strong backdoors to a scattered class of CSPs. The authors create an variable-constraint incidence graph based on the CSP and focus on "forbidden sets" with respect to a set of variables S which is a set of constraints C such that there is an assignment of S on which for every CSP class, one of the constraints under this assignment does not belong to this class. We identify an equivalent notion of forbidden set in our problem as a subset of vertices C such that for every finite forbidden graph class, there is a subset of C such that the graph induced on the subset belongs to a forbidden member of this class. Since the forbidden set is identified by a collection of finite subgraphs here, instead of a set of constraint vertices in the CSP case where the graph properties are not relevant, there are more difficulties arising from it. For example, the CSP result uses a connecting gadget that adds extra vertices and edges to preserve connectivity of some subsets. They manage this by associating tautological constraints to these extra vertices which can go into any of the CSP classes. But adding new vertices and edges can create new subgraphs which could be a finite forbidden subgraph, in our case. Due to this, we had to come up with an algorithm avoiding any use of connecting gadgets. Another difficulty is in creating the smaller graph instance G' using gadgets as described in the previous paragraph where we could recursively solve the instance. In the CSP case, identifying the set of vertices that need to be preserved in the smaller instance was easier as the forbidden sets were a set of constraint vertices. In our case, this gets more complicated as we deal with forbidden sets involving subgraphs. Specifically this results in a more complex marking procedure in Lemma 3.17 to prove its third claim.

2 Preliminaries

Graph Theory. For $\ell \in \mathbb{N}$, we use P_ℓ to denote the path on ℓ vertices. We use standard graph theoretic terminology from Diestel's book [8]. A *tree* is a connected graph with no cycles. A *forest* is a graph, every connected component of which is a tree. A *paw* is a graph G with vertex set $V(G) = \{x_1, x_2, x_3, x_4\}$ and edge set $E(G) = \{x_1x_2, x_2x_3, x_3x_1, x_3x_4\}$. A graph is a *block graph* if all its biconnected components are cliques. For a set $X \subseteq G$, we use $G[X]$ to denote the graph induced on the vertex set X and we use $G - X$ (or $G \setminus X$) to denote the graph induced by the vertex set $V(G) \setminus X$. We say that a subset $Z \subseteq V(G)$ *disconnects* a subset $S \subseteq V(G)$ if there exists $v, w \in S$ with $v \neq w$ such that v and w occur in different connected components of the graph $G \setminus Z$.

▶ **Definition 2.1.** *Let G be a graph and disjoint subsets $X, S \subseteq V(G)$. We denote by $R_G(X, S)$ the set of vertices that lie in the connected component containing X in the graph $G \setminus S$. We denote $R_G[X, S] = R_G(X, S) \cup S$. Finally we denote $NR_G(X, S) = V(G) \setminus R_G[X, S]$ and $NR_G[X, S] = NR_G(X, S) \cup S$. We drop the subscript G if it is clear from the context.*

▶ **Definition 2.2** ([15])**.** *Let G be a graph and $X, Y \subseteq V(G)$.*

- *A vertex set S disjoint from X and Y is said to disconnect X and Y if $R_G(X, S) \cap Y = \phi$. We say that S is an $X - Y$ **separator** in the graph G.*

- *An $X - Y$ separator is **minimal** if none of its proper subsets is an $X - Y$ separator.*
- *An $X - Y$ separator S_1 is said to **cover** an $X - Y$ separator S with respect to X if $R(X, S) \subset R(X, S_1)$.*
- *Two $X - Y$ separators S_1 and S_2 are said to be incomparable if neither covers the other.*
- *In a set $\mathcal{H}$ of $X - Y$ separators, a separator S is said to be component-maximal if there is no separator S' in $\mathcal{H}$ which covers S. Component-minimality is defined analogously.*
- *An $X - Y$ separator S_1 is said to dominate an $X - Y$ separator S with respect to X if $|S_1| \leq |S|$ and S_1 covers S with respect to X.*
- *We say that S is an important $X - Y$ separator if it is minimal and there is no $X - Y$ separator dominating S with respect to X.*

For the basic definitions of Parameterized Complexity, we refer to [6].

3 Deletion to d finite scattered classes

▶ **Definition 3.1.** *We call a set Z a* **$(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator** *if every connected component of $G \setminus Z$ is in one of the graph classes Π_i for $i \in d$.*

Organization of the section. This section is divided into three subsections. In the first section, we use iterative compression to transform the $(\Pi_1, \Pi_2, \ldots, \Pi_d)$ Vertex Deletion problem into a compressed version of the problem named Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. In the second section, we give an algorithm for Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC for the special case of having a non-separating solution which we will define later. Finally in the third section, we give a general algorithm for Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC and subsequently $(\Pi_1, \Pi_2, \ldots, \Pi_d)$ Vertex Deletion using the algorithm in the second section as a subroutine.

3.1 Iterative Compression

We use the standard technique of iterative compression to transform the $(\Pi_1, \Pi_2, \ldots, \Pi_d)$ Vertex Deletion problem into the following problem Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$ Vertex Deletion Compression(Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC) such that an FPT algorithm with running time $\mathcal{O}^*(f(k))$ for the latter gives a $\mathcal{O}^*(2^{k+1} f(k))$ time algorithm for the former. The details are moved to the full version.

Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC	**Parameter:** k
Input: A graph G, an integer k, finite forbidden sets $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_d$ for graph classes $\Pi_1, \Pi_2, \ldots, \Pi_d$ and a subset W of $V(G)$ such that W is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$- modulator of size $k + 1$.	
Question: Is there a subset $Z \subseteq V(G) \setminus W, \lvert Z\rvert \leq k$ such that Z is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator of the graph G?	

3.2 Finding non-separating solutions

In this section, we focus on solving instances of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC which have a non-separating property defined as follows.

▶ **Definition 3.2.** *Let (G, k, W) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *and Z be a solution for this instance. Then Z is called a **non-separating** solution if W is contained in a single connected component of $G \setminus Z$ and **separating** otherwise. If an instance has only separating solutions, we call it a separating instance. Otherwise, we call it non-separating.*

We begin the description of the algorithm with the following reduction rule.

▶ **Reduction Rule 1.** *If a connected component of G belongs to some graph class Π_i, then remove all the vertices of this connected component.*

▶ **Lemma 3.3** ($\star$[2])**.** *Reduction rule 1 is safe.*

We now develop the following notion of forbidden sets which can be used to identify if a connected component of a graph belongs to any of the classes Π_i for $i \in [d]$.

▶ **Definition 3.4.** *We say that a subset of vertices $C \subseteq V(G)$ is a **forbidden set** of G if C occurs in a connected component of G and there exists a subset $C_i \subseteq C$ such that $G[C_i] \in \mathcal{F}_i$ for all $i \in [d]$ and C is a minimal such set.*

Clearly if a connected component of G contains a forbidden set, then it does not belong to any of the graph classes Π_i for $i \in [d]$. We note that even though the forbidden set C is of finite size, the lemma below rules out the possibility of a simple algorithm involving just branching over all the vertices of C.

▶ **Lemma 3.5** ($\star$)**.** *Let G be a graph and $C \subseteq V(G)$ be a forbidden set of G. Let Z be a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator of G. Then Z disconnects C or $Z \cap C \neq \emptyset$.*

We now have the following lemma on important separators which is helpful in our algorithm to compute non-separating solutions.

▶ **Lemma 3.6** ([4])**.** *For every $k \geq 0$ and subsets $X, Y \subseteq V(G)$, there are at most 4^k important $X - Y$ separators of size at most k. Furthermore, there is an algorithm that runs in $\mathcal{O}(4^k kn)$ time that enumerates all such important $X - Y$ separators and there is an algorithm that runs in $n^{\mathcal{O}(1)}$ time that outputs one arbitrary component-maximal $X - Y$ separator.*

We now have the following lemma which connects the notion of important separators with non-separating solutions of our problem.

▶ **Lemma 3.7** ($\star$)**.** *Let (G, k, W) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *obtained after exhaustively applying Reduction Rule 1 and Z be a non-separating solution. Let v be a vertex such that Z is a $\{v\} - W$ separator. Then there is a solution Z' which contains an important $v - W$ separator of size at most k in G.*

We use the above lemma along with Lemma 3.6 to obtain our algorithm for non-separating instances. The algorithm finds a minimal forbidden set C in polynomial time which is finite. Then it branches on the set C and also on $v - W$ important separators of size at most k of G for all $v \in C$.

▶ **Lemma 3.8.** *Let (G, k, W) be a non-separating instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC*. Then it can be solved in $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ time where $|V| = n$.*

Proof. We first apply Reduction Rule 1 exhaustively. If the graph is empty, we return YES. Else, there is a connected component of G which does not belong to any graph classes Π_i for $i \in [d]$. Therefore, there exists a forbidden set $C \subseteq V(G)$ of G present in this connected component. We find C by checking for each graph class Π_i, if a graph in $\mathcal{F}_i$ exists as an induced subgraph in a any connected component $\mathcal{X}$ of G, take the union of these vertices and make it minimal by removing vertices and seeing if the set still remains forbidden. We

[2] The proofs of theorems and lemmas marked ($\star$) are moved to the full version due to lack of space.

branch in $|C|$-many ways by going over all the vertices $v \in C$ and in each branch, recurse on the instance $(G - v, k - 1, W)$. Then for all $v \in C$, we branch over all important $v - W$ separators X of size at most k in G and recurse on instances $(G \setminus X, k - |X|, W)$.

We now prove the correctness of the algorithm. Let Z be a solution of the instance. From Lemma 3.5, we know that a forbidden set C of G is disconnected by Z or $Z \cap C \neq \phi$. In the latter case, we know that Z contains a vertex $x \in C$ giving us one of the branched instances obtained by adding x into the solution.

Now we are in the case when C is disconnected by Z. Since Reduction rule 1 is applied exhaustively, the connected component containing C also contains some vertices in W. Since Z is a non-separating solution, W goes to exactly one connected component of $G \setminus Z$ and there exists some non-empty part of C that is not in this component. Hence, there exists some vertex $x \in C$ that gets disconnected from W by Z. From Lemma 3.7, we know that there is also a solution Z' which contains an important $x - W$ separator of size at most k in G. Since we have branched over all such $x - W$ important separators, we have correctly guessed on one such branch.

We now bound the running time. Since $|C| = \mathcal{O}(d)$, any forbidden set in G can be obtained via brute force in $n^{\mathcal{O}(d)}$ time. For each $i \in [k]$, we know that there are at most 4^i important separators of size $1 \leq i \leq k$ which can be enumerated using Lemma 3.6 in $\mathcal{O}(4^i \cdot i \cdot n)$ time. For the instance (G, k, W), if we branch on $v \in C$, k drops by 1 and if we branch on a $v - W$ separator of size i, k drops by i. Hence if $T(k)$ denotes the time taken for the instance (G, k, W), we get the recurrence relation $T(k) = \mathcal{O}(d)T(k-1) + \sum_{i=1}^{k} 4^i T(k-i)$ upon solving with taking into account that d is a constant, we get that $T(k) = 2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$. ◀

3.3 Solving general instances

We now solve general instances of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC using the algorithm for solving non-separating instances as a subroutine. We first focus on solving separating instances of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. We guess a subset $W_1 \subset W$ such that for a solution Z, W_1 is exactly the intersection of W with a connected component of $G \setminus Z$. For $W_2 = W \setminus W_1$, we are looking for a solution Z containing a $W_1 - W_2$ separator. Formally, let $W = W_1 \uplus W_2$ be a set of size $k+1$ which is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator. We look for a set $Z \subseteq V(G) \setminus W$ of size at most k such that Z is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator, Z contains a minimal (W_1, W_2)-separator X and W_1 occurs in a connected component of $G \setminus Z$.

From here on, we assume that the separating instance (G, k, W) of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC is represented as (G, k, W_1, W_2) where $W = W_1 \uplus W_2$. We branch over all partitions of W into W_1 and W_2 which adds a factor of 2^{k+1} to the running time. We now introduce the notion of tight separator sequences and t-boundaried graphs which are used to design the algorithm.

3.3.1 Tight Separator Sequences

We first look at a type of $W_1 - W_2$ separators where the graph induced on the vertices reachable from W_1 satisfy the property as defined below.

▶ **Definition 3.9.** *Let (G, k, W_1, W_2) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. *We call a $W_1 - W_2$ separator X in G **ℓ-good** if there exists a set K of size at most ℓ such that $K \cup X$ is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator for the graph $G[R[W_1, X]]$. Else we call it **ℓ-bad**.*

▶ **Lemma 3.10** (⋆). *Let (G, k, W_1, W_2) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *and let X and Y be disjoint $W_1 - W_2$ separators in G such that X covers Y. If X is ℓ-good, then Y is also ℓ-good.*

▶ **Definition 3.11.** *Let (G, k, W_1, W_2) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *and let X and Y be $W_1 - W_2$ separators in G such that Y dominates X. Let ℓ be the smallest integer i for which X is i-good. If Y is ℓ-good, then we say that Y* ***well-dominates*** *X. If X is ℓ-good and there is no $Y \neq X$ which well-dominates X, then we call X* ***ℓ-important****.*

The following lemma allows us to focus on solutions containing an ℓ-important $W_1 - W_2$ separator for some appropriate value of ℓ.

▶ **Lemma 3.12.** *Let (G, k, W_1, W_2) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *and Z be a solution. Let $P \subseteq Z$ be a non-empty minimal $W_1 - W_2$ separator in G and let P' be a $W_1 - W_2$ separator in G well-dominating P. Then there is also a solution Z' for the instance containing P'.*

Proof. Let $Q = Z \cap R[W_1, P]$. Note that Q is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator for the graph $G[R[W_1, P]]$. Let $Q' \supseteq P'$ be a smallest $(\Pi_1, \Pi_2, \ldots, \Pi_d)$- modulator for the graph $G[R[W_1, P']]$ extending P'. We claim that $Z' = (Z \setminus Q) \cup Q'$ is a solution for (G, k, W_1, W_2). Since P' well-dominates P, $|Z'| \leq |Z|$. We now show that Z' is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator. Suppose not. Then there exists a forbidden subset C present in a connected component $\mathcal{X}$ of $G \setminus Z'$.

We first consider the case when $\mathcal{X}$ is disjoint from the set $Z \setminus Z'$. Then there is a component $\mathcal{H}$ in $G \setminus Z$ which contains $\mathcal{X}$ and hence C, contradicting that Z is a solution. We now consider the case when $\mathcal{X}$ intersects $Z \setminus Z'$. By definition of Z', $\mathcal{X}$ is contained in the set $R(W_1, P')$. Since $Z' \setminus Q'$ is disjoint from $R(W_1, P')$ and is separated from $R(W_1, P')$ by just P', we can conclude that $\mathcal{X}$ and hence C is contained in a single connected component of $G[R[W_1, P']] \setminus Q'$. But this contradicts that Q' is not a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator in the graph $G[R[W_1, P']]$. ◀

We now define the notion of tight separator sequence. It gives a natural way to partition the graph into parts with small *boundaries*. This helps us focus our problem on local parts of the graph which eases the task of solving it.

▶ **Definition 3.13.** *An $X - Y$* ***tight separator sequence of order k*** *of a graph G with $X, Y \subseteq V(G)$ is a set $\mathcal{H}$ of $X - Y$ separators such that every separator has size at most k, the separators are pairwise disjoint, for any pair of separators in the set, one covers another and the set is maximal with respect to the above properties.*

▶ **Lemma 3.14 ($\star$).** *Given a graph G, disjoint vertex sets X, Y and integer k, a tight separator sequence $\mathcal{H}$ of order k can be computed in $|V(G)|^{\mathcal{O}(1)}$ time.*

3.3.2 Boundaried graphs

▶ **Definition 3.15.** *A* ***t-boundaried graph*** *G be a graphs with t distinguished labelled vertices. We call the set of labelled vertices $\partial(G)$ the boundary of G and the vertices in $\partial(G)$ terminals. Let G_1 and G_2 be two t-boundaried graphs with the graphs $G_1[\partial(G_1)]$ and $G_2[\partial(G_2)]$ being isomorphic. Let $\mu : \partial(G_1) \to \partial(G_2)$ be a bijection which is an isomorphism of the graphs $G_1[\partial(G_1)]$ and $G_2[\partial(G_2)]$. We denote the graph $G_1 \otimes_\mu G_2$ as a t-boundaried graph obtained by the following gluing operation. We take the union of graphs G_1 and G_2 and identify each vertex $x \in \partial(G_1)$ with vertex $\mu(x) \in \partial(G_2)$. The t-boundary of the new graph is the set of vertices obtained by unifying.*

▶ **Definition 3.16.** *A* ***t-boundaried graph with an annotated set*** *is a t-boundaried graph with a second set of distinguished but unlabelled vertices disjoint from the boundary. The set of annotated vertices is denoted by $\Delta(G)$.*

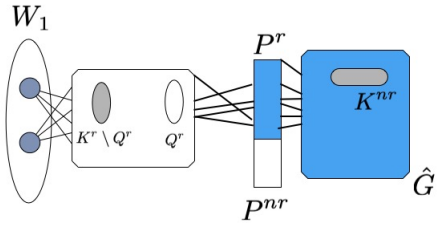


Figure 1 The graph G' obtained by gluing some $\hat{G} \in \mathcal{H}$ to $G[R[W_1, P]]$.

We now prove the following crucial lemma for giving the algorithm for solving separating instances of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. Suppose we are looking at an instance of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC which has a solution Z. We know that the solution Z contains a minimal $W_1 - W_2$ separator Q which is incomparable to a given ℓ-good $W_1 - W_2$ separator P in G. Then some part of Z, say K, lies in the set $R[W_1, Q]$. We show that by carefully replacing parts outside of $R[W_1, P]$ with a small gadget, we can get a smaller graph G' which preserves the part of K inside the set $R[W_1, P]$. Also we show that there is some part of K lying outside the set $R[W_1, P]$ and hence the size of the solution in G' is strictly smaller than in G.

The gadget is loosely speaking a set obtained by a marking procedure which preserves all possible types of forbidden sets in the original graph. Let us look at a forbidden set C and a subset $C_i \subseteq C$ such that $G[C_i] = H_i \in \mathcal{F}_i$. Let us look at the subset $C'_i \subseteq C_i$ that lies outside $R(W_1, P)$, the corresponding graph being an induced subgraph of H_i. For all $i \in [d]$ and all possible induced subgraphs of H_i, we mark sets of vertices outside $R(W_1, P)$ such that there is a corresponding forbidden set C in the graph G whose intersections with $NR[W_1, P]$ are exactly these graphs. Let G' be the graph formed by removing the unmarked vertices outside $R(W_1, P)$. For any forbidden set C in G, we can replace parts of C_i outside $R(W_1, P)$ with the corresponding marked vertices and get a forbidden set C' in graph G'. This allows us to focus on the smaller graph G' and use the solution in G' to construct a solution in G.

▶ **Lemma 3.17** ($\star$). *Let (G, k, W_1, W_2) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *and let Z be a solution. Let $Q \subseteq Z$ be a minimal $W_1 - W_2$ separator in the graph G. Let $K = Z \cap R[W_1, Q]$ with $\ell = |K \setminus Q|$. Let P be a minimal $W_1 - W_2$ separator in G which is disjoint from K and incomparable to Q. Let $Q^r = Q \cap R(W_1, P)$ and $Q^{nr} = Q \setminus Q^r$. Similarly, let $P^r = P \cap R(W_1, Q)$ and $P^{nr} = P \setminus P^r$. Let $K^r = K \cap R[W_1, P]$. Let $G_1 = G[R[W_1, P]]$ be a boundaried graph with P^r as the boundary.*

Then there exists a $|P^r|$-boundaried graph $\hat{G}$ which is $k^{\mathcal{O}(1)}$ in size with an annotated set of vertices, and a bijection $\mu : \partial(\hat{G}) \to P^r$ such that the glued graph $G' = G_1 \otimes_\mu \hat{G}$ has the following properties (see Figure 1).

1. *The set W_1 is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator for the graph G'.*
2. *The set Q^r is a $|K^r \setminus Q^r|$-good $W_1 - P^{nr}$ separator in the graph $G' \setminus \Delta(\hat{G})$.*
3. *For any Q' which is a $W_1 - P^{nr}$ separator in $G' \setminus \Delta(\hat{G})$ well-dominating Q^r in G', the set $Q' \cup Q^{nr}$ well dominates the $W_1 - W_2$ separator Q in G.*
4. *There is a a family $\mathcal{H}$ of boundaried graphs with an annotated set of vertices such that $\mathcal{H}$ contains $\hat{G}$, has size bounded by $2^{k^{\mathcal{O}(1)}}$ and can be computed in $2^{k^{\mathcal{O}(1)}} n^{O(1)}$.*

3.3.3 Algorithm for general instances

The following lemma focuses on the particular case when W_1 and W_2 are already disconnected in G.

▶ **Lemma 3.18** ($\star$). *Let (G, k, W_1, W_2) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *where W_1 and W_2 are in distinct components of G. Let Z be its solution such that W_1 exactly occurs in a connected component of $G \setminus Z$. Also let $R(W_1)$ be the set of vertices reachable from W_1 in G. Let $Z' = Z \cap R(W_1)$. Then $(G[R(W_1)], |Z'|, W_1)$ is a non-separating YES-instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *and conversely for any non-separating solution Z'' for $(G[R(W_1)], |Z'|, W_1)$, the set $\hat{Z} = (Z \setminus Z') \cup Z''$ is a solution for the original instance such that W_1 exactly occurs in a connected component of $G \setminus Z''$.*

We have the following reduction rule.

▶ **Reduction Rule 2.** *Let (G, k, W_1, W_2) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *where W_1 and W_2 are disconnected in G. Compute a non-separating solution Z' for the instance (G', k', W_1) where $G' = G[R(W_1)]$ and k' is the least integer $i \leq k$ for which (G', i, W_1) is a YES-instance. Delete Z' and return the instance $(G \setminus Z', k - |Z'|, W_2)$.*

The safeness of Reduction Rule 2 comes from Lemma 3.18. The running time for the reduction is $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ which comes from that of the algorithm in Lemma 3.8. Henceforth, we assume that Reduction Rule 2 is no longer applicable. We know that every solution Z of (G, k, W_1, W_2) contains an ℓ-good non-empty $W_1 - W_2$ separator X in the graph G. Let us denote Main-Algorithm(G, k, W_1, W_2) as the main algorithm procedure. We now describe a subroutine Branching-Set$((G, k, W_1, W_2), \lambda, \ell)$ which is used in Main-Algorithm where $0 \leq \ell \leq k$ and $1 \leq \lambda \leq k$.

▶ **Lemma 3.19.** *Let (G, k, W_1, W_2) be an instance of* Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *and let $0 \leq \ell \leq k$ and $1 \leq \lambda \leq k$. There is an algorithm Branching-Set$((G, k, W_1, W_2), \lambda, \ell)$ that returns a set $\mathcal{R}$ containing at most $2^{k^{\mathcal{O}(1)}}$ vertices such that for every solution Z of the given instance containing an ℓ-important $W_1 - W_2$ separator X of size at most λ in G, the set $\mathcal{R}$ intersects Z.*

Proof. Since Reduction Rule 2 is applied exhaustively, there is a $W_1 - W_2$ path in the graph G. If there is no $W_1 - W_2$ separator of size λ in the graph G, we declare the tuple invalid. Else we execute Lemma 3.14 to obtain a tight $W_1 - W_2$ separator sequence $\mathcal{I}$ of order λ. We then partition $\mathcal{I}$ into ℓ-good and ℓ-bad separators. For this we need a procedure to check whether a given separator P is ℓ-good or not. If both $\lambda, \ell < k$, we do so by recursively calling the main algorithm procedure Main-Algorithm$(G[R[W_1, P]], \ell, W_1, P)$. The recursive call is possible as ℓ is strictly less than k. We note that since $\lambda \geq 1$, the cardinality of P is non-zero. From the definition of ℓ-good separators, we can conclude that $\ell < k$ as the solution set in $G[R[W_1, P]]$ is at most k and it contains non-empty set P as its subset.

Let P_1 be component maximal among all the good separators in $\mathcal{I}$ if any exists and P_2 be component minimal among all the bad separators in $\mathcal{I}$ if any exists. We set $\mathcal{R} := P_1 \cup P_2$. For $i \in \{1, 2\}$, we do the following.

We execute the algorithm of Lemma 3.17, Claim 4 to compute a family $\mathcal{H}$ of boundaried graphs with an annotated set of vertices. Then for every choice of $P_i^r \subseteq P_i$, for every instance $\hat{G} \in \mathcal{H}$ with $|P_i^r|$ terminals and every possible bijection $\delta : \partial(\hat{G}) \to P_i^r$, we construct the glued graph $G_{P_i^r, \delta} = G[R[W_1, P_i]] \otimes_\delta \hat{G}$, where the boundary of $G[R[W_1, P_i]]$ is P_i^r.We then recursively call Branching-Set$((G_{P_i^r, \delta} \setminus \tilde{S}, k - j, W_1, P_i \setminus P_i^r), \lambda', \ell')$ for every $0 \leq \lambda' < \lambda$, $1 \leq j \leq k - 1$ and $0 \leq \ell' \leq \ell$, where $\tilde{S}$ is the set of annotated vertices in $\hat{G}$. We add the union of all the vertices returned by these recursive instances to $\mathcal{R}$ and return the resulting set.

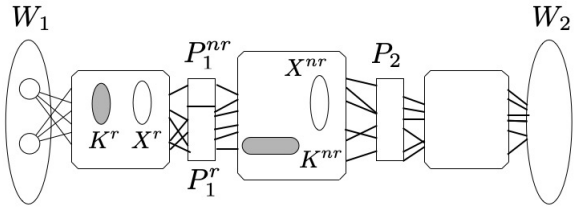


Figure 2 The case where X is incomparable with P_1.

This completes the description of the algorithm. We now proceed to the proof of correctness.

Correctness: We prove by induction on λ. We first consider the base case when $\lambda = 1$ where there is a $W_1 - W_2$ ℓ-good separator $X \subseteq Z$ of size one. Since X has size one, it cannot be incomparable with the separator P_1. Hence either X is equal to P_1 or is covered by P_1. In either case, we are correct as P_1 is contained in $\mathcal{R}$. We now assume that the algorithm correctly runs for all tuples where $\lambda < \hat{\lambda}$ for some $\hat{\lambda} \geq 2$. We now look at the case when the algorithm runs on a tuple with $\lambda = \hat{\lambda}$.

Let Z be a solution for the instance containing an ℓ-important separator X. If X intersects $P_1 \cup P_2$ we are done as $\mathcal{R}$ intersects X. Hence we assume that X is disjoint from $P_1 \cup P_2$. Suppose X is covered by P_1. Then we conclude that P_1 well-dominates X contradicting that X is ℓ-important.

By Lemma 3.10, since X is ℓ-good and P_2 is not, X cannot cover P_2. Suppose X covers P_1 and itself is covered by P_2. Then X must be contained in the maximal tight separator sequence $\mathcal{I}$ contradicting that P_1 is component maximal.

Finally we are left with the case where X is incomparable with P_1 or P_2 if P_1 does not exist. Without loss of generality, assume X is incomparable with P_1. The argument in the case when P_1 does not exist follows by simply replacing P_1 with P_2 in the proof.

Let $K \subseteq Z$ be a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator for the graph $G[R[W_1, X]]$ extending X. If $P_1 \cap K$ is empty, we have that $P_1 \cap Z$ is empty. Since P_1 is contained in $\mathcal{R}$, the algorithm is correct as $\mathcal{R}$ intersects Z. Hence assume that P_1 and K are disjoint.

Let $X^r = R(W_1, P_1) \cap X$ and $X^{nr} = X \setminus X^r$. Similarly, define $P_1^r = R(W_1, X) \cap P_1$ and $P_1^{nr} = P_1 \setminus P_1^r$. Since X and P_1, the sets X^r, X^{nr}, P_1^r and P_1^{nr} are all non-empty. Let $K^r = K \cap R(W_1, P_1)$. If there is a vertex in P_1^r that is not in the same connected component as W_1 in $G \setminus Z$, then as X separates P_1^r from W_2, we can conclude that $\mathcal{R}$ contains a vertex which is separated from W by Z implying that the algorithm is correct. Hence assume that P_1^r is contained in the same connected component as W_1 in the graph $G \setminus Z$.

We observe that the sets defined above satisfy the conditions for Lemma 3.17 with $P = P_1$ and $Q = X$. Therefore there exists a $|P_1^r|$-boundaried graph $\hat{G}$ with an annotated set $\tilde{S}$ and an appropriate bijection $\mu : \partial(\hat{G}) \to P_1^r$ with the properties claimed in the statement of Lemma 3.17. Now consider the recursive instance $\langle (G_{P_i^r,\delta} \setminus \tilde{S}, k_1, W_1, P_i^{nr}), \lambda', \ell' \rangle$ where $G_{P_i^r,\delta}$ is the graph obtained by gluing together $G[R[W_1, P_1]]$ and $\hat{G}$ via a bijection μ , $\lambda' = |X^r|, k_1 = |K^r|$ and $\ell' = |K^r|$.

To apply induction hypothesis on the above tuple, we first show that the tuple is valid. We show this by showing that $(G_{P_i^r,\delta} \setminus \tilde{S}, k_1, W_1, P_1^{nr})$ is a valid instance of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. For this we need that $W_1 \cup P_1^{nr}$ is a $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-modulator for the graph $G_{P_i^r,\delta} \setminus \tilde{S}$ which is true from Lemma 3.17, Claim 1. Hence the tuple is valid and we can apply the induction hypothesis.

Since X is ℓ-important, from Lemma 3.17, Claims 2 and 3, it follows that X^r must also be k_1-important in the graph $G_{P_i^r,\delta} \setminus \tilde{S}$. By induction hypothesis, the tuple returns a set $\mathcal{R}'$ which intersects K^r. Since $K^r \subseteq Z$, we can conclude that $\mathcal{R}'$ intersects Z as well.

Bounding the set $\mathcal{R}$: We have the value of λ dropping at every level of the search tree. Since $\lambda \leq k$, the depth of the search tree is bounded by k. The number of branches at each node is at most $k^3 \cdot k! \cdot 2^{k^{\mathcal{O}(1)}}$ (k^3 for choice of λ', j and ℓ', $k!$ for the choice of the bijection δ and $2^{k^{\mathcal{O}(1)}}$ for the size of $\mathcal{H}$). Since, at each internal node, we add at most $2k$ vertices (corresponding to $P_1 \cup P_2$), we can conclude that the size of $\mathcal{R}$ is bounded by $2^{k^{\mathcal{O}(1)}}$. ◀

We now describe the details of Main-Algorithm procedure.

▶ **Lemma 3.20.** *There is a procedure Main-Algorithm that given an instance (G, k, W_1, W_2) of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC, runs in $2^{k^{\mathcal{O}(1)}} n^{O(1)}$ and either computes a solution for the instance containing a $W_1 - W_2$ separator or concludes that no such solution exists.*

Proof. Initially, if Reduction Rule 2 is applicable (this is the case when the size of the minimum cut λ in G is zero), we use it to reduce the instance. Hence we can assume that the $W_1 - W_2$ separator is non-empty. For every $0 \leq \ell' \leq k$ and $1 \leq \lambda' \leq k$, we invoke the algorithm Branching-Set$((G, k, W_1, W_2), \lambda', \ell')$ from Lemma 3.19 to compute a set $\mathcal{R}_{\lambda',\ell'}$. We define $\mathcal{R}$ as the union of the sets $\mathcal{R}_{\lambda',\ell'}$ for all possible values of λ' and ℓ'. After this, we simply branch on every vertex v of R adding v to the solution creating a new instance $(G - v, k - 1, W_1, W_2)$ of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. If $k < 0$, we return NO. If Reduction Rule 2 applies, we use it to reduce the instance. If this results in a non-separating instance with $W = W_1 \cup W_2$, we apply the algorithm in Lemma 3.8 to solve the instance. Else we recursively run Main-Algorithm on the new instance.

We now bound the running time $T(k)$ for Main-Algorithm. The depth of the branching tree is bounded by k and the branching factor at each node is $|\mathcal{R}| \leq 2^{k^{\mathcal{O}(1)}}$. The time taken at each node is dominated by the time taken for the procedure Branching-Set. Let $Q(k)$ denote the time taken for Branching-Set. We have $T(k) = 2^{k^{\mathcal{O}(1)}} T(k-1) + Q(k)$. Let us focus on the search tree for Branching-Set. In Lemma 3.19, we proved that the depth of the tree is bounded by k and the branching factor is bounded by $2^{k^{\mathcal{O}(1)}}$. The time spent at each node is dominated by algorithm of Lemma 3.17 and that of the sub-instances of Main-Algorithm called at the node with strictly smaller values of k which is bounded by $2^{k^{\mathcal{O}(1)}} n^{O(1)} + T(k-1)$. Hence overall we have $Q(k) = 2^{k^{\mathcal{O}(1)}} Q(k-1) + 2^{k^{\mathcal{O}(1)}} n^{O(1)} + T(k-1)$. Substituting $Q(k)$ in the recurrence relation for the running time of Main-Algorithm and simplifying, we have $T(k) = \sum_{i=1}^{k} 2^{ik^{\mathcal{O}(1)}} n^{O(1)} T(k-i) \leq k \cdot 2^{k \cdot k^{\mathcal{O}(1)}} n^{O(1)} T(k-1)$ on solving we get $T(k) = 2^{k^{\mathcal{O}(1)}} n^{O(1)}$.

The correctness follows from the correctness of Lemma 3.19, of Reduction Rule 2 and Lemma 3.8. ◀

▶ **Lemma 3.21.** Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC *can be solved in $2^{k^{\mathcal{O}(1)}} n^{O(1)}$ time.*

Proof. Let (G, k, W) be the instance of Disjoint $(\Pi_1, \Pi_2, \ldots, \Pi_d)$-VDC. We first apply Lemma 3.8 to see if there is a non-separating solution for the instance. If not, we branch over all $W_1 \subset W$ and for each such choice of W_1, apply Lemma 3.20 to check if $(G, k, W_1, W_2 = W \setminus W_1)$ has a solution containing a $W_1 - W_2$ separator. The correctness and running time follows from those of Lemma 3.8 and Lemma 3.20. ◀

▶ **Theorem 3.22.** $(\Pi_1, \Pi_2, \ldots, \Pi_d)$ Vertex Deletion *can be solved in* $2^{k^{\mathcal{O}(1)}} n^{O(1)}$ *time.*

Proof. As mentioned in Section 3.1, the time taken to solve $(\Pi_1, \Pi_2, \ldots, \Pi_d)$ Vertex Deletion is $2^{k+1} \cdot 2^{k^{\mathcal{O}(1)}} n^{O(1)} = 2^{k^{\mathcal{O}(1)}} n^{O(1)}$. ◀

4 Finite Forbidden Set with Paths

We observe that several natural graph classes (like cluster graphs, edgeless graphs, P_5-free graphs) contain a path in their forbidden sets. In this section, we develop significantly faster FPT algorithms for the vertex deletion problem if each connected component of the resulting graph belongs to one of two graph classes and the forbidden set of one of them contains a path.

Deletion to Π_1 and Π_2 with Path
Input: An undirected graph G, and an integer k. Furthermore, for a fixed i, an induced path P_i is a forbidden set for Π_1.
Goal: Does G have a set S of at most k vertices such that every connected component of $G - S$ is either in Π_1 or in Π_2?

Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be the finite forbidden sets for the graph classes Π_1 and Π_2 respectively. Let d_1 be the size of a maximum sized finite forbidden set in $\mathcal{F}_1$ and d_2 be the size of a maximum sized forbidden set in $\mathcal{F}_2$. We first recall the graph operation called *gluing* that was defined in Section 3.3. We spell it out for our purpose.

Gluing Operation. Let G_1 and G_2 be graphs with H being an induced subgraph of both of them. Let $S_1 \subseteq V(G_1)$ and $S_2 \subseteq V(G_2)$ be such that $G_1[S_1] = G_2[S_2] = H$. Let f be an automorphism of H, i.e. an isomorphism from H onto H, that maps vertices of H to vertices of H. We define a collection of graphs $G \in Glue(G_1, G_2, H, S_1, S_2, f)$ with $V(G) = V(G_1) \cup V(G_2) \setminus S_2$ whose edge set $E(G)$ is as follows. We treat $V(G)$ as containing all vertices of G_1, and containing vertices of $V(G_2) \setminus S_2$. I.e. we identify vertices of S_1 and S_2 using the function f.

(1) For any pair u, v of vertices in $V(G_1)$ in $V(G)$, we add an edge if and only if $uv \in E(G_1)$, **(2)** for any pair u, v of vertices in $V(G_2) \setminus S_2$, we add an edge if and only if $uv \in E(G_2)$, **(3)** for a pair u, v of vertices where $u \in S_1$ and $v \in V(G_2) \setminus S_2$, we add the edge if and only if $f(u)v \in E(G_2)$, **(4)** for a pair u, v of vertices where $u \in V(G_1) \setminus S_1$ and $v \in V(G_2) \setminus S_2$, we have two choices. In the first choice, we add edge uv to G. In the second choice, we do not add edge uv to G.

Note that this construction provides a collection of graphs based on the last bullet point above, even for a fixed f, and for a fixed S_1 and S_2. Now towards the deletion algorithm for two classes where one of them contains a path, we first construct the following family of graphs from $\mathcal{F}_1$ and $\mathcal{F}_2$.

Construction of a new family $\mathcal{F}$ from $\mathcal{F}_1$ and $\mathcal{F}_2$. For every $F_1 \in \mathcal{F}_1, F_2 \in \mathcal{F}_2$, we construct a collection of graph $\mathcal{F}'$ by $Glue(F_1, F_2, S_1, S_2, H, f)$ for every graph H which is a subgraph of both F_1 and F_2 and for every subsets S_1 and S_2 of vertices of F_1 and F_2 respectively such that $F_1[S_1] = H = F_2[S_2]$ and for every automorphism f of H. In addition, we construct $\mathcal{F}'', \mathcal{F}'''$ as follows. The set $\mathcal{F}''$ contains all graphs formed by disjoint union of $F_1 \in \mathcal{F}_1, F_2 \in \mathcal{F}_2$ and some subset of edges between them. The set $\mathcal{F}'''$ contains all graphs formed by disjoint union of $F_1 \in \mathcal{F}_1, F_2 \in \mathcal{F}_2$ and a path P of length at most i (with $i - 1$ additional vertices) starting from a vertex in F_1 and ending at a vertex in F_2. We denote $\mathcal{F} = \mathcal{F}' \cup \mathcal{F}'' \cup \mathcal{F}'''$.

▶ **Lemma 4.1** (⋆). *Let Π_1, Π_2 be two graph classes such that a graph $G_1 \in \Pi_1$ has no induced subgraph from $\mathcal{F}_1$, and a graph $G_2 \in \Pi_2$ has no induced subgraph from $\mathcal{F}_2$. Let d_1 be the maximum number of vertices in any graph in $\mathcal{F}_1$ and d_2 be the maximum number of vertices in any graph in $\mathcal{F}_2$. Furthermore let $\mathcal{F}_1$ contains a path of length i. Construct the set $\mathcal{F}$ from $\mathcal{F}_1$ and $\mathcal{F}_2$ as described by "gluing operation" just before the lemma. Let Π be a class of graphs such that $G \in \Pi$ if and only if G has no induced subgraph present in $\mathcal{F}$. Then, every connected component of G is either in Π_1 or in Π_2 if and only if $G \in \Pi$. Furthermore, the maximum number of vertices in a set in $\mathcal{F}$ is at most $d_1 + d_2 + i$.*

▶ **Theorem 4.2** (⋆). *Let d_1 be the maximum size of an obstruction in $\mathcal{F}_1$ and d_2 be the maximum size of an obstruction in $\mathcal{F}_2$ for* Deletion to Π_1 and Π_2 with Path *problem. Then,* Deletion to Π_1 and Π_2 with Path *admits $(d_1 + d_2 + i)^k n^{\mathcal{O}(1)}$ time algorithm. It also admits $(d_1 + d_2 + i)$ factor approximation algorithm, and a polynomial sized kernel.*

Proof (Sketch). Given an instance (G, k) of Deletion to Π_1 and Π_2 with Path problem, we use "gluing operation" described above to construct a finite obstruction family $\mathcal{F}$. From Lemma 4.1, all obstructions in $\mathcal{F}$ have size at most $d_1 + d_2 + i$. We can prove that Π_1 or Π_2 Deletion is an instance of an implicit $d_1 + \hat{d} + 2 + i$-Hitting Set problem [5]. We can find induced graph H in G which is isomorphic to any forbidden set in $\mathcal{F}$ in polynomial time. By branching over the vertices of H we get an FPT algorithm running in time $(d_1+d_2+i)^k n^{\mathcal{O}(1)}$. We can get a $d_1 + d_2 + i$ factor approximation algorithm by greedily adding all vertices of H to the solution as usually done in implicit $(d_1 + d_2 + i)$-Hitting Set problems. We can use Sunflower Lemma [5, 7] to get a polynomial kernel for Π_1 or Π_2 Deletion. ◀

5 Deletion to Trees and Cliques

In this section, we consider the problem when Π_1 is the set of all cliques, and Π_2 is the set of all trees. That is, the problem is to delete k vertices so that in the resulting graph, each connected component is a tree or a clique.

Trees and Cliques Deletion Set	**Parameter:** k
Input: An undirected graph G, and an integer k	
Question: Does G have a set S of at most k vertices such that every connected component of $G - S$ is either a tree or a clique?	

We call a subset $S \subseteq V(G)$ a *trees-and-cliques deletion set* if $G \setminus S$ is such that every connected component of $G - S$ is either a tree or a clique. Note that when each component is a tree, the deletion problem is precisely the Feedback Vertex Set problem, with the resulting class of all acyclic graphs which have the set of all cycles as the (infinite) forbidden set. This has an FPT algorithm with the best runtime $O^*(3.618^k)$ [11]. See also [19] for the special deletion problem where we want the resulting graph to be a tree. When each connected component is a clique, the deletion problem is precisely the Cluster Vertex Deletion problem, with the resulting class of cluster graphs contains the graphs forbidding P_3s, paths on three vertices. For our problem, as the forbidden set for one of the graph classes is infinite, the fixed-parameter tractability does not follow from our Theorem 3.22 or Theorem 4.2.

In this section, we describe a $\mathcal{O}^*(4^k)$ time algorithm for Trees and Cliques Deletion Set problem. Recall that a *block graph* is a graph whose biconnected components are cliques. If every connected component of a graph is a tree or a clique, then clearly it is a block graph. First, we precisely characterize such block graphs whose components are trees or cliques. Recall that a paw is a graph on 4 vertices that contains a triangle and the fourth vertex is adjacent only to one vertex of the triangle. We have the following two lemmas.

▶ **Lemma 5.1.** *Let G be a block graph. Then, G is paw-free if and only if every connected component of G is either a tree or a clique.*

Proof. Suppose that every connected component of G is either a tree or a clique. As a paw has a cycle and is not a clique, G can not contain a paw. Conversely suppose G does not contain a paw, but one of its components C, is neither a tree or a clique. Then, the component must have a cycle, and a nonadjacent pair of vertices. But as G is a block graph, every biconnected component of G, and hence of C must be a clique. As C is not a clique, C has at least two biconnected components B_1 and B_2 that are cliques intersecting at a (cut) vertex. Let B_1 be the component containing a cycle. Then B_1 contains a triangle, B_2 contains an edge, and B_1 and B_2 share a vertex. This triangle and the edge form a paw, which is a contradiction. ◀

▶ **Lemma 5.2** ([2])**.** *A graph G is a block graph if and only if it has no induced cycles of length at least four and no induced D_4 (i.e., $K_4 - e$, or a diamond).*

The following corollary follows from the above lemmas.

▶ **Corollary 5.3.** *Every connected component of a graph G is either a tree or a clique if and only if it has no cycle of length at least four or a diamond or paw as induced subgraphs.*

Now, we will first focus on the following restricted version of Trees and Cliques Deletion Set problem where we assume that the graph has no C_4 (a cycle of length four), D_4 or a paw as an induced subgraph.

Restricted Trees and Cliques Deletion Set	**Parameter:** k
Input: A connected undirected graph G without C_4, D_4 and paw as induced subgraphs, and an integer k	
Question: Does G have a set S of at most k vertices such that every connected component of $G - S$ is either a tree or a clique?	

Now we give a fixed-parameter tractable algorithm for Restricted Trees and Cliques Deletion Set. We have the following lemma from [1].

▶ **Lemma 5.4** ([1])**.** *Let G be a graph that does not contain C_4 or D_4 as an induced subgraph. Then **(1)** any pair of maximal cliques of G intersects in at most one vertex, and **(2)** the number of maximal cliques in G is at most n^2.*

Let $\mathcal{C}$ denote the set of all maximal cliques of G. A vertex $v \in V(G)$ is called *external* if it is part of at least two maximal cliques of $\mathcal{C}$. We construct an auxiliary bipartite graph $\hat{G}$ from G with $V(\hat{G}) = V(G) \uplus V_{\mathcal{C}}$ where $V_{\mathcal{C}}$ has a vertex v_c for every $c \in \mathcal{C}$. In the graph $\hat{G}$, we add an edge from a vertex $v \in V(G)$ to a vertex $v_c \in V_{\mathcal{C}}$ if and only if v is one of the external vertices of the clique $c \in \mathcal{C}$. We prove the following lemma which is similar to Lemma 7 in [1].

▶ **Lemma 5.5** ($\star$)**.** *Let G be a graph without C_4, D_4 and paw as induced subgraphs and $S \subseteq V(G)$. Then S is a trees-and-cliques deletion set of G if and only if $\hat{G} \setminus S$ is acyclic.*

Now we consider the weighted feedback vertex set problem

Weighted Feedback Vertex Set	**Parameter:** k
Input: A weighted undirected graph $G = (V, E)$ with $w : V(G) \to \mathbb{N}$, and an integer k.	
Question: Find a set $S \subseteq V(G)$ of minimum weight that contains at most k vertices.	

that has an $O^*(3.618^k)$ algorithm [1].

Hence to solve the Restricted Trees and Cliques Deletion Set, we can simply form the auxiliary graph as described in Lemma 5.5. Then we define a weight function on the vertices of $\hat{G}$ as those vertices of G taking weight 1, and those vertices of the maximal cliques taking value $k+1$. Then we apply the algorithm for weighted feedback vertex set to check whether the minimum weight of, a feedback vertex set of (the weighted graph) $\hat{G}$ containing at most k vertices, is at most k. Thus we have

▶ **Lemma 5.6.** Restricted Trees and Cliques Deletion Set *can be solved in* $\mathcal{O}^*(3.618^k)$ *time.*

Now by branching on vertices of C_4s, D_4s and paws and then applying the algorithm of Lemma 5.6, we obtain the following theorem.

▶ **Theorem 5.7.** *Given a graph G on n vertices, we can determine in $\mathcal{O}^*(4^k)$ time whether G has at most k vertices whose deletion results in a graph where every connected component is a tree or a clique.*

Proof. Let $S \subseteq V(G)$ be a set of vertices such that every connected component of $G - S$ is either a tree or a clique. Observe that $G - S$ cannot contain a C_4, diamond or paw as induced subgraph. Hence, S must intersect all induced C_4, all induced diamonds and all induced paws in G. As long as we find a set of four vertices A such that $G[A]$ induces a C_4, or a paw, or a diamond, we branch on every vertex $v \in A$, and solve recursively the instance $(G - \{v\}, k-1)$. If one of these branches returns a solution X, we return $X \cup \{v\}$ as a solution of G. Otherwise, we return that (G, k) is a no-instance. After G has no four vertices that induces a diamond, or a C_4, or a paw, then, we do not make any further recursive call. We invoke Lemma 5.6 to apply the algorithm for Restricted Trees and Cliques Deletion Set and return the output of the algorithm. Since, the algorithm for Restricted Trees and Cliques Deletion Set takes $\mathcal{O}^*(3.618^k)$ time and we branch on at most k C_4, diamonds and paws, our algorithm takes $\mathcal{O}^*(4^k)$ time. This completes the proof. ◀

6 Conclusion

We have initiated a study on vertex deletion problems to scattered graph classes and showed that when there are a finite number of graph classes each characterized by a finite forbidden set, the problem is fixed-parameter tractable. The existence of a polynomial kernel for this case is a natural open problem. Other open problems include obtaining improved algorithms at least for special cases of finite classes and investigating other scattered graph classes when some of them have infinite forbidden sets.

References

1. Akanksha Agrawal, Sudeshna Kolay, Daniel Lokshtanov, and Saket Saurabh. A faster FPT algorithm and a smaller kernel for block graph vertex deletion. In *LATIN 2016: Theoretical Informatics*, pages 1–13. Springer, 2016.
2. Andreas Brandstadt, Jeremy P Spinrad, et al. *Graph classes: a survey*, volume 3. Siam, 1999.
3. Leizhen Cai. Fixed-Parameter Tractability of Graph Modification Problems for Hereditary Properties. *Inf. Process. Lett.*, 58(4):171–176, 1996.
4. Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009.
5. M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

6 Marek Cygan, Fedor V Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 3. Springer, 2015.

7 Walter A. Deuber, Paul Erdös, David S. Gunderson, Alexandr V. Kostochka, and A. G. Meyer. Intersection Statements for Systems of Sets. *J. Comb. Theory, Ser. A*, 79(1):118–132, 1997.

8 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

9 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. *ACM Trans. Algorithms*, 13(2):29:1–29:32, 2017.

10 Pinar Heggernes, Pim van 't Hof, Bart M. P. Jansen, Stefan Kratsch, and Yngve Villanger. Parameterized complexity of vertex deletion into perfect graph classes. *Theor. Comput. Sci.*, 511:172–180, 2013.

11 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic Feedback Vertex Set. *Inf. Process. Lett.*, 114(10):556–560, 2014.

12 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980.

13 Daniel Lokshtanov. Wheel-free deletion is W[2]-hard. In *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, pages 141–147, 2008.

14 Daniel Lokshtanov and MS Ramanujan. Parameterized tractability of multiway cut with parity constraints. In *International Colloquium on Automata, Languages, and Programming*, pages 750–761. Springer, 2012.

15 Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.

16 Geevarghese Philip, Ashutosh Rai, and Saket Saurabh. Generalized pseudoforest deletion: Algorithms and uniform kernel. *SIAM J. Discrete Math.*, 32(2):882–901, 2018.

17 Ashutosh Rai and M. S. Ramanujan. Strong parameterized deletion: Bipartite graphs. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016*, pages 21:1–21:14, 2016.

18 Ashutosh Rai and Saket Saurabh. Bivariate complexity analysis of almost forest deletion. *Theor. Comput. Sci.*, 708:18–33, 2018.

19 Venkatesh Raman, Saket Saurabh, and Ondrej Suchý. An FPT algorithm for Tree Deletion Set. *J. Graph Algorithms Appl.*, 17(6):615–628, 2013.

20 Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM J. Comput.*, 10(2):310–327, 1981.

Structural Parameterizations with Modulator Oblivion

Ashwin Jacob
The Institute of Mathematical Sciences, HBNI, Chennai, India
ajacob@imsc.res.in

Fahad Panolan
Department of Computer Science and Engineering, IIT Hyderabad, India
fahad@cse.iith.ac.in

Venkatesh Raman
The Institute of Mathematical Sciences, HBNI, Chennai, India
vraman@imsc.res.in

Vibha Sahlot
The Institute of Mathematical Sciences, HBNI, Chennai, India
sahlotvibha@gmail.com

Abstract

It is known that problems like Vertex Cover, Feedback Vertex Set and Odd Cycle Transversal are polynomial time solvable in the class of chordal graphs. We consider these problems in a graph that has at most k vertices whose deletion results in a chordal graph, when parameterized by k. While this investigation fits naturally into the recent trend of what are called "structural parameterizations", here we assume that the deletion set is not given.

One method to solve them is to compute a k-sized or an approximate ($f(k)$ sized, for a function f) chordal vertex deletion set and then use the structural properties of the graph to design an algorithm. This method leads to at least $k^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ running time when we use the known parameterized or approximation algorithms for finding a k-sized chordal deletion set on an n vertex graph.

In this work, we design $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time algorithms for these problems. Our algorithms do not compute a chordal vertex deletion set (or even an approximate solution). Instead, we construct a tree decomposition of the given graph in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ where each bag is a union of four cliques and $\mathcal{O}(k)$ vertices. We then apply standard dynamic programming algorithms over this special tree decomposition. This special tree decomposition can be of independent interest.

Our algorithms are, what are sometimes called *permissive* in the sense that given an integer k, they detect whether the graph has no chordal vertex deletion set of size at most k or output the special tree decomposition and solve the problem.

We also show lower bounds for the problems we deal with under the Strong Exponential Time Hypothesis (SETH).

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized Complexity, Chordal Graph, Tree Decomposition, Strong Exponential Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.19

Related Version Full version available at `https://arxiv.org/abs/2002.09972`.

1 Introduction and Motivation

Main motivation for parameterized complexity and algorithms is that hard problems have a number of parameters in their input, and feasible algorithms can be obtained when some of these parameters tend to be small. However, barring width parameters (like treewidth and cliquewidth), early parameterizations of problems were mostly in terms of solution size.

However starting from the work of Fellows et al. [16] and Jansen et al. [26, 17], there has been a lot of study on parameterizations by some structure of the input. The motivations for these parameterizations are that many problems are computationally easy on special classes of graphs like edge-less graphs, forests and interval graphs. Thus parameterizing by the size of a modulator (set of vertices in the graph whose removal results a graph in easy graph class) became a natural choice of investigation. Examples of such parameterizations include Clique and Feedback Vertex Set parameterized by the size of minimum vertex cover (i.e., modulator to edge-less graphs), Vertex Cover parameterized by the size of minimum feedback vertex set (i.e., modulator to forests) [26, 27]. See also [33, 34] for more such parameterizations.

We continue this line of work on problems in input graphs that are not far from a chordal graph. By distance to a chordal graph, we mean the minimum number of vertices in the graph whose deletion results in a chordal graph. We call this set as a chordal vertex deletion set (CVD). Specifically, we look at Vertex Cover, Feedback Vertex Set and Odd Cycle Transversal and some generalizations of these problems, parameterized by the size of a CVD, as these problems are polynomial time solvable in chordal graphs [22, 38, 10].

In problems for which the parameter is the size of a modulator, it is also assumed that the modulator is given with the input. This assumption can be removed if finding the modulator is also fixed-parameter tractable (FPT) parameterized by the modulator size. However, there are instances where finding the modulator is more expensive than solving the problem if the modulator is given. For example, finding a subset of k vertices whose deletion results in a perfect graph is known to be W-hard [24], whereas if the deletion set is given, then one can show (as explained a bit later in this section) that Vertex Cover (thus Independent Set) is FPT when parameterized by the size of the deletion set.

Hence Fellows et al. [17] ask whether Independent Set (or equivalently, Vertex Cover) is FPT when parameterized by a (promised) bound on the vertex-deletion distance to a perfect graph, without giving a minimum deletion set in the input. While we do not answer this question, we address a similar question in the context of problems parameterized by the distance to chordal graphs, another well-studied class of graphs where Vertex Cover is polynomial time solvable whereas the best-known algorithm to find a k-sized chordal deletion set takes $\mathcal{O}^*(k^{\mathcal{O}(k)})$ [1] time [8]. We also do not know of a constant factor (FPT) approximation algorithm for CVD even with $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ running time. There are many recent results on polynomial time approximation algorithms for Chordal Vertex Deletion [28, 1, 30] with the current best algorithm having a $\mathcal{O}(\mathsf{opt}\log\mathsf{opt})$ ratio, where opt is the size of minimum CVD [30]. If we use this approximation algorithm and do branching (see Section 1.1 below), then we can obtain a $2^{\mathcal{O}(k^2\log k)}n^{\mathcal{O}(1)}$ time algorithm for Vertex Cover.

Hence, in a similar vein to the question by Fellows et al., we ask whether (minimum) Vertex Cover (and other related problems) can be solved in $\mathcal{O}^*(2^{\mathcal{O}(k)})$ time with only a promise on the size k of the chordal deletion set, and answer the question affirmatively. Our algorithms even go one step further, in not even needing the promise. They solve the problem or determine that the chordal deletion set is of size more than k.

[1] $\mathcal{O}^*$ notation hides polynomial factor in the input length

Our Results. Specifically we give $\mathcal{O}^*(2^{\mathcal{O}(k)})$ algorithms for the problems defined below.

> Weighted d-Colorable Subgraph by CVD
> **Input:** A graph $G = (V, E)$, a weight function $w : V(G) \to \mathbb{R}$ and $k, \ell, d \in \mathbb{N}$.
> **Parameter:** k
> **Question:** Determine if there is a vertex set X of weight at most ℓ in G such that $G - X$ is d-colorable or output that minimum chordal vertex deletion set of G is of size more than k?

When $d = 1$ and $d = 2$, the problem reduces to Weighted Vertex Cover by CVD (WVC By CVD) and Weighted Odd Cycle Transversal by CVD (WOCT by CVD) where we require the graph $G - X$ to be an independent set and bipartite, respectively. We also define Weighted Feedback Vertex Set by CVD (WFVS by CVD) where we require the graph $G - X$ to be a forest.

If all the weights are 1, we call them the unweighted version of these problems. We remark that our algorithms do not necessarily address the question of whether the input graph has a CVD of size at most k, and may actually solve the problem sometimes even when the CVD size is more than k.

We also show that the unweighted versions of all the problems mentioned above cannot be solved in $\mathcal{O}^*((2 - \epsilon)^k)$ time under Strong Exponential Time Hypothesis (SETH) even if a CVD of size k is given as part of the input. This matches the upper bound of the known algorithm for WVC By CVD when the modulator is given.

1.1 Related Work

When CVD is given. If we are given a CVD S of size k along with an n-vertex graph G as the input, then one can easily get a $2^k n^{\mathcal{O}(1)}$ time algorithm (call it $\mathcal{A}$) for Vertex Cover as follows. First, we guess the subset X of S that is part of our solution. Let Y be the subset of vertices in $V(G) \setminus S$ such that for each $y \in Y$ there is an edge between y and a vertex in $S \setminus X$. Clearly, $X \cup Y$ is part of the Vertex Cover solution and it will cover all the edges incident on S. Then we are left with finding an optimum weighted vertex cover in $G - (S \cup Y)$ which is a chordal graph. This can be done in polynomial time. As we have 2^k choices for X, the total running time of the algorithm is $2^k n^{\mathcal{O}(1)}$. An FPT algorithm with $O^*(k^{O(k)})$ time for WFVS by CVD is given by Jansen et al [29] where they first find the modulator. This algorithm follows the algorithm to find a minimum FVS in bounded treewidth graphs and a similar trick works for Odd Cycle Transversal too, when the modulator is given.

When the modulator is given, the FPT algorithms discussed above have been generalized for other problems and other classes of graphs (besides those that are k away from the class of chordal graphs). Let Φ be a Counting Monadic Second Order Logic (CMSO) formula and $t \geq 0$ be an integer. For a given graph $G = (V, E)$, the task is to maximize $|X|$ subject to the following constraints: there is a set $F \subseteq V$ such that $X \subseteq F$, the subgraph $G[F]$ induced by F is of treewidth at most t, and structure $(G[F], X)$ models Φ. Note that the problem corresponds to finding a minimum vertex cover and a minimum feedback vertex set when $t = 0$ and $t = 1$ respectively when Φ is a tautology. For a polynomial $poly$, let G_{poly} be the class of graphs such that, for any $G \in G_{poly}$, graph G has at most $poly(n)$ minimal separators. Fomin et al [20] gave a polynomial time algorithm for solving this optimization problem on the graph class G_{poly}. Consider $G_{poly} + kv$ to be the graph class formed from G_{poly} where to each graph we add at most k vertices of arbitrary adjacencies. Liedloff et al. [31] further proved that, the above problem is FPT on $G_{poly} + kv$, with parameter k,

where the modulator is also a part of input. As a chordal graph has polynomially many minimal separators [22], we obtain that WVC By CVD and WFVS by CVD are FPT when the modulator is given.

Other "permissive" problems. Similar problems have been termed as "permissive problems" in the context of testing satisfiability of CSPs (constraint satisfaction problems) with small sized strong backdoors [21]. While detecting strong backdoors to a general CSP is hard, the authors address the question of satisfiability of CSPs where the backdoor set is not given, and the algorithm was supposed to solve satisfiability or determine that the backdoor set size is more than k.

An example line of work where faster constant factor approximation algorithm is available, is in the context of optimization problems parameterized by treewidth. For example, the Independent Set problem parameterized by treewidth of the graph tw can be solved using standard dynamic programming (DP) in $2^{tw} \cdot tw^{O(1)} \cdot n$ time[12]. But the best known algorithm for outputting a tree-decomposition of minimum width takes time $\mathsf{tw}^{\mathcal{O}(\mathsf{tw}^3)}n$ [2]. Thus, the total running time is $\mathsf{tw}^{\mathcal{O}(\mathsf{tw}^3)}n$, when a tree decomposition is not given as an input. But one can overcome this by obtaining a tree decomposition of width $5\mathsf{tw}$ in time $2^{\mathcal{O}(\mathsf{tw})}n$ [4] and then applying the DP algorithm over the tree decomposition.

One previous example we know of a parameterized problem where the FPT algorithm solves the problem without the modulator or even the promise, is Vertex Cover parameterized by the size of König Vertex Deletion set k. A König vertex deletion set of G is a subset of vertices of G whose removal results in a graph where the size of its minimum vertex cover and maximum matching are the same. In Vertex Cover by König Vertex Deletion, we are given graph $G = (V, E)$, $k, \ell \in \mathbb{N}$ and an assumption that there exists a König vertex deletion set of size k in G, here k is parameter. We want to ask whether there exist a vertex cover of size ℓ in G? Lokshtanov et al. [32] solve Vertex Cover by König Vertex Deletion in $\mathcal{O}^*(1.5214^k)$ time without the promise.

Finally we remark that there is an analogous line of work in the classical world of polynomial time algorithms. For example, it is known that finding a maximum clique in a unit disk graph is polynomial time solvable given a unit disk representation of the unit disk graph [9], though it is NP-hard to recognize whether a given graph is a unit disk graph [7]. Raghavan and Spinrad [36] give a permissive algorithm that given a graph either finds a maximum clique in the graph or outputs a certificate that the given graph is not a unit disk graph. See also [6, 23, 20] for some other examples of permissive algorithms.

1.2 Our Techniques

The first step in our algorithms is to obtain, what we call a semi clique tree decomposition of the given graph if one exists. It is known [22] that every chordal graph has a clique-tree decomposition, i.e., a tree decomposition where every bag is a clique in the graph. If the modulator is given, then we can add it to each bag, and obtain a tree-decomposition where each bag is a clique plus at most k vertices. In our case (where the modulator is not given), we obtain a tree decomposition in $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time where each bag can be partitioned into $C \uplus N$, where C can be covered by at most 4 cliques in G and $|N| \leq 7k + 5$. Here we also know a partition $C_1 \uplus C_2 \uplus C_3 \uplus C_4$ of C where each C_i is a clique. We call this tree decomposition a $(4, 7k + 5)$-semi clique tree decomposition. Our result in this regard is formalized in the following theorem.

▶ **Theorem 1.** *There is an algorithm that given a graph G and an integer k runs in time $\mathcal{O}(2^{7k} \cdot (kn^4 + n^{\omega+2}))$ where ω is the matrix multiplication exponent and either constructs a $(4, 7k+5)$-semi clique tree decomposition $\mathcal{T}$ of G or concludes that there is no chordal vertex deletion set of size k in G. Moreover, the algorithm also provides a partition $C_1 \uplus C_2 \uplus C_3 \uplus C_4 \uplus N$ of each bag of $\mathcal{T}$ such that $|N| \leq 7k+5$ and C_i is a clique in G for all $i \in \{1,2,3,4\}$.*

After getting a $(4, 7k+5)$-semi clique tree decomposition, we then design DP algorithms for Vertex Cover, Feedback Vertex Set and Odd Cycle Transversal on this tree decomposition. Since the vertex cover of a clique has to contain all but one vertex of the clique, the number of ways the solution might intersect a bag of the tree is at most $\mathcal{O}(2^{7k}n^4)$. Using this fact, one can bound the running time for the DP algorithm to $\mathcal{O}(2^{7k}n^5)$. The overall running time would be the sum of the time taken to construct a $(4, 7k+5)$-semi clique tree decomposition and the time of the DP algorithm on this tree decomposition which is bounded by $\mathcal{O}(2^{7k}n^5)$. In the case of Feedback Vertex Set and Odd Cycle Transversal, again from each clique all but two vertices will be in the solution. Using this fact one can bound the running time of WFVS By CVD and WOCT by CVD to be $\mathcal{O}^*(2^{O(k)})$.

Very recently, Fomin and Golovach [18] give subexponential algorithms to various problems on graphs which can be turned into a chordal graph by adding k edges. Similar to the line of work in this paper, they come up with an almost-clique tree decomposition (where each bag can be converted to a clique by adding k edges) and then apply dynamic programming algorithms on this tree decompositions. We use the dynamic programming algorithms in this paper on the tree decomposition we constructed to give algorithms for Weighted d-Colorable Subgraph parameterized by minimum CVD size.

Organization of the paper. In Section 2, we state the notations used in this paper and give the necessary preliminaries on tree decomposition and parameterized complexity. In Section 3, we prove Theorem 1. In Section 4, we first address Weighted d-Colorable Subgraph by CVD using dynamic programming on semi clique tree decomposition. We then give more direct and faster algorithms for WVC By CVD and WOCT by CVD and also for WFVS by CVD. We then conclude this section with lower bounds on these problems assuming SETH.

2 Preliminaries

For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$. We use $A \uplus B$ to denote the set formed from the union of disjoint sets A and B. For a function $w : X \to \mathbb{R}$, we use $w(D) = \sum_{x \in D} w(x)$.

For $V' \subseteq V$, $G[V']$ and $G - V'$ denote the graph induced on V' and $V \setminus V'$, respectively. For a vertex $v \in V$, $G - v$ denotes the graph $G - \{v\}$. For a vertex $v \in V$, $N_G(v)$ and $N_G[v]$ denote the open neighborhood and closed neighborhood of v, respectively. That is, $N_G(v) = \{u \colon \{v, u\} \in E\}$ and $N_G[v] = N_G(v) \cup \{v\}$. Also we define for a subset $X \subseteq V(G)$, $N_G(X) = \bigcup_{v \in X}(N_G(v) \setminus X)$ and $N_G[X] = N_G(X) \cup X$. We omit the subscript G, when the graph is clear from the context.

We use the term graph for a simple undirected graph without loops and parallel edges. For a graph G, we use $V(G)$ and $E(G)$ to denote its vertex set and edge set, respectively. A graph is *chordal* if it does not contain a cycle of length greater than or equal to 4 as an induced subgraph. A subset $S \subseteq V(G)$ such that $G - S$ is a chordal graph is called a *chordal vertex deletion set*. We say that a graph G is a union of ℓ cliques if $V(G) = V_1 \uplus \ldots \uplus V_\ell$ and V_i is a clique in G for all $i \in \{1, \ldots, \ell\}$. We use standard notation and terminology from the book [14] for graph-related terms which are not explicitly defined here.

A graph G is k-colorable if its vertices can be colored in such a way that the endpoints of every edge of G has two different colors.

▶ **Definition 2** (Separator and Separation). *Given a graph G and vertex subsets $A, B \subseteq V(G)$, a subset $C \subseteq V(G)$ is called a separator of A and B if every path from a vertex in A to a vertex in B (we call it $A-B$ path) contains a vertex from C. A pair of vertex subsets (A, B) is a separation in G if $A \cup B = V(G)$ and $A \cap B$ is a separator of $A \setminus B$ and $B \setminus A$.*

▶ **Definition 3** (Balanced Separator and Balanced Separation). *For a graph G, a weight function $w : V(G) \to \mathbb{R}_{\geq 0}$ and $0 < \alpha < 1$, a set $S \subseteq V(G)$ is called an α-balanced separator of G with respect to w if for any connected component C of $G - S$, $w(V(C)) \leq \alpha \cdot w(V(G))$. A pair of vertex subsets (A, B) is an α-balanced separation in G with respect to w if (A, B) is a separation in G and $w(A \setminus B) \leq \alpha \cdot w(V(G))$ and $w(B \setminus A) \leq \alpha \cdot w(V(G))$.*

▶ **Definition 4** (Tree decomposition). *A tree decomposition of a graph G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, where T is a tree and for any $t \in V(T)$, a vertex subset $X_t \subseteq V(G)$ is associated with it, called a* bag*, such that the following conditions holds.*

- $\bigcup_{t \in V(T)} X_t = V(G)$.
- *For any edge $\{u, v\} \in E(G)$, there is a node $t \in V(T)$ such that $u, v \in X_t$.*
- *For any vertex $u \in V(G)$, the set $\{t \in V(T) \colon u \in X_t\}$ of nodes induces a connected subtree of T.*

The width of the tree decomposition $\mathcal{T}$ is $\max_{t \in V(T)} |X_t| - 1$ and the treewidth of G is the minimum width over all tree decompositions of G.

▶ **Proposition 5** ([15]). *Let G be a graph and C be a clique in G. Let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be a tree decomposition of G. Then, there is a node $t \in V(T)$ such that $C \subseteq X_t$.*

▶ **Definition 6** (Clique tree decomposition). *A clique tree decomposition of a graph G is a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ where X_t is a clique in G for all $t \in V(T)$.*

▶ **Proposition 7** ([22]). *A graph is chordal if and only if it has a clique tree decomposition.*

▶ **Definition 8.** *A graph G is called an (c, ℓ)-semi clique if there is a partition $C \uplus N$ of $V(G)$ such that $G[C]$ is a union of at most c cliques and $|N| \leq \ell$.*

▶ **Definition 9** ((c, ℓ)-semi clique tree decomposition). *For a graph G and $c, \ell \in \mathbb{N}$, a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G is a (c, ℓ)-semi clique tree decomposition if $G[X_t]$ is a (c, ℓ)-semi clique for each $t \in V(T)$.*

We define the Node Multiway Cut problem where we are given an input graph $G = (V, E)$, a set $T \subseteq V$ of terminals and an integer k. We want to ask whether there exists a set $X \subseteq V \setminus T$ of size at most k such that any path between two different terminals intersects X.

We use the following lemma in Section 3.

▶ **Proposition 10** ([19]). *Let T be a tree and $x, y, z \in V(T)$. Then there exists a vertex $v \in V(T)$ such that every connected component of $T - v$ has at most one vertex from $\{x, y, z\}$.*

SETH. For $q \geq 3$, let δ_q be the infimum of the set of constants c for which there exists an algorithm solving q-SAT with n variables and m clauses in time $2^{cn} \cdot m^{\mathcal{O}(1)}$. The *Strong Exponential-Time Hypothesis (SETH)* conjectures that $\lim_{q \to \infty} \delta_q = 1$. SETH implies that CNF-SAT on n variables cannot be solved in $\mathcal{O}^*((2 - \epsilon)^n)$ time for any $\epsilon > 0$.

For definitions and notions on parameterized complexity, we refer to [12]. Throughout the paper, ω denotes the matrix multiplication exponent.

3 Semi Clique Tree Decomposition

Given a graph G and an integer k, our aim is to construct a $(4, 7k + 5)$-semi clique tree decomposition $\mathcal{T}$ of G or conclude that G has no CVD of size at most k. We loosely follow the ideas used for the tree decomposition algorithm in [37] to construct a tree decomposition of a graph G of width at most $4\mathsf{tw}(G) + 4$, where $\mathsf{tw}(G)$ is the tree width of G. But before that we propose the following lemmas that we use in getting the required $(4, 7k + 5)$-semi clique tree decomposition.

▶ **Lemma 11.** *Let G be a graph having a CVD of size k. Then G has a $(1, k)$-semi clique tree decomposition.*

▶ **Lemma 12.** *For a graph G on n vertices with a CVD of size k, the number of maximal cliques in G are bounded by $\mathcal{O}(2^k \cdot n)$. Furthermore, there is an algorithm that given any graph G either concludes that there is no CVD of size k in G or enumerates all the maximal cliques of G in $\mathcal{O}(2^k \cdot n^{\omega+1})$ time where ω is the matrix multiplication exponent.*

Proof. Let $X \subseteq V(G)$ be of size at most k such that $G - X$ is a chordal graph. For any maximal clique C in G let $C_X = C \cap X$ and $C_{G-X} = C \setminus X$. Since $G - X$ is a chordal graph, it has at most $n - k$ maximal cliques [22].

We claim that for a subset $C_X \subseteq X$ and a maximal clique Q in $G - X$, there is at most one subset $Q' \subseteq Q$ such that $C_X \cup Q'$ forms a maximal clique in G. If there are two distinct subsets Q_1, Q_2 of Q such that $C_X \cup Q_1$ and $C_X \cup Q_2$ are cliques in G, then $C_X \cup Q_1 \cup Q_2$ is a clique larger than the cliques $C_X \cup Q_1$ and $C_X \cup Q_2$. Thus, since there are at most 2^k subsets of X and at most n maximal cliques in G, the total number of maximal cliques in G is upper bounded by $2^k(n - k)$.

There is an algorithm that given a graph H, enumerates all the maximal cliques of H with $\mathcal{O}(|V(H)|^\omega)$ delay (the maximum time taken between outputting two consecutive solutions) [35]. If G has a CVD of size k, there are at most $2^k n$ maximal cliques in G which can be enumerated in $\mathcal{O}(2^k n^{\omega+1})$ time. So the algorithm to enumerate clique runs for at most $2^k n + 1$ rounds, if we note that the number of maximal cliques enumerated is more than $2^k n$ then we return that G has no CVD of size k. ◀

▶ **Lemma 13.** *Let G be a graph having a CVD of size k and $w : V(G) \to \mathbb{R}_{\geq 0}$ be a weight function on $V(G)$. There exists a $\frac{2}{3}$-balanced separation (A, B) of G with respect to w such that the graph induced on the corresponding separator $G[A \cap B]$ is a $(1, k)$-semi clique.*

Proof. First we prove that there is a $\frac{1}{2}$-balanced separator X such that $G[X]$ is a $(1, k)$-semi clique. By Lemma 11, there is a $(1, k)$-semi clique tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G. Arbitrarily root the tree of T at a node $r \in V(T)$. For any node $y \in V(T)$, let T_y denote the subtree of T rooted at node y and G_y denote the graph induced on the vertices of G present in the bags of nodes of T_y. That is $V(G_y) = \bigcup_{t \in V(T_y)} X_t$. Let t be the farthest node of T from the root r such that $w(V(G_t)) > \frac{1}{2}w(V(G))$. That is, for all nodes $t' \in V(T_t) \setminus \{t\}$, we have that $w(V(G_{t'})) \leq \frac{1}{2}w(V(G))$.

We claim that $X = X_t$ is a $\frac{1}{2}$-balanced separator of G. Let $t_1, \ldots, t_p$ be the children of t. Since X is a bag of the tree decomposition $\mathcal{T}$, each of the connected components of $G - X$ are contained either in $G_{t_i} - X$ for some $i \in [p]$ or $G[V(G) \setminus V(G_t)]$. Since $w(V(G_t)) > \frac{1}{2}w(V(G))$, we have $w(V(G) \setminus V(G_t)) < \frac{1}{2}w(V(G))$. By the choice of t, we have $w(V(G_{t_i})) \leq \frac{1}{2}w(V(G))$ for all $i \in [p]$.

Now we define a $\frac{2}{3}$-balanced separation (A, B) for G such that the set $X = A \cap B$ ($\frac{1}{2}$ balanced separator). Let $D_1, \ldots, D_q$ be the vertex sets of the connected components of $G - X$. Let $a_i = w(D_i)$ for all $i \in [q]$. Without loss of generality, assume that $a_1 \geq \ldots \geq a_q$.

Let q' be the smallest index such that $\sum_{i=1}^{q'} a_i \geq \frac{1}{3}w(V(G))$ or $q' = q$ if no such index exists. Clearly, $\sum_{i=q'+1}^{q} a_i \leq \frac{2}{3}w(V(G))$. We prove that $\sum_{i=1}^{q'} a_i \leq \frac{2}{3}w(V(G))$. If $q' = 1$, $\sum_{i=1}^{q'} a_i = a_{q'} \leq \frac{1}{2}w(V(G))$ and we are done. Else, since q' is the smallest index such that $\sum_{i=1}^{q'} a_i \geq \frac{1}{3}w(V(G))$, we have $\sum_{i=1}^{q'-1} a_i < \frac{1}{3}w(V(G))$. We also note that $a_{q'} \leq a_{q'-1} \leq \sum_{i=1}^{q'-1} a_i < \frac{1}{3}w(V(G))$. Hence $\sum_{i=1}^{q'} a_i = \sum_{i=1}^{q'-1} a_i + a_{q'} \leq \frac{2}{3}w(V(G))$.

Now we define $A = X \cup \bigcup_{i\in[q']} D_i$ and $B = X \cup \bigcup_{i\in[q]\setminus[q']} D_i$. Notice that $X = A \cap B$ and (A, B) is a separation of G. Also notice that $w(A \setminus B) = \sum_{i=1}^{q'} a_i \leq \frac{2}{3}w(V)$ and $w(B \setminus A) = \sum_{i=q'+1}^{q} a_i \leq w(V(G)) - \frac{1}{3}w(V(G)) = \frac{2}{3}w(V(G))$ as $\sum_{i=1}^{q'} a_i \geq \frac{1}{3}w(V(G))$. Since X is a bag of the tree decomposition $\mathcal{T}$, $G[X]$ is a $(1, k)$-semi clique. ◀

Using Lemmas 12 and 13, we obtain the following corollary.

▶ **Corollary 14.** *Let G be a graph with a CVD of size k. Let $N \subseteq V(G)$ with $5k + 3 \leq |N| \leq 6k + 4$. Then there exists a partition (N_A, N_B) of N and a vertex subset $X \subseteq V(G)$ satisfying the following properties.*

- *$|N_A|, |N_B| \leq 4k + 2$.*
- *X is a vertex separator of N_A and N_B in the graph G.*
- *$G[X]$ is a $(1, k)$-semi clique.*

Moreover, there is an algorithm that given any graph G, either concludes that there is no CVD of size k in G or computes such a partition (N_A, N_B) of N and the set X in $\mathcal{O}(2^{7k} \cdot (kn^3 + n^{\omega+1}))$ time.

Proof. Let us define a weight function $w : V(G) \to \mathbb{R}_{\geq 0}$ such that $w(v) = 1$ if $v \in N$ and 0 otherwise. From Lemma 13, we know that there exists a pair of vertex subsets (A, B) which is the balanced separation of G with respect to w where the graph induced on the corresponding separator $G[A \cap B]$ is a $(1, k)$-semi clique.

Let us define the partition (N_A, N_B). We add $(A \setminus B) \cap N$ to N_A and $(B \setminus A) \cap N$ to N_B. Since (A, B) is a balanced separation of G with respect to w, $|(A \setminus B) \cap N|, |(B \setminus A) \cap N| \leq \frac{2}{3}|N| \leq 4k+2$. For each vertex $u \in (A \cap B) \cap N$, we iteratively add u to the currently smaller of the two sets of N_A and N_B. Since $|N| \leq 6k + 4 \leq 2 \cdot (4k + 2)$, we have $|N_A|, |N_B| \leq 4k + 2$ even after this process. This shows the existence of subsets N_A, N_B and $X = A \cap B$. But the proof is not constructive as the existence of (A, B) uses the $(1, k)$-semi clique tree decomposition of G which requires the chordal vertex deletion set.

We now explain how to compute these subsets without the knowledge of a $(1, k)$-semi clique tree decomposition of G. Let $X = C'' \uplus N''$ where C'' is a clique and $|N''| \leq k$. We use Lemma 12 to either conclude that G has no CVD of size k or go over all maximal cliques of G to find a maximal clique D such that $C'' \subseteq D$. We can conclude that in the remaining graph $G[V \setminus D]$, there exists a separator $Z \subseteq N'' = X \setminus C''$ of size at most k for the sets N_A and N_B.

We go over all $2^{|N|} \leq 2^{6k+4}$ 2-partitions of N to guess the partition (N_A, N_B). Then we apply the classic Ford-Fulkerson maximum flow algorithm to find the separator Z of the sets N_A and N_B in the graph $G[V \setminus D]$. If $|Z| > k$, we can conclude that G has no CVD of size k in G. Thus, we obtained a set $X' = D \uplus Z$ such that $G[X']$ is a $(1, k)$-semi clique and X' is a vertex separator of N_A and N_B in the graph G.

Now we estimate the time taken to obtain these sets. We first go over all $\mathcal{O}(2^k \cdot n)$ maximal cliques of the graph which takes $\mathcal{O}(2^k \cdot n^{\omega+1})$ time. Then for each of the $\mathcal{O}(2^k \cdot n)$ maximal cliques, we go over at most 2^{6k+4} guesses for N_A and N_B. Finally we use the Ford-Fulkerson maximum flow algorithm to find the separator of size at most k for N_A and N_B which takes $\mathcal{O}(k(n + m))$ time. Overall the running time is $\mathcal{O}(2^k \cdot n^{\omega+1} + (2^k n) \cdot 2^{6k} \cdot (k(n + m))) = \mathcal{O}(2^{7k} \cdot (kn^3 + n^{\omega+1}))$. ◀

▶ **Lemma 15.** *Let G be a graph having a CVD of size k. Let C_1, C_2, C_3 be three distinct cliques in G. Then there exists a vertex subset $X \subseteq V(G)$ such that $G[X]$ is a $(1,k)$-semi clique and X is a separator of C_i and C_j for all $i,j \in \{1,2,3\}$ and $i \neq j$. Moreover, there is an algorithm that given any graph G, either concludes that there is no CVD of size k in G or computes X in $\mathcal{O}(4^k \cdot (kn^3 + n^{\omega+1}))$ time.*

Proof. By Lemma 11, there is a $(1,k)$-semi clique tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G. By Proposition 5, we know that there exist nodes $t_1, t_2, t_3 \in V(T)$ such that $C_1 \subseteq X_{t_1}$, $C_2 \subseteq X_{t_2}$ and $C_3 \subseteq X_{t_3}$. If two of the three nodes t_1, t_2, t_3 is the same node t, then it can be easily seen that $X = X_t$ is the required separator as only at most one of C_1, C_2, and C_3 remains after its deletion.

Hence assume that all three nodes t_1, t_2, t_3 are distinct. From Proposition 10, we know that there exists a node $t \in V(T)$ such that (i) t_1, t_2 and t_3 are in different connected components of $T - t$. We claim that $X = X_t$ is the required separator. Since X is a bag in the $(1,k)$-semi clique tree decomposition $\mathcal{T}$, $G[X]$ is a $(1,k)$-semi clique. Because of statement (i), we have that X is a separator of C_i and C_j for all $i,j \in \{1,2,3\}$ and $i \neq j$. The proof is not constructive as we do not have a $(1,k)$-semi clique tree decomposition of G.

We compute a set X' such that $G[X']$ is a $(1,k)$-semi clique and X' is a separator of C_i and C_j for all $i,j \in \{1,2,3\}$ and $i \neq j$, without the knowledge of a $(1,k)$-semi clique tree decomposition of G. Let $X = C'' \uplus N''$ where C'' is a clique and $|N''| \leq k$. Using Lemma 12, we either conclude that G has no CVD of size k or we go over all the maximal cliques of the graph G. We know that $C'' \subseteq D$ for one of such maximal cliques D. Now in the graph $G[V \setminus D]$, we know that there exists a set $Z \subseteq N'' = X \setminus C''$ of size at most k which separates the cliques $C_x \setminus D, C_y \setminus D$ and $C_z \setminus D$. To find Z, we add three new vertices x', y' and z'. We make x' adjacent to all the vertices of $C_x \setminus D$, y' adjacent to all the vertices of $C_y \setminus D$ and z' adjacent to all the vertices of $C_z \setminus D$. We find the node multiway cut Y of size at most k with the terminal set being $\{x', y', z'\}$. The set Y can be found in $\mathcal{O}(2^k km)$ using the known algorithm for node multiway cut [13, 25]. If the algorithm returns that there is no such set Y of size k, we conclude that there is no CVD of size at most k in G. Else we get a set $X' = D \uplus Y$ which satisfies the properties of X.

Now we estimate the time taken to obtain X'. We get all the $\mathcal{O}(2^k \cdot n)$ maximal cliques of the graph in $\mathcal{O}(2^k \cdot n^{\omega+1})$ time. Now for each maximal clique we use the $\mathcal{O}(2^k km)$ algorithm for node multiway cut. Thus, the overall running time is $\mathcal{O}(2^k \cdot n^{\omega+1} + (2^k n) \cdot (2^k km)) = \mathcal{O}(4^k \cdot (kn^3 + n^{\omega+1}))$. ◀

Now we prove our main result (i.e., Theorem 1) in this section. For convenience we restate it here.

Theorem 1. *There is an algorithm that given a graph G and an integer k runs in time $\mathcal{O}(2^{7k} \cdot (kn^4 + n^{\omega+2}))$ and either constructs a $(4, 7k+5)$-semi clique tree decomposition $\mathcal{T}$ of G or concludes that there is no chordal vertex deletion set of size k in G. Moreover, the algorithm also provides a partition $C_1 \uplus C_2 \uplus C_3 \uplus C_4 \uplus N$ of each bag of $\mathcal{T}$ such that $|N| \leq 7k+5$ and C_i is a clique in G for all $i \in \{1,2,3,4\}$.*

Proof. We assume that G is connected as if not we can construct a $(4, 7k+5)$-semi clique tree decomposition for each connected components of G and attach all of them to a root node whose bag is empty to get the required $(4, 7k+5)$-semi clique tree decomposition of G.

To construct a $(4, 7k+5)$-semi clique tree decomposition $\mathcal{T}$, we define a recursive procedure $\mathsf{Decompose}(W, S, d)$ where $S \subset W \subseteq V(G)$ and $d \in \{0,1,2\}$. The procedure returns a rooted $(4, 7k+5)$-semi clique tree decomposition of $G[W]$ such that S is contained in the root bag of the tree decomposition. The procedure works under the assumption that the following invariants are satisfied.

- $G[S]$ is a $(d, 6k+4)$-semi clique and $W \setminus S \neq \emptyset$.
- $S = N_G(W \setminus S)$. Hence S is called the *boundary* of the graph $G[W]$.

To get the required $(4, 7k+5)$-semi clique tree decomposition of G, we call Decompose$(V(G), \emptyset, 0)$ which satisfies all the above invariants. The procedure Decompose(W, S, d) calls procedures Decompose(W', S', d') and a new procedure SplitCliques(W', S') whenever $d = 2$. For these subprocedures, we will show that $|W' \setminus S'| < |W \setminus S|$. Hence by induction on cardinality of $W \setminus S$, we will show the correctness of the Decompose procedure.

The procedure SplitCliques(W, S) with $S \subset W \subseteq V(G)$ also outputs a rooted $(4, 7k+5)$-semi clique tree decomposition of $G[W]$ such that S is contained in the root bag of the tree decomposition. But the invariants under which it works are slightly different which we list below.

- $G[S]$ is a $(3, 5k+3)$-semi clique and $W \setminus S \neq \emptyset$.
- $S = N_G(W \setminus S)$.

Notice that the only difference between invariants for Decompose and SplitCliques is the first invariant where we require $G[S]$ to be a $(3, 5k+3)$-semi clique for SplitCliques and $(d, 6k+4)$-semi clique for Decompose.

The procedure SplitCliques(W, S) calls procedures Decompose$(W', S', 2)$ where we will again show that $|W' \setminus S'| < |W \setminus S|$. Hence again by induction on cardinality of $W \setminus S$, we will show the correctness. Now we describe how the procedure Decompose is implemented.

Implementation of Decompose(W, S, d): Notice that $d \in \{0, 1, 2\}$. Firstly, if $|W \setminus S| \leq k+1$, we output the tree decomposition as a node r with bag $X_r = W$ and stop. Clearly the graph $G[X_r]$ is a $(4, 7k+5)$-semi clique and it contains S. Otherwise, we do the following.

We construct a set $\hat{S}$ with the following properties.

1. $S \subset \hat{S} \subseteq W \subseteq V(G)$.
2. $G[\hat{S}]$ is a $(d+1, 7k+5)$-semi clique. Let $\hat{S} = C' \uplus N'$ where $G[C']$ is the union of $d+1$ cliques and $|N'| \leq 7k+5$.
3. Every connected component of $G[W \setminus \hat{S}]$ is adjacent to at most $5k+3$ vertices of N'.

Since $G[S]$ is a $(d, 6k+4)$-semi clique, we have that $S = C \uplus N$, where $G[C]$ is the union of d cliques and $|N| \leq 6k+4$.

Case 1: $|N| < 5k+3$. We set $\hat{S} = S \cup \{u\}$, where u is an arbitrary vertex in $W \setminus S$. Note that this is possible as $W \setminus S \neq \emptyset$. Clearly $\hat{S}$ follows all the properties above.

Case 2: $5k+3 \leq |N| \leq 6k+4$. Note that $G[W]$ being a subgraph of G also has a chordal vertex deletion set of size at most k if G has it. Applying Corollary 14 for the graph $G[W]$ and the subset N, we either conclude that G has no CVD of size k or get a partition (N_A, N_B) of N, a subset $X \subseteq W$ and a partition $D \uplus Z$ of X, where D is a clique in $G[W]$ and $|Z| \leq k$, in time $\mathcal{O}(2^{7k} \cdot (kn^3 + n^{\omega+1}))$ such that $|N_A|, |N_B| \leq 4k+2$ and X is a vertex separator of N_A and N_B in the graph $G[W]$.

We define $\hat{S} = S \cup X \cup \{u\}$ where u is an arbitrary vertex in $W \setminus S$. We need to verify that $\hat{S}$ satisfies the required properties.

▷ Claim 16. The set $\hat{S}$ satisfies properties (1), (2) and (3).

Proof. Since $u \in W \setminus S$, $S \subset \hat{S}$. Hence $\hat{S}$ satisfies property (1).

We now show that $\hat{S}$ satisfies property (2). Recall that $S = C \uplus N$, where $G[C]$ is the union of d cliques and $|N| \leq 6k+4$. We define sets $C' = C \cup D$ and $N' = ((N \cup Z) \setminus C') \cup \{u\}$ Notice that $\hat{S} = C' \cup N'$. Clearly $G[C']$ is the union of $d+1$ cliques. Also $|N'| \leq |N| + |Z| + 1 \leq (6k+4) + k + 1 \leq 7k+5$. Thus $\hat{S}$ satisfies property (2).

We now show that $\hat{S}$ satisfies property (3). Recall $\hat{S} = C' \cup N'$, where $C' = C \cup D$ and $N' = ((N \cup Z) \setminus C') \cup \{u\}$. Recall that $X = D \cup Z \subseteq \hat{S}$ is separator of N_A and N_B. where $N = N_A \uplus N_B$ and $|N_A|, |N_B| \leq 4k + 2$. This implies that any connected component H in $G[W \setminus X]$ can contain at most $4k + 2$ vertices from N as the neighborhood of $V(H)$ is contained in X, because X is a separator. Moreover $|Z| \leq k$. This implies that any connected component in $G[W \setminus \hat{S}]$ is adjacent to at most $4k + 2$ vertices in N and at most k vertices in Z, and hence at most $5k + 3$ vertices in $N' = ((N \cup Z) \setminus C') \cup \{u\}$. ◁

Now we define the recursive subproblems arising in the procedure Decompose (W, S, d) using the constructed set $\hat{S}$. If $\hat{S} = W$, then there will not be any recursive subproblem. Otherwise, let $P_1, P_2, \ldots, P_q$ be vertex sets of the connected components of $G[W \setminus \hat{S}]$ and $q \geq 1$ because $\hat{S} \neq W$. We have the following cases:

Case 1: $d < 2$. For each $i \in [q]$, recursively call the procedure Decompose$(W' = N_G[P_i], S' = N_G(P_i), d + 1)$.

We now show that the invariants are satisfied for procedures Decompose$(W' = N_G[P_i], S' = N_G(P_i), d + 1)$ for all $i \in [q]$. We start by noticing that since $d < 2$, $d + 1 \leq 2$ which is required for the validity of the procedure. Let $Q_i = S' \cap N'$. Note that from condition (3) for $\hat{S}$, we have $|Q_i| \leq 5k + 3$. Since $S' \setminus Q_i \subseteq C'$ and $G[C']$ is a union of $d + 1$ cliques, $G[S']$ forms a $(d + 1, 5k + 3)$-semi clique which is also a $(d + 1, 6k + 4)$-semi clique. Also by definition of neighbourhoods, $P_i = N_G[P_i] \setminus N_G(P_i) = W' \setminus S'$. Since P_i is a non-empty set by definition, $W' \setminus S'$ is non-empty. Hence the first invariant required for the Decompose is satisfied. Since $S' = N_G(P_i) = N_G(N_G[P_i] \setminus N_G(P_i)) = N_G(W' \setminus S')$, the second invariant is satisfied.

Case 2: $d = 2$. For each $i \in [q]$, recursively call the procedure SplitCliques$(W' = N_G[P_i], S' = N_G(P_i))$. We can show that the invariants for SplitCliques are satisfied with the proofs similar to previous case.

We now explain how to construct the $(4, 7k + 5)$-semi clique tree decomposition using Decompose(W, S, d). Here, we assume that Decompose$(W', S', d+1)$ and SplitCliques(W', S') return a $(4, 7k + 5)$-semi clique tree decomposition $G[W']$ when $|W' \setminus S'| < |W \setminus S|$. That is, we apply induction on $|W \setminus S|$. Look at the subprocedures Decompose(W', S', d) and SplitCliques(W', S'). We have $W' \setminus S' = N_G[P_i] \setminus N_G(P_i) = P_i$ which is a subset of $W \setminus \hat{S}$ which in turn is a strict subset of $W \setminus S$. Hence $|W' \setminus S'| < |W \setminus S|$. Hence we apply induction on $|W \setminus S|$ to the subprocedures. Let $\mathcal{T}_i$ be the $(4, 7k + 5)$-semi clique tree decomposition obtained from the subprocedure with $W' = N_G[P_i]$ and $S' = N_G(P_i)$. Let r_i be the root of $\mathcal{T}_i$ whose associated bag is X_{r_i}. By induction hypothesis $S' \subseteq X_{r_i}$. We create a node r with the corresponding bag $X_r = \hat{S}$. For each $i \in [q]$, we attach $\mathcal{T}_i$ to r by adding edge (r, r_i). Let us call the tree decomposition obtained so with root r as $\mathcal{T}$. We return $\mathcal{T}$ as the output of Decompose(W, S, d). By construction, it easily follows that $\mathcal{T}$ is a $(4, 7k + 5)$-semi clique tree decomposition of the graph $G[W]$ with the root bag containing S. We note that when $W = \hat{S}$, the procedure returns a single node tree decomposition with $X_r = W = \hat{S}$.

Implementation of SplitCliques Procedure: Again if $|W \setminus S| \leq k + 1$, we output the tree decomposition as a node r with bag $X_r = W$ and stop. Clearly the graph $G[X_r]$ is a $(4, 7k + 5)$-semi clique and it contains S. Otherwise we do the following. Let $S = C \uplus N = (C_x \uplus C_y \uplus C_z) \uplus N$ where C_x, C_y and C_z are the vertex sets of the three cliques in $G[C]$. We apply Lemma 15 to graph $G[W]$ and sets C_x, C_y and C_z, to either conclude that G has no CVD of size k or obtain a set Y such that Y separates the sets C_x, C_y and C_z and $G[Y]$ is a $(1, k)$-semi clique. Let $Y = D \uplus X$ where D is a clique and $|X| \leq k$.

Let $Y' = Y \cup \{u\}$ where u is any arbitrary vertex from $W \setminus S$ which we know to be non-empty. If $S \cup Y' = W$, then it will not call any recursive subproblem. Otherwise, let $P_1, P_2, \ldots, P_q$ be the connected components of the graph $G[W \setminus (S \cup Y')]$. We recursively call Decompose$(W' = N_G[P_i], S' = N_G(P_i), 2)$ for all $i \in [q]$.

Since Y' is a separator of the cliques C_x, C_y and C_z, any connected component P_i will have neighbours to at most one of the three cliques $C_x \setminus Y', C_y \setminus Y'$ and $C_z \setminus Y'$ in $G[W \setminus (S \cup Y')]$. We show that the invariants required for the procedure Decompose are satisfied in these subproblems. Let us focus on the procedure Decompose$(W' = N_G[P_i], S' = N_G(P_i), 2)$ which has neighbours only to the set $C_x \setminus Y'$. We define sets $C' = C_x \cup D$ and $N' = (N \cup X \cup \{u\}) \setminus C'$. The vertex set P_i has neighbours only to the set $(C_x \uplus N) \cup Y' = (C_x \uplus N) \cup (D \uplus X) \cup \{u\} = (C_x \cup D) \cup (N \cup X \cup \{u\}) = C' \uplus N'$. Clearly $G[C']$ is the union of at most two cliques and $|N'| \leq |N| + |X| + 1 = 5k + 3 + k + 1 \leq 6k + 4$. Hence the first invariant is satisfied for the procedure Decompose$(N_G[P_i], N_G(P_i), 2)$. The proof of the second invariant is the same as to that of the subproblems of Decompose procedure. The satisfiability of invariants for other subprocedures can also be proven similarly.

We now construct the $(4, 7k + 5)$-semi clique tree decomposition returned by SplitCliques (W, S). Again we apply induction on $|W \setminus S|$. Consider the subprocedures Decompose(W', S', d). We have $W' \setminus S' = N_G[P_i] \setminus N_G(P_i) = P_i$ which is a subset of $W \setminus (S \cup Y')$ which in turn is a strict subset of $W \setminus S$ as $u \in W \setminus S$ is present in Y'. Hence $|W' \setminus S'| < |W \setminus S|$ and we apply induction on $|W \setminus S|$ to the subprocedures. Let $\mathcal{T}_i$ be the $(4, 7k + 5)$-semi clique tree decomposition obtained from the subprocedure with $W' = N_G[P_i]$ and $S' = N_G(P_i)$. Let r_i be the root of $\mathcal{T}_i$ whose bag X_{r_i} we show contains S'. We create a node r with the corresponding bag $X_r = S \cup Y' = (C_x \uplus C_y \uplus C_z \uplus D) \uplus N'$. For each $i \in [q]$, we attach $\mathcal{T}_i$ to r by adding edge (r, r_i). Let us call the tree decomposition obtained so with root r as $\mathcal{T}$. We return $\mathcal{T}$ as the output of SplitCliques(W, S, d). By construction, it easily follows that $\mathcal{T}$ is a $(4, 7k + 5)$-semi clique tree decomposition of the graph $G[W]$ with the root bag containing S. We mention that when $W = S \cup Y'$, the procedure returns a single node tree decomposition with $X_r = W$.

Running time analysis: In the procedure Decompose, we invoke Corollary 14 which takes $\mathcal{O}(2^{7k} \cdot (kn^3 + n^{\omega+1}))$ time. For the procedure SplitCliques, we invoke Lemma 15 which takes $\mathcal{O}(4^k \cdot (kn^3 + n^{\omega+1}))$ time. All that is left is to bound the number of calls of the procedures Decompose and SplitCliques. Each time Decompose or SplitCliques is called, it creates a set $\hat{S}$ (in the case of SplitCliques, $\hat{S} = S \cup Y'$) which is a strict superset of S. This allows us to map each call of Decompose or SplitCliques to a unique vertex $u \in \hat{S} \setminus S$ of $V(G)$. Hence the total number of calls of Decompose and SplitCliques is not more than the total number of vertices n. Hence the overall running time of the algorithm which constructs the $(4, 7k + 5)$-semi clique tree decomposition of G is $\mathcal{O}(2^{7k} \cdot (kn^4 + n^{\omega+2}))$. ◀

4 Structural Parameterizations with Chordal Vertex Deletion Set

▶ **Theorem 17.** Weighted d-Colorable Subgraph by CVD *can be solved in* $d^{4d+7k+5} 2^{3(7k+5)} n^{\mathcal{O}(d)}$ *time.*

Proof. First, we use Theorem 1 to construct a $(4, 7k + 5)$-semi clique tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G in $\mathcal{O}^*(2^{7k})$ time. Now we use the dynamic programming algorithm on tree decompositions given by Fomin and Golovach (Theorem 1 of [18]) on $\mathcal{T}$ to find the maximum sized induced subgraph H of G such that H is d-colorable. Note that the set $V(G) \setminus V(H)$ is the solution that we are looking for and if its weight is at most ℓ, we return YES. Else we return NO.

This dynamic programming algorithm defines a state $cost(t, S, c)$ for all nodes $t \in V(T)$, subsets $S \subseteq X_t$ such that $G[S]$ is d-colorable and a function $c : S \to [d]$. Since each bag X_t of $\mathcal{T}$ is a $(4, 7k + 5)$ semi clique, at most d vertices of each clique can be part of S as else there is a presence of a $(d + 1)$ sized clique in S which is not d-colorable. Hence we can bound the size of S as $4d + 7k + 5$ and also bound the number of possible subsets S as $n^{4d}2^{7k+5}$. Number of possible functions c is at most $d^{|S|}$ which is at most $d^{4d+7k+5}$. Hence we bound the number of states as $d^{4d+7k+5}2^{7k+5}n^{\mathcal{O}(d)}$. For each state, the time taken is $\mathcal{O}(|S|^2)$. Hence the overall running time is $d^{4d+7k+5}2^{3(7k+5)}n^{\mathcal{O}(d)}$. ◀

▶ **Corollary 18.** Weighted Vertex Cover by CVD *and* Weighted OCT by CVD *can be solved in* $2^{21k}n^{\mathcal{O}(1)}$ *and* $2^{28k}n^{\mathcal{O}(1)}$ *time, respectively.*

We can directly use the dynamic programming on bounded treewidth to get algorithms with better running times for WVC By CVD and WOCT by CVD and for WFVS by CVD using the fact that any vertex cover contains all but one from each clique and any odd cycle transversal and feedback vertex set contains all but two from each clique.

▶ **Theorem 19** ($\star$).[2] *Given a graph G and an integer k, there exist algorithms that determines that G has no CVD of size k or*

- *find a minimum weighted vertex cover in* $2^{7k}n^{\mathcal{O}(1)}$ *time.*
- *find a minimum weighted odd cycle transversal in* $3^{7k}n^{\mathcal{O}(1)}$ *time.*
- *find a minimum weighted feedback vertex set in* $2^{\omega 7k}n^{\mathcal{O}(1)}$ *time.*

Proof.

Proof sketch of algorithm for WVC By CVD: First, we use Theorem 1 to construct a $(4, 7k + 5)$-semi clique tree decomposition $\mathcal{T} = (T, \{X_t\}_{t\in V(T)})$ of G in $\mathcal{O}^*(2^{7k})$ time. In the tree decomposition $\mathcal{T} = (T, \{X_t\}_{t\in V(T)})$, for any vertex $t \in V(T)$, we call D_t to be the set of vertices that are descendant of t. We define G_t to be the subgraph of G on the vertex set $X_t \cup \bigcup_{t'\in D_t} X_{t'}$. We briefly explain the DP table entries on $\mathcal{T}$. Arbitrarily root the tree T at a node r. Let $X_t = C_{t,1} \uplus \ldots \uplus C_{t,4} \uplus N_t$ where $|N_t| \leq 7k + 5$ and $C_{t,j}$ is a clique in G for all $j \in \{1, \ldots, 4\}$. In a standard DP for each node $t \in V(T)$ and $Y \subseteq X_t$, we have a table entry $DP[Y, t]$ which stores the value of a minimum vertex cover S of G_t such that $Y = X_t \cap S$ and if no such vertex cover exists, then $DP[Y, t]$ stores ∞. In fact we only need to store $DP[Y, t]$ whenever it is not equal to ∞. Now consider a bag X_t in $\mathcal{T}$. For any $Y \subseteq X_t$, if $|C_{t,j} \setminus Y| \geq 2$ for any $j \in [4]$, then $DP[Y, t] = \infty$ because $C_{t,j}$ is a clique. Therefore, we only need to consider subsets $Y \subseteq X_t$ for which $|C_{t_j} \setminus Y| \leq 1$ for all $j \in [4]$. The number of choices of such subsets Y is bounded by $\mathcal{O}(2^{7k}n^4)$. This implies that the total number of DP table entries is $\mathcal{O}(2^{7k}n^5)$. All these values can be computed in time $\mathcal{O}(2^{7k}n^{\mathcal{O}(1)})$ time using standard dynamic programming in a bottom up fashion. For more details about dynamic programming over tree decomposition, see [12].

Proof sketch of algorithm for Weighted OCT By CVD: Let $\mathcal{T} = (T, \{X_t\}_{t\in V(T)})$ be a $(4, 7k + 5)$-semi clique tree decompositionn of the input graph G. For any $t \in V(T)$, let $X_t = C_{t,1} \uplus \ldots \uplus C_{t,4} \uplus N_t$ where $|N_t| \leq 7k + 5$ and $C_{t,j}$ is a clique in G for all $j \in \{1, \ldots, 4\}$. Then, any odd cycle transversal contains all but at most two vertices from each clique $C_{1,j}$, $i \in [4]$. Using this fact we can bound the number of DP table entries to be at most $3^{7k}n^{\mathcal{O}(1)}$. Then, by compute the entries in a bottom up fashion in time $3^{7k}n^{\mathcal{O}(1)}$ using standard arguments.

[2] A more detailed proof is available in the full version of the paper.

Proof sketch of algorithm for Weighted FVS By CVD: We use the ideas from the DP algorithm for Feedback Vertex Set using the rank based approach [3]. We again use Theorem 1 to construct a $(4, 7k+5)$-semi clique tree decomposition of G. We create an auxiliary graph G' by adding a vertex v_0 to G and making it adjacent to all the vertices of G. Let E_0 be the set of newly added edges. Thus we add v_0 to all the bags to get the tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G' and this can be done in $\mathcal{O}^*(2^{7k})$ time. We use a dynamic programming algorithm for Feedback Vertex Set on $\mathcal{T}$ where the number of entries of the DP table we will show to be $2^{7k+5}n^{11}$. Let $X_t = C_{t,1} \uplus \ldots \uplus C_{t,4} \uplus N_t$ for all $t \in V(T)$ where $|N_t| \leq 7k+5$ and $C_{t,j}$ is a clique in G for all $j \in \{1, \ldots, 4\}$. For a node $t \in V(T)$, a subset $Y \subseteq X_t$ and integers $i, j \in [n]$, we define the entry $DP[t, Y, i, j]$. The entry $DP[t, Y, i, j]$ stores a partition $\mathcal{P}$ of Y if

- there exists a vertex subset $X \subseteq D_t$, $v_0 \in X$ such that $X \cap X_t = Y$ and
- there exists an edge subset $X_0 \subseteq E(G_t) \cap E_0$ such that in the graph $(X, E(G_t[X \setminus \{v_0\}]) \cup X_0)$, we have i vertices, j edges, no connected component is fully contained in $D_t \setminus X_t$ and the elements of Y are connected according to the partition $\mathcal{P}$.

We set $DP[t, Y, i, j] = \infty$ if the entry can be inferred to be invalid from Y.

We can claim that Feedback Vertex Set by CVD is a yes instance if and only if for the root r of $\mathcal{T}$ with $X_r = \{v_0\}$ and some $i \geq |V| - \ell$, we have $DP[r, \{v_0\}, i, i-1]$ to be non-empty.

The recurrences for computing $DP[t, Y, i, j]$ remains very similar to that in [3]. Since the number of table entries is $\mathcal{O}(2^{7k}n^{13})$, U is at most $7k+13$ and $\mathcal{A}$ is at most $2^{|U|}$, we have the total time bounded to be $\mathcal{O}(2^{(\omega-1)7k}(7k)^{\mathcal{O}(1)}2^{7k}n^{13}) = \mathcal{O}(2^{\omega 7k}(7k)^{\mathcal{O}(1)}n^{13})$. Using the ideas from [3], it can be proven that the number of table entries is $\mathcal{O}(2^{7k}n^{13})$ and the total time is $\mathcal{O}(2^{\omega 7k}(7k)^{\mathcal{O}(1)}n^{13})$. ◀

4.1 SETH Lower Bounds

A graph G is called a cluster graph if it is a disjoint union of complete graphs. It can be seen that all cluster graphs are chordal. We define a problem called Vertex Cover by ClsVD.

Vertex Cover by ClsVD
Input: A graph $G = (V, E)$, $k, \ell \in \mathbb{N}$ and a set $S \subseteq V(G)$ with $|S| \leq k$ such that $G[V \setminus S]$ is a cluster graph.
Parameter: k
Question: Is there a vertex cover of size ℓ in G?

Assuming SETH, we show that Vertex Cover by ClsVD, FVS by CVD and OCT by CVD cannot have an $\mathcal{O}^*((2-\epsilon)^k)$ FPT algorithm. As the class of all cluster graphs is a subclass of the class of chordal graphs, deletion distance to a chordal graph is a smaller parameter. Hence the lower bound also holds for WVC By CVD.

To show the following theorem, we give a parameterized reduction from Hitting Set parameterized by the size of the universe n to Vertex Cover by ClsVD and use the fact that assuming SETH, Hitting Set cannot be solved in $\mathcal{O}^*((2-\epsilon)^n)$ time.

▶ **Theorem 20.** Vertex Cover by ClsVD *cannot be solved in* $\mathcal{O}^*((2-\epsilon)^k)$ *time for any* $\epsilon > 0$ *assuming SETH.*

Proof. We give a reduction from Hitting Set defined as follows.
Hitting Set : In any instance of Hitting Set, we are given a set of elements U with $|U| = n$, a family of subsets $\mathcal{F} = \{F \subseteq U\}$ and a natural number k. The objective is to find a set $F \subseteq U$, $|F| \leq k$ such that $S \cap F \neq \emptyset$ for all $S \in \mathcal{F}$.

The problem cannot be solved in $\mathcal{O}^*((2-\epsilon)^n)$ time assuming SETH [11].

Consider a Hitting Set instance $(U, \mathcal{F})$. We construct an instance of Vertex Cover by ClsVD as follows. For each element $u \in U$, we add a vertex v_u. For each set $S \in \mathcal{F}$, we add $|S|$ vertices corresponding to the elements in S. We also make the vertices of S into a clique. Finally, for each element $u \in U$, we add edges from v_u to the vertex corresponding to u for each set in S that contains u. See Figure 1.

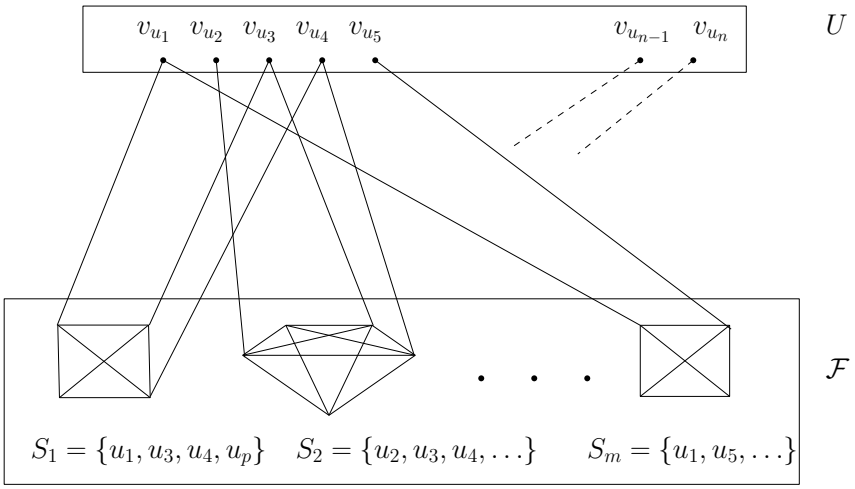


Figure 1 Reduction from Hitting Set to Vertex Cover by ClsVD.

Note that the set of vertices $\bigcup_{u \in U} v_u$ forms a cluster vertex deletion set of size n for the graph G we constructed.

We claim that there is a hitting set of size k in the instance $(U, \mathcal{F})$ if and only if there is a vertex cover of size $k + \sum_{S \in \mathcal{F}} (|S| - 1)$ in G.

Let $X \subseteq U$ be the hitting set of size k. For each set $S \in \mathcal{F}$, mark an element of X which intersects S. Now we create a subset of vertices Y in G consisting of vertices corresponding to elements in X plus the vertices corresponding to all the unmarked elements in S for every set $S \in \mathcal{F}$. Clearly $|Y| = k + \sum_{S \in \mathcal{F}} (|S| - 1)$. We claim that Y is a vertex cover of G. Let us look at an edge of G between an element vertex u and its corresponding copy vertex in S containing u. If u is unmarked in S, then it is covered as the vertex corresponding to u in S is present in Y. If it is marked, then the element v_u is present in Y which covers the edge. All the other edges of G have both endpoints in a set $S \in \mathcal{F}$. Since one of them is unmarked, it belongs to Y which covers the edge.

Conversely, let Z be a vertex cover of G of size $k + \sum_{S \in \mathcal{F}} (|S| - 1)$. Since the graph induced on vertices of set S forms a clique for each $S \in \mathcal{F}$, Z should contain all the vertices of the clique except one to cover all the edges of the clique. Let us mark these vertices. This means that at least $\sum_{S \in \mathcal{F}} (|S| - 1)$ of the vertices of Z are not element vertices v_u. Now the remaining k vertices of Z should hit all the remaining edges in G. Suppose it contains another vertex x corresponding to an element u in set $S \in \mathcal{F}$. Since x can only cover the edge from x to the element vertex v_u out of the remaining edges, we could remove x and add v_u as it is not present in Z and still get a vertex cover of G of the same size. Hence we can assume, without loss of generality that all the remaining vertices of Z are element vertices v_u. Let X' be the union of the k elements corresponding to these element vertices. We claim that

X' is a hitting set of $(U, \mathcal{F})$ of size k. Suppose X' does not hit a set $S \in \mathcal{F}$. Look at the unmarked vertex x in the vertices of S. There is an edge from x to its element vertex v_u. Since $u \notin X'$, this edge is uncovered in G giving a contradiction.

Hence given a Hitting Set instance $(U, \mathcal{F})$, we can construct an instance for Vertex Cover by ClsVD with parameter n. Hence, if we could solve Vertex Cover by ClsVD in $\mathcal{O}^*((2-\epsilon)^k)$ time, we can solve Hitting Set in $\mathcal{O}^*((2-\epsilon)^n)$ time contradicting SETH. ◀

The proof of the following theorem works by modifying the reduction in the above proof to replace edges by triangles.

▶ **Theorem 21.** FVS by CVD *and* OCT by CVD *given the modulator cannot be solved in* $\mathcal{O}^*((2-\epsilon)^k)$ *time for any* $\epsilon > 0$ *assuming SETH.*

Proof. To prove the above theorem, we again give a reduction very similar to the reduction given in the proof of Theorem 20. Consider a Hitting Set instance $(U, \mathcal{F})$. To create an instance of Feedback Vertex Set by CVD or Odd Cycle Transversal by CVD, we replace each edge in the above reduction by a triangle. It can be easily shown that the graph obtained after removing the vertices corresponding to elements in U forms a chordal graph. The proof follows on similar lines. ◀

5 Conclusion

Our main contribution is to develop techniques for addressing structural parameterization problems when the modulator is not given. The question, of Fellows et al. about whether there is an FPT algorithm for Vertex Cover parameterized by perfect deletion set with only a promise on the size of the deletion set, is open. Regarding problems parameterized by chordal deletion set size, though our algorithms are based on treewidth DP, we remark that not all problems that have FPT algorithms when parameterized by treewidth necessarily admit an FPT algorithm parameterized by CVD. For example, Dominating Set parameterized by treewidth admits an FPT algorithm [12] while Dominating Set parameterized by CVD is para-NP-hard as the problem is NP-hard in chordal graphs [5]. Generalizing our algorithms for other problems, for example, for the optimization problems considered by Liedloff et al. [31] would be an interesting direction.

Finally, we believe that this whole notion of permissive problems need to be explored in many facets of structural parameterizations where finding the modulator is more expensive than solving the problem when the modulator is given.

References

1 Akanksha Agrawal, Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Polylogarithmic approximation algorithms for Weighted-F-Deletion Problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, pages 1:1–1:15, 2018.

2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

3 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.

4 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A c^kn 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016.

5 Kellogg S. Booth and J. Howard Johnson. Dominating sets in chordal graphs. *SIAM J. Comput.*, 11:191–199, 1982.

6 Andreas Brandstädt. On robust algorithms for the maximum weight stable set problem. In *International Symposium on Fundamentals of Computation Theory*, pages 445–458. Springer, 2001.

7 Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Comput. Geom.*, 9(1-2):3–24, 1998. doi:10.1016/S0925-7721(97)00014-X.

8 Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016.

9 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

10 Derek G. Corneil and Jean Fonlupt. The complexity of generalized clique covering. *Discrete Applied Mathematics*, 22(2):109–118, 1988.

11 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016. doi:10.1145/2925416.

12 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 3. Springer, 2015.

13 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *TOCT*, 5(1):3:1–3:11, 2013. doi:10.1145/2462896.2462899.

14 R. Diestel. *Graph Theory*. Springer, 2006.

15 Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

16 Michael Fellows and Frances Rosamond. The complexity ecology of parameters: an illustration using bounded max leaf number. In *Conference on Computability in Europe*, pages 268–277. Springer, 2007.

17 Michael R. Fellows, Bart M.P. Jansen, and Frances Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European Journal of Combinatorics*, 34(3):541–566, 2013.

18 Fedor V. Fomin and Petr A. Golovach. Subexponential parameterized algorithms and kernelization on almost chordal graphs. *arXiv preprint*, 2020. arXiv:2002.08226.

19 Fedor V. Fomin, Petteri Kaski, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Parameterized single-exponential time polynomial space algorithm for steiner tree. In *International Colloquium on Automata, Languages, and Programming*, pages 494–505. Springer, 2015.

20 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM J. Comput.*, 44(1):54–87, 2015. doi:10.1137/140964801.

21 Serge Gaspers and Stefan Szeider. Backdoors to satisfaction. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 287–317. Springer, 2012. doi:10.1007/978-3-642-30891-8_15.

22 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004.

23 Michel Habib and Christophe Paul. A simple linear time algorithm for cograph recognition. *Discrete Applied Mathematics*, 145(2):183–197, 2005.

24 Pinar Heggernes, Pim van 't Hof, Bart M. P. Jansen, Stefan Kratsch, and Yngve Villanger. Parameterized complexity of vertex deletion into perfect graph classes. *Theor. Comput. Sci.*, 511:172–180, 2013. doi:10.1016/j.tcs.2012.03.013.

25 Yoichi Iwata, Yutaro Yamaguchi, and Yuichi Yoshida. 0/1/all csps, half-integral a-path packing, and linear-time FPT algorithms. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 462–473. IEEE, 2018.

26 Bart M.P. Jansen. *The power of data reduction: Kernels for fundamental graph problems*. PhD thesis, Utrecht University, 2013.

27 Bart M.P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited. *Theory of Computing Systems*, 53(2):263–299, 2013.

28 Bart M.P. Jansen and Marcin Pilipczuk. Approximation and kernelization for chordal vertex deletion. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1399–1418. Society for Industrial and Applied Mathematics, 2017.

29 Bart M.P. Jansen, Venkatesh Raman, and Martin Vatshelle. Parameter ecology for feedback vertex set. *Tsinghua Science and Technology*, 19(4):387–409, 2014.

30 Eun Jung Kim and O-joung Kwon. Erdős-pósa property of chordless cycles and its applications. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1665–1684. SIAM, 2018.

31 Mathieu Liedloff, Pedro Montealegre, and Ioan Todinca. Beyond classes of graphs with "few" minimal separators: FPT results through potential maximal cliques. *Algorithmica*, 81(3):986–1005, 2019. doi:10.1007/s00453-018-0453-2.

32 Daniel Lokshtanov, NS Narayanaswamy, Venkatesh Raman, M.S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Transactions on Algorithms (TALG)*, 11(2):15, 2014.

33 Diptapriyo Majumdar and Venkatesh Raman. Structural parameterizations of undirected feedback vertex set: FPT algorithms and kernelization. *Algorithmica*, 80(9):2683–2724, 2018.

34 Diptapriyo Majumdar, Venkatesh Raman, and Saket Saurabh. Polynomial kernels for vertex cover parameterized by small degree modulators. *Theory Comput. Syst.*, 62(8):1910–1951, 2018. doi:10.1007/s00224-018-9858-1.

35 Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. In *Scandinavian Workshop on Algorithm Theory*, pages 260–272. Springer, 2004.

36 Vijay Raghavan and Jeremy P. Spinrad. Robust algorithms for restricted domains. *J. Algorithms*, 48(1):160–172, 2003. doi:10.1016/S0196-6774(03)00048-8.

37 Neil Robertson and Paul D. Seymour. Graph minors. xiii. the disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995.

38 Jeremy P. Spinrad. *Efficient graph representations*. American Mathematical Society, 2003.

Parameterized Complexity of Geodetic Set

Leon Kellerhals
Technische Universität Berlin, Faculty IV, Algorithmics and Computational Complexity, Germany
https://kellerhals.io
leon.kellerhals@tu-berlin.de

Tomohiro Koana
Technische Universität Berlin, Faculty IV, Algorithmics and Computational Complexity, Germany
https://tomohirokoana.github.io/
tomohiro.koana@tu-berlin.de

Abstract

A vertex set S of a graph G is *geodetic* if every vertex of G lies on a shortest path between two vertices in S. Given a graph G and $k \in \mathbb{N}$, the NP-hard Geodetic Set problem asks whether there is a geodetic set of size at most k. Complementing various works on Geodetic Set restricted to special graph classes, we initiate a parameterized complexity study of Geodetic Set and show, on the negative side, that Geodetic Set is W[1]-hard when parameterized by feedback vertex number, path-width, and solution size, combined. On the positive side, we develop fixed-parameter algorithms with respect to the feedback edge number, the tree-depth, and the modular-width of the input graph.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases NP-hard graph problems, Shortest paths, Tree-likeness, Parameter hierarchy, Data reduction, Integer linear programming

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.20

Related Version A full version of the paper is available at https://arxiv.org/abs/2001.03098.

Funding *Tomohiro Koana*: Partially supported by the DFG projects FPTinP (NI 369/16) and MATE (NI 369/17).

Acknowledgements We thank Lucia Draque Penso (Ulm University) for suggesting studying Geodetic Set from a view of parameterized complexity, and we thank André Nichterlein and Rolf Niedermeier (both TU Berlin) for helpful feedback and discussion. We are also grateful to an anonymous reviewer for suggesting that the ILP instances in Section 4 can be solved more efficiently.

1 Introduction

Let G be an undirected, simple graph with vertex set $V(G)$ and edge set $E(G)$. The *interval* $I[u, v]$ of two vertices u and v of G is the set of vertices of G that are contained in any shortest path between u and v. In particular, $u, v \in I[u, v]$. For a set S of vertices, let $I[S]$ be the union of the intervals $I[u, v]$ over all pairs of vertices u and v in S. A set of vertices S is called *geodetic* if $I[S]$ contains all vertices of G. In this work we study the following problem (see an exemplary illustration in Figure 1):

Geodetic Set

Input: A graph G and an integer k.
Question: Does G have a geodetic set of cardinality at most k?

Atici [2] showed that Geodetic Set is NP-complete on general graphs, and it was shown that the hardness holds even if the graph is planar [8], subcubic [7], chordal, or bipartite chordal [11]. Although not stated, W[2]-hardness for the solution size k directly follows from the reduction for the latter result of Dourado et al. [11]. On the positive side, the

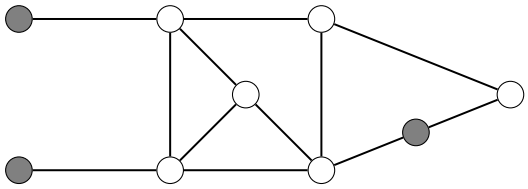


Figure 1 An exemplary graph. The gray vertices form a minimum geodetic set. The shortest paths between the top left and the bottom right gray vertex cover all vertices except for the bottom left vertex. Observe that every geodetic set contains all degree-one vertices.

problem was shown to be polynomial-time solvable for cographs, split graphs and unit interval graphs [11]. Also, upper bounds on the geodetic set size in Cartesian product graphs were studied [6].

For a graph G and $k \in \mathbb{N}$, the closely related Geodetic Hull problem asks whether there is a vertex set $S \subseteq V(G)$ with $I^{|V(G)|}[S] = V(G)$ and $|S| \leq k$, where $I^0[S] = S$ and $I^j[S] = I[I^{j-1}[S]]$ for $j > 0$. Geodetic Hull is NP-hard on bipartite [1], chordal [4], and P_9-free graphs [12]. Recently, Kanté et al. [17] studied the *parameterized complexity* of Geodetic Hull: they proved that the problem is W[2]-hard when parameterized by k, and W[1]-hard but in XP when parameterized by tree-width.[1]

Our Contributions. Comparing the algorithmic complexity of Geodetic Hull and Geodetic Set, one can observe that both problems are trivial on trees (take all leaves into the solution). But while Geodetic Hull is polynomial-time solvable on graphs of constant tree-width, the complexity of Geodetic Set on graphs of tree-width two is unknown to the best of our knowledge. Motivated by this gap, we study the parameterized complexity of Geodetic Set for structural parameters such as tree-width that measure the tree-likeness of the input graph, providing both positive and negative results.

We start off by showing that Geodetic Set is W[1]-hard with respect to tree-width. More specifically, we show that Geodetic Set is W[1]-hard for feedback vertex number, path-width, and solution size, all three combined (Section 3), using a parameterized reduction from the W[1]-hard Grid Tiling problem [20]. Since this reduction implies NP-hardness, this complements previous results by providing a more fine-grained view on computational tractability in terms of parameterized complexity instead of studying special graph classes.

We complement the W[1]-hardness by presenting two fixed-parameter tractability results for Geodetic Set. First, we show that Geodetic Set is fixed-parameter tractable with respect to the feedback edge number (Section 4). It turns out to be quite effortful to obtain fixed-parameter tractability, requiring the design and analysis of polynomial-time data reduction rules and branching before employing the main technical trick: Integer Linear Programming (ILP) with a bounded number of variables. To the best of our knowledge, this is the first usage of ILP when solving Geodetic Set.

Second, we show that Geodetic Set is fixed-parameter tractable with respect to clique-width combined with diameter (Section 5); note that Geodetic Set is NP-hard even on graphs with constant diameter [11], and W[1]-hard with respect to clique-width (this follows from our first result). Our result exploits the fact that we can express Geodetic Set in an MSO_1 logic formula, the length of which is upper-bounded in a function of the diameter of the graph. A direct consequence of this result is that Geodetic Set is fixed-parameter tractable with respect to tree-depth and with respect to modular-width.

[1] Informally, this means it can be solved in polynomial time for graphs of constant tree-width.

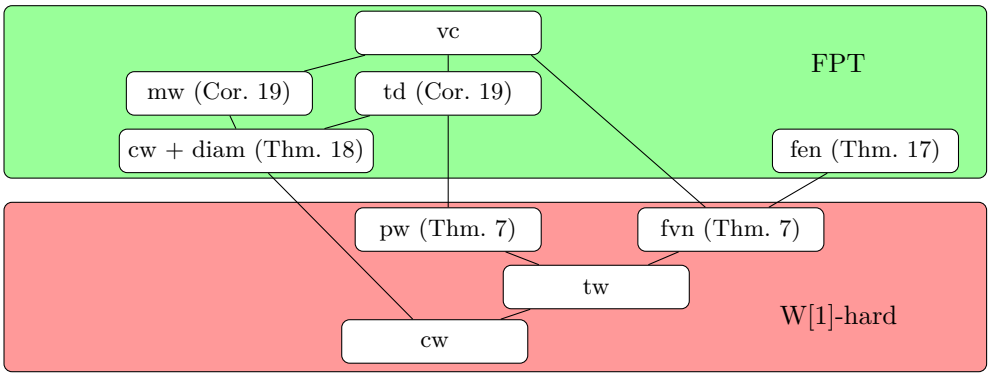


Figure 2 An overview of our results for Geodetic Set, containing the parameters vertex cover number (vc), modular-width (mw), tree-depth (td), clique-width (cw), diameter (diam), feedback edge number (fen), path-width (pw), feedback vertex number (fvn) and tree-width (tw). An edge between two parameters indicates that the one below is smaller than some function of the other.

Figure 2 gives an overview of the parameters for which we obtain positive and negative results, and presents their interdependence.

2 Preliminaries

For $n \in \mathbb{N}$ let $[n] = \{1, 2, \ldots, n\}$. The *distance* $d_G(u, v)$ between two vertices u and v in G is the length of a shortest path between u and v (also called *shortest u–v-path*). We drop the subscript $\cdot_G$ if G is clear from context. Note that w belongs to $I[u, v]$ if and only if $d_G(u, v) = d_G(u, w) + d_G(w, v)$. The *diameter* $\mathrm{diam}(G)$ of G is the maximum distance between any two vertices of G. A *multigraph* G consists of a vertex set and an edge multiset. Note that in a multigraph, we count self-loops twice for the vertex degree.

A set $F \subseteq E(G)$ is a *feedback edge set* if $G \setminus F$ is a forest. The *feedback edge number* $\mathrm{fen}(G)$ is the size of a smallest such set. Analogously, a set $V' \subseteq V(G)$ is a *feedback vertex set* if $G - V'$ is a forest. The *feedback vertex number* $\mathrm{fvn}(G)$ is the size of a smallest such set.

For a graph G, a *tree decomposition* is a pair (T, B), where T is a tree and $B \colon V(T) \to 2^{V(G)}$ such that (i) for each edge $uv \in E(G)$ there exists $x \in V(T)$ with $u, v \in B(x)$, and (ii) for each $v \in V(G)$ the set of nodes $x \in V(T)$ with $v \in B(x)$ forms a nonempty, connected subtree in T. The *width* of (T, B) is $\max_{x \in V(T)}(|B(x)| - 1)$. The *tree-width* $\mathrm{tw}(G)$ of G is the minimum width of all tree decompositions of G. The *path-width* $\mathrm{pw}(G)$ of G is the minimum width of all tree decompositions (T, B) of G for which T is a path.

The *tree-depth* of a connected graph G is defined as follows [21]. Let T be a rooted tree with vertex set $V(G)$, such that if $xy \in E(G)$, then x is either an ancestor or a descendant of y in T. We say that G is *embedded in* T. The *depth* of T is the number of vertices in a longest path in T from the root to a leaf. The *tree-depth* $\mathrm{td}(G)$ of G is the minimum t such that there is a rooted tree of depth t in which G is embedded.

We next define the *modular-width* of a graph G [15]. A vertex set $M \subseteq V(G)$ is a *module* if for all $u, v \in M$ it holds that $N(v) \cap V(G) \setminus M = N(w) \cap V(G) \setminus M$. We call a module M *trivial*, if $|M| \leq 1$ or $M = V$, and we call it *strong* if for every other module M' of G we have that $M \cap M' = \emptyset$, or that one is a subset of the other. A graph that only admits trivial modules is called *prime*. Every non-singleton graph can be uniquely partitioned into maximal strong modules $\mathcal{P} = \{M_1, \ldots, M_\ell\}$ with $\ell \geq 2$. Recursively partitioning the graphs $G[M_i]$ in this way until every module is a single vertex yields a *modular decomposition* of G. The modular-width is the largest number of trivial modules in a *prime subgraph* $G[M_i]$ of the modular decomposition of G.

A *parameterized problem* is a subset $L \subseteq \Sigma^* \times \mathbb{N}$ over a finite alphabet Σ. Let $f\colon \mathbb{N} \to \mathbb{N}$ be a computable function. A problem L is *fixed-parameter tractable (in* FPT*)* with respect to k if $(I,k) \in L$ is decidable in time $f(k) \cdot |I|^{O(1)}$ and L is in XP if $(I,k) \in L$ is decidable in time $|I|^{f(k)}$. There is a hierarchy of computational complexity classes for parameterized problems: FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \cdots \subseteq$ XP. To show that a parameterized problem L is (presumably) not in FPT one may use a *parameterized reduction* from a W[1]-hard problem to L. A parameterized reduction from a parameterized problem L to another parameterized problem L' is a function that acts as follows: For functions f and g, given an instance (I,k) of L, it computes in $f(k) \cdot |I|^{O(1)}$ time an instance (I',k') of L' so that $(I,k) \in L \iff (I',k') \in L'$ and $k' \leq g(k)$.

3 Hardness for Path-width and Feedback Vertex Number

In this section we show that Geodetic Set is W[1]-hard with respect to the feedback vertex number, the path-width and the solution size, combined. To this end, we present a parameterized reduction from Grid Tiling, which is W[1]-hard with respect to k [20]:

Grid Tiling

Input: A collection $\mathcal{S}$ of k^2 sets $S^{i,j} \subseteq [m] \times [m]$, $i,j \in [k]$ (called tile sets), each of cardinality exactly n.

Question: Can one choose a tile $(x^{i,j}, y^{i,j}) \in S^{i,j}$ for each $i,j \in [k]$ such that $x^{i,j} = x^{i,j'}$ with $j' = (j+1) \bmod k$ and $y^{i,j} = y^{i',j}$ with $i' = (i+1) \bmod k$?

This distinguishes our reduction from most parameterized reductions to show W[1]-hardness, as one typically reduces from Clique, or its multicolored variant. Grid Tiling though seemed to be a much better fit, since the values of the tiles can be expressed by lengths of paths. This is the central idea for our reduction: We place a connection gadget between each pair of adjacent tile sets. Placing paths of fitting lengths, the connection gadget ensures that the vertices corresponding to the tiles agree with each other, that is, the appropriate coordinates of the two tiles are equal.

▶ Remark. Throughout this section we write i' and j' as shorthands for $(i+1) \bmod k$ and $(j+1) \bmod k$, respectively. Moreover, we assume that the grid size k is even.

Construction. Let $I = (\mathcal{S}, k, m, n)$ be an instance of Grid Tiling. We construct an instance of Geodetic Set $I' = (G, k')$ as follows: First, we set $k' = k^2 + 4$. We add the *global vertices* $\Xi = \{\alpha, \beta, \gamma, \delta\}$ and $\Xi' = \{\alpha', \beta', \gamma', \delta'\}$, and add four edges $\alpha\alpha'$, $\beta\beta'$, $\gamma\gamma'$ and $\delta\delta'$. Next, for each $i,j \in [k]$ we introduce *tile vertices* $S^{i,j} = \{s_1^{i,j}, \ldots, s_n^{i,j}\}$. For a tile vertex v we denote by (x_v, y_v) the corresponding tile. Moreover, for each $i,j \in [k]$ we introduce two copies of the horizontal and two copies of the vertical connection gadget.

The construction of a *horizontal connection gadget next to tile set* $S^{i,j}$ is as follows. Let $S = S^{i,j}$ and let $S' = S^{i,j'}$ be the vertices of the two horizontally adjacent tile sets. We introduce the vertices a and b called *hidden vertices* and the vertices a^* and b^* called *exposed vertices*. Next, for every tile vertex $s \in S$ with its corresponding tile (x_s, y_s), we add a path of length $16m + 2x_s + 1$ from s to a, and a path of length $16m - 2x_s + 1$ from s to b. For every tile vertex $s' \in S'$ with its corresponding tile $(x_{s'}, y_{s'})$, we add a path of length $16m - 2x_{s'} + 1$ from s' to a, and a path of length $16m + 2x_{s'} + 1$ from s' to b. We call these paths *tile paths towards* S, respectively S'. We call the neighbors of a, respectively b, *connector vertices towards* S, respectively S'. The exposed vertices a^*, respectively b^* are adjacent to all neighbors of a, respectively b. Moreover, each of a^* and b^* has one additional

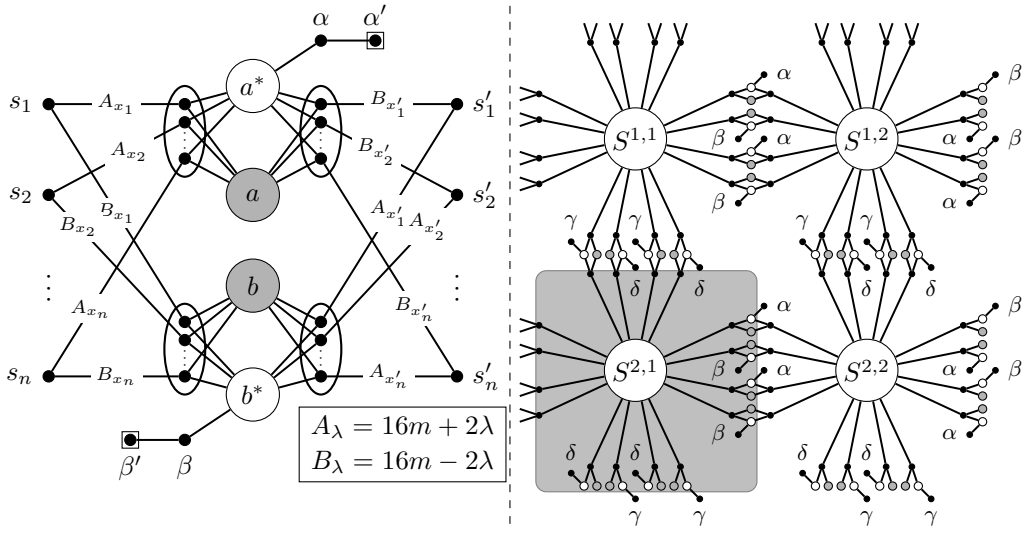


Figure 3 *Left:* One copy of a horizontal connection gadget next to $S^{i,j} = \{s_1, \ldots, s_n\}$ where j is even, connecting the tile sets $S^{i,j}$ and $S^{i,j'}$. Edges with label ℓ in the figure represent paths of length ℓ. The ellipses mark the connector vertices towards $S^{i,j}$ and $S^{i,j'}$. *Right:* An exemplary reduction from an instance of Grid Tiling, where $k = 2$. Between every pair of horizontally, resp. vertically adjacent tile sets (big circles) there are two copies of horizontal, resp. vertical connection gadgets. Note that $\alpha, \beta, \gamma, \delta \in \Xi$ are global; every vertex labeled such is the same vertex. The gray square marks the vertices of $Q^{2,1}$ (note that $\beta, \delta \notin Q^{3,2}$). Note that this illustration wraps around its boundaries.

neighbor: If j is even, then α is a neighbor of a^* and β is a neighbor of b^*. If j is odd, then β is a neighbor of a^* and α is a neighbor of b^*. See Figure 3 (left) for an illustration of a horizontal connection gadget next to $S^{i,j}$ for even j.

The construction of a *vertical connection gadget next to tile set* $S^{i,j}$ is identical to the construction of a horizontal gadget, except for the following differences:

- the gadget connects tile sets $S = S^{i,j}$ and $S' = S^{i',j}$;
- the lengths of the tile paths depend on the y-coordinates; and
- if i is even, then γ is a neighbor of a^* and δ is a neighbor of b^*, and if i is odd, then δ is a neighbor of a^* and γ is a neighbor of b^*.

This concludes the construction. See Figure 3 (right) for an overview.

Let J be the set of all hidden vertices and let J^* be the set of all exposed vertices. We now show that this construction has the desired properties for showing W[1]-hardness with respect to solution size, feedback vertex number and path-width, combined.

▶ **Observation 1.** *The constructed graph* G *has* $\mathrm{pw}(G) \leq 16k^2 + 2$ *and* $\mathrm{fvn}(G) \leq 16k^2$.

Proof. The graph $G' = G - (J \cup J^*)$ consists of paths of length one and subdivisions of stars. Clearly, $\mathrm{fvn}(G') = 0$, and since removing the center vertex of a subdivision of a star yields disjoint paths, $\mathrm{pw}(G') = 2$. Adding a vertex to a graph increases each of the two parameters by at most one. Now, as $|J \cup J^*| = 16k^2$, the claim follows. ◀

Correctness. Let us first point out that the central challenge is to cover all hidden vertices J, as every other vertex is covered by the four degree-one vertices in Ξ'.

▶ **Observation 2** ($\star$[2])**.** $I[\Xi'] = V(G) \setminus J$.

[2] Results marked with ($\star$) are deferred to the full version.

Then the forward direction becomes straightforward: Our geodetic set V' consists of Ξ' and, for every tile in the solution of instance I, the corresponding tile vertex. It is easy to see that for every (copy of a) connection gadget, there are two shortest paths between the chosen tile vertices of any two adjacent tiles, each covering one of the two hidden vertices in the connection gadget. Compare with Figure 3 (hidden vertices are gray).

The backward direction is more involved. We show in two steps that every solution of our constructed instance consists of Ξ' and exactly one tile vertex of each tile set. For this we make use of two properties of our construction. First, if two vertices are sufficiently far apart, then there is a shortest path via some global vertex that connects them.

▶ **Lemma 3 (⋆).** *For any two vertices $u, v \in V(G)$ there is a u–v-path of length at most $36m+6$ that visits some global vertex.*

With Lemma 3 at hand, it is easy to derive from Figure 3 (left) the following observation, which is also the reason why the vertices in J are called *hidden*.

▶ **Observation 4.** *Let $u, v \in V(G) \setminus (\Xi \cup \Xi')$. If a shortest u–v-path visits a global vertex, then none of its inner vertices is a hidden vertex.*

We introduce some additional notation. The *square* $Q^{i,j}$ of tile set $S^{i,j}$ is the vertex set consisting of the tile vertices $S^{i,j}$, the paths between tile vertices and connector vertices towards $S^{i,j}$, and all hidden vertices and exposed vertices that are in the connection gadgets next to $S^{i,j}$. See Figure 3 (right) for an illustration of a square. Note that the squares are pairwise disjoint. We say that two squares are adjacent if they contain vertices of the same connection gadget. The *adjacency* $\mathrm{Adj}(Q^{i,j})$ of a square $Q^{i,j}$ is the union of squares adjacent to $Q^{i,j}$. The *closed adjacency* of a square $Q^{i,j}$ is the vertex set $\mathrm{Adj}[Q^{i,j}] = \mathrm{Adj}(Q^{i,j}) \cup Q^{i,j}$.

We show that any solution of (G, k') contains exactly one vertex per square.

▶ **Lemma 5.** *A geodetic set $V' \subseteq V(G)$ of size at most k' consists of the four vertices in Ξ', and exactly one vertex in each square $Q^{i,j}$, for each $i, j \in [k]$.*

Proof sketch. Recall that $k' = k^2 + 4$. The four vertices in Ξ' are the only vertices of degree one and are part of every geodetic set. Further we may assume that $V' \cap \Xi = \emptyset$ as $I[V'] = I[V' \setminus \Xi]$. So V' consists of the four vertices in Ξ' and a set of at most k^2 vertices within the squares, denoted by W.

For contradiction, assume that there are $q > 0$ squares $Q_1, \ldots, Q_q$ such that $Q_p \cap W = \emptyset$ for $p \in [q]$. We call these squares *empty*, and all other squares *non-empty*. We claim that there is an empty square Q_p such that $|\mathrm{Adj}(Q_p) \cap W| \leq 8$. Let $W' \subseteq W$ be an arbitrary subset consisting of *exactly one vertex of W per non-empty square*. So $|W'| = k^2 - q$ and $|W \setminus W'| \leq q$. Clearly, for each $p \in [q]$, we have $|\mathrm{Adj}(Q_p) \cap W'| \leq 4$, thus $\sum_{p=1}^{q} |\mathrm{Adj}(Q_p) \cap W'| \leq 4q$. Since $\sum_{p=1}^{q} |\mathrm{Adj}(Q_p) \cap \{v\}| \leq 4$ for any vertex $v \in V(G)$, we also have

$$\sum_{p=1}^{q} |\mathrm{Adj}(Q_p) \cap (W \setminus W')| = \sum_{p=1}^{q} \sum_{v \in W \setminus W'} |\mathrm{Adj}(Q_p) \cap \{v\}| \leq 4q.$$

Consequently,

$$\sum_{p=1}^{q} |\mathrm{Adj}(Q_p) \cap W| = \sum_{p=1}^{q} |\mathrm{Adj}(Q_p) \cap W'| + \sum_{p=1}^{q} |\mathrm{Adj}(Q_p) \cap (W \setminus W')| \leq 4q + 4q = 8q.$$

It follows that there exists an empty square Q for which $|\mathrm{Adj}(Q) \cap W| \leq 8$.

Let $J_Q = J \cap N[Q]$ be the sixteen hidden vertices that are either in Q or adjacent to vertices of Q. The next two claims are consequences of Lemma 3 and Observation 4 (see the full version for proofs of the claims):

(1) no shortest path between a vertex outside of Q and a vertex outside of $\mathrm{Adj}[Q]$ can visit any vertex in J_Q, and
(2) W covers at most $|\mathrm{Adj}(Q) \cap W| \leq 8$ vertices of J_Q.

Since $|J_Q| = 16$, the set V' is not geodetic; so there cannot be an empty square in G. There are k^2 squares and $|W| = |V' \setminus \Xi'| \leq k^2$. So $|V' \cap Q^{i,j}| = 1$ for each $i, j \in [k]$. ◀

Using Lemma 5, we show that every solution vertex in a square must be a tile vertex.

▶ **Lemma 6.** *A geodetic set $V' \subseteq V(G)$ of size at most k' consists of the four vertices in Ξ' and exactly one vertex of $S^{i,j}$, for each $i, j \in [k]$.*

Proof. For $i, j \in [k]$, let $S = S^{i,j}$, $S' = S^{i,j'}$, $Q = Q^{i,j}$, and $Q' = Q^{i,j'}$. Without loss of generality, assume that j is even (see Figure 3 for an illustration). Let X_1 and X_2 be the two copies of the horizontal connection gadget next to tile S, let $a_1, b_1 \in V(X_1)$ and $a_2, b_2 \in V(X_2)$ be the hidden vertices, and let $a_1^*, b_1^* \in V(X_1)$ and $a_2^*, b_2^* \in V(X_2)$ be the exposed vertices. By Lemma 5, V' contains exactly one vertex u in Q and exactly one vertex v in Q'.

Consider a vertex $w \in V(G) \setminus (Q \cup Q')$. Note that any shortest u–w-path and any shortest v–w-path going through one of a_1, a_2, b_1, b_2 must use tile vertices in S and S'. It is easy to verify that due to its length, such a path must visit some global vertex, thus it cannot visit any hidden vertex (Observation 4). It follows that $\{a_1, a_2, b_1, b_2\} \subseteq I[u, v]$.

For the sake of contradiction, suppose that $u \notin S$. In particular, we assume without loss of generality that $u \in V(X_1)$. Let $u' \in S$ be the tile vertex such that u lies on the tile path between u' and a_1. Observe that $d(u, a_1) < d(u, a_2)$. Hence, no shortest u–v-path visits a_2 if $d(v, a_1) \leq d(v, a_2)$. It follows that v lies on some tile path between some tile vertex $v' \in S'$ and a_2. Since there are shortest u–v-paths visiting a_1 and a_2, we have

$$d(u, v) = (d(a_1, u') - d(u', u)) + d(a_1, v') + d(v, v') \text{ and}$$
$$d(u, v) = (d(a_2, v') - d(v, v')) + d(a_2, u') + d(u, u').$$

By construction, $d(a_1, u') = d(a_2, u') = 16m + 2x_{u'} + 1$ and $d(a_1, v') = d(a_2, v') = 16m - 2x_{v'} + 1$. Thus, we obtain $d(u, u') = d(v, v')$ and $d(u, v) = 32m + 2x_{u'} - 2x_{v'} + 2$. Note that there is a u–v-path visiting α that is of length

$$\ell = (d(u', a_1^*) - d(u, u')) + 2 + (d(a_2^*, v') - d(v', v)).$$

Since $d(a_1, u') = 16m + 2x_{u'} + 1$ and $d(v', a_1) = 16m - 2x_{v'} + 1$ (by construction), and since $\ell \geq d(u, v)$, we obtain $d(u, u') = d(v, v') \leq 1$. By the assumption that $u \notin S$, we have $d(u, u') > 0$. It follows that $d(u, u') = d(v, v') = 1$. Finally, observe that the shortest path from u to v that visits b_1 is of length

$$\ell' = d(u, b_1) + d(b_1, v) = 32m - 2x_{u'} + 2x_{v'} + 4.$$

Since $\ell' = d(u, v)$, we obtain $4x_{u'} - 4x_{v'} = 2$, so one of $x_{u'}, x_{v'}$ cannot be integer – a contradiction. ◀

Now, given Lemma 6, if there is a solution for our instance of Geodetic Set, then the tiles corresponding to the chosen tile vertices are a solution for our instance of Grid Tiling. The main theorem of the section follows:

▶ **Theorem 7** (⋆). Geodetic Set *is* W[1]*-hard with respect to the feedback vertex number, the path-width, and the solution size, combined.*

4 Fixed-Parameter Tractability for Feedback Edge Number

We now show that Geodetic Set is fixed-parameter tractable for feedback edge number. In fact, we present a fixed-parameter algorithm for the following, more general variant:

Extended Geodetic Set

Input: A graph G, a vertex set $T \subseteq V(G)$, and an integer k.
Question: Does G have a geodetic set $S \supseteq T$ of cardinality at most k?

The algorithm works in three steps: We first apply some polynomial-time data reduction rules. The graph may be arbitrarily large even after they are applied exhaustively. However, together with some branching steps, they lead to an instance in which a part of the solution vertices are fixed and can be extended to a minimum geodetic set by adding vertices on paths of degree-two vertices. We determine these vertices using an ILP formulation with $O(\text{fen}(G)^2)$ variables, showing that (Extended) Geodetic Set is fixed-parameter tractable for feedback edge number.

Although feedback edge number is considered one of the largest structural graph parameters, our algorithm is still technically involved and it has an impractical running time. This hints at the difficulty of designing efficient algorithms for Geodetic Set. We also remark that some of the techniques presented may be of independent interest. For example, the presented approach may also be useful to show fixed-parameter tractability of the closely related Metric Dimension problem[3] for feedback edge number, which was posed as an open problem by Eppstein [13] (so far, it is only known to be in XP for this parameter [14]).

This section is divided into three parts. In Section 4.1, we provide some polynomial-time data reduction rules, which allow us to bound the number of vertices with degree at least three. In Section 4.2, we guess parts of the solution. Finally, in Section 4.3, we present our ILP formulation to determine the vertices in the solution.

Throughout this section we assume without loss of generality that G is connected.

4.1 Preprocessing

In this section we present three data reduction rules and some observations on the instance obtained after their exhaustive application. We will also introduce the *feedback edge graph* $\widetilde{G}$ in this subsection, which will be used throughout the presentation of this algorithm.

Our first reduction rule deletes degree-one vertices. This reduction rule is based on the observation that a geodetic set contains every degree-one vertex.

▶ **Reduction Rule 8.** *If there is a degree-one vertex $v \in V(G)$ with $N(v) = \{u\}$, then*
- *decrease k by 1 if $u \in T$,*
- *add u to T if $u \notin T$, and*
- *delete v from $V(G)$ (and from T).*

Henceforth we assume that Reduction Rule 8 has been exhaustively applied (which can be done in linear time). Suppose that $\text{fen}(G) = 1$. Then G is a cycle, and any minimal geodetic set $S \supseteq T$ is of size at most $|T| + 3$. So Extended Geodetic Set can be solved in polynomial time when $\text{fen}(G) \leq 1$ (in fact, further analysis yields a linear-time algorithm for $\text{fen}(G) = 1$). We thus assume that $\text{fen}(G) \geq 2$.

[3] Given a graph, Metric Dimension asks for a set S of at most k vertices such that for any pair of vertices u and v, there is a vertex in S which has distinct distances to u and v.

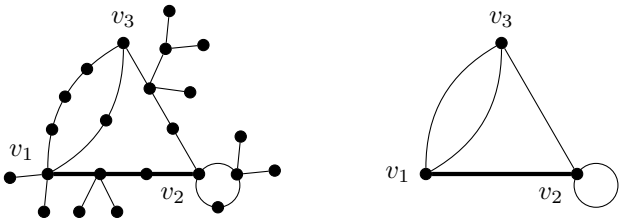


Figure 4 An illustration of an input graph G (left) and $\widetilde{G}$ after Reduction Rule 8 has been exhaustively applied (right). Observe that $\widetilde{G}$ contains no degree-one or degree-two vertex. For instance, a thick edge p in $\widetilde{G}$ (right) corresponds to a path P of length $h_p = 3$ in G(left). Moreover, we have $T_p = \{0, 1\}$ after Reduction Rule 8 has been applied exhaustively.

Now we introduce the *feedback edge graph* $\widetilde{G}$, a multigraph which is obtained from G as follows: As long as there is a degree-two vertex v with neighbors u, w, we remove v and add an edge (multiedge) uw. Using the handshake lemma, one can easily obtain the following.

▶ **Observation 9** ($\star$). *It holds that* $|V(\widetilde{G})| \leq 2\,\mathrm{fen}(G) - 2$ *and* $|E(\widetilde{G})| \leq 3\,\mathrm{fen}(G) - 3$.

Observe that each edge p in $\widetilde{G}$ is associated with a path $P = (p^0, p^1, \ldots, p^{h_p})$ in G where all of its inner vertices are of degree 2. We sometimes refer to the endpoints p^0, p^{h_p} as $p^{\leftarrow}, p^{\rightarrow}$, respectively. Moreover, let $T_p = \{i \mid p^i \in T\}$ and let $p_T^{\leftarrow} = p^{t_p^{\leftarrow}}$ and $p_T^{\rightarrow} = p^{t_p^{\rightarrow}}$, where $t_p^{\leftarrow} = \min T_p$ and $t_p^{\rightarrow} = \max T_p$. We illustrate the definitions in Figure 4.

The following reduction rule deals with self-loops in $\widetilde{G}$.

▶ **Reduction Rule 10** ($\star$). *If $v \in V(\widetilde{G})$ has a self-loop p in $\widetilde{G}$, then decrease k as follows:*

- *If $T_p = \emptyset$, then decrease k by $(h_p \bmod 2)$.*
- *If $T_p \neq \emptyset$ and $V(P) \not\subseteq I[T_p \cup \{v\}]$, then decrease k by $|T_p|$.*
- *If $T_p \neq \emptyset$ and $V(P) \subseteq I[T_p \cup \{v\}]$, then decrease k by $|T_p| - 1$.*

Moreover, add v to T and remove $V(P) \setminus \{v\}$.

The next reduction rule ensures that for every $p \in E(\widetilde{G})$ with $T_p \neq \emptyset$, there is a shortest path from an endpoint of P to the closest vertex in T_p that is contained inside P. For this we introduce the following notation. Let $\mathcal{R} = \{\leftarrow, \rightarrow\}$. For $r \in \mathcal{R}$, we denote by $\overline{r} \in \mathcal{R} \setminus \{r\}$ the opposite direction.

▶ **Reduction Rule 11** ($\star$). *Let $p \in E(\widetilde{G})$ with $T_p \neq \emptyset$, and let $r \in \mathcal{R}$. If $d_P(p_T^r, p^r) > d_P(p_T^r, p^{\overline{r}}) + d_G(p^{\overline{r}}, p^r)$, then add p' to T, where p' is between p_T^r and p^r and $d(p', p_T^r) = \lfloor (h_p + d_G(p^{\leftarrow}, p^{\rightarrow}))/2 \rfloor$.*

4.2 Guessing

Towards obtaining a geodetic set S of size at most k, we extend our current set T of vertices fixed in the solution. First we guess the set of endpoints that are in the solution. Next, using another reduction rule, we fix further vertices that are required to be in the geodetic set of our interest. These vertices possibly depend on the (previously guessed) endpoints that are in the solution. Finally, we guess how many vertices we need to add to every path P for $p \in E(\widetilde{G})$. Then, the exact positions of these vertices are determined using ILP.

Suppose that (G, T, k) is a yes-instance. We fix a solution S of minimum size that maximizes the number $|S \cap V(\widetilde{G})|$ of endpoints among all such solutions. Intuitively, our goal is to find S. To do so, we first guess the set $\widetilde{S} = S \cap V(\widetilde{G})$ of endpoints in S; there are at most $2^{|V(\widetilde{G})|} \leq 2^{2\,\mathrm{fen}(G)-2}$ possibilities by Observation 9. We extend T by adding all vertices from $\widetilde{S}$. So we will henceforth assume that $S \cap V(\widetilde{G}) = T \cap V(\widetilde{G})$. Using another reduction rule, we ensure that for every $p \in E(\widetilde{G})$, the vertices between $p_T^{\leftarrow}$ and $p_T^{\rightarrow}$ are covered.

▶ **Reduction Rule 12 (⋆).** *Let $p \in E(\widetilde{G})$. If there are $t < t' \in T_p$ such that $[t+1, t'-1] \cap T_p = \emptyset$ and $d_G(p^t, p^{t'}) < t' - t$ (equivalently, $d_G(p^{\leftarrow}, p^{\rightarrow}) + h_p < 2t' - 2t$), then add $\lfloor (t+t')/2 \rfloor$ to T.*

We will prove two lemmata required for the next guessing step and for the subsequent ILP formulation. First, we show that S contains no vertex on a path P for $p \in E(\widetilde{G})$ with $T_p \neq \emptyset$.

▶ **Lemma 13.** *Let $p \in E(\widetilde{G})$ with $T_p \neq \emptyset$. Then, $S \cap V(P) \subseteq T_p$.*

Proof. For $r \in \mathcal{R}$, suppose that S contains a vertex $p^i \in V(P) \setminus T_p$ that lies between p^r and p^r_T. Since Reduction Rule 11 is applied exhaustively, $(S \setminus \{p^i\}) \cup \{p^r\}$ is also a solution of minimum size, contradicting the maximality of $|S \cap V(\widetilde{G})|$. Thus, it remains to show that S contains no vertex that lies between $p^{\leftarrow}_T$ and $p^{\rightarrow}_T$ in P. Note that after applying Reduction Rule 12, each vertex in P between $p^{\leftarrow}_T$ and $p^{\rightarrow}_T$ are included in $I[T_p]$. Due to its minimality, S contains no vertex $p^i \in V(P) \setminus T_p$ between $p^{\leftarrow}_T$ and $p^{\rightarrow}_T$ in P. ◀

We also show that S contains at most two inner vertices of P if $T_p = \emptyset$ for $p \in E(\widetilde{G})$.

▶ **Lemma 14.** *Let $p \in E(\widetilde{G})$ with $T_p = \emptyset$. Then, $|S \cap V(P)| \leq 2$.*

Proof. If $|S \cap V(P)| = 3$, then $(S \setminus V(P)) \cup \{p^{\leftarrow}, p^{\lfloor h_p/2 \rfloor}, p^{\rightarrow}\}$ is also a minimum solution, contradicting the fact that $|S \cap V(\widetilde{G})|$ is maximized. ◀

Now we make further guesses. For each edge $p \in E(\widetilde{G})$, we guess the number $n_p \in \{0, 1, 2\}$ of inner vertices in $S \cap V(P)$. Note that there are at most $3^{|E(\widetilde{G})|} \leq 3^{3\,\mathrm{fen}(G)-3}$ possibilities by Observation 9. The next step is to determine exactly which vertices to take using ILP.

4.3 Finding a minimum geodetic set via ILP

Let $E_n = \{p \in E(\widetilde{G}) \mid T_p = \emptyset, n_p = n\}$ for $n \in \{0, 1, 2\}$ and let $E' = \{p \in E(\widetilde{G}) \mid T_p \neq \emptyset\}$. Further, let $\mathcal{E} = E_1 \cup E_2 \cup E' = E(\widetilde{G}) \setminus E_0$. Note that S contains at least one vertex in $V(P)$ for every $p \in \mathcal{E}$. For each $p \in \mathcal{E}$, we introduce two nonnegative variables $x^{\leftarrow}_p, x^{\rightarrow}_p$, and let $p^{\leftarrow}_S = p^{x^{\leftarrow}_p}$ and $p^{\rightarrow}_S = p^{h_p - x^{\rightarrow}_p}$. The intended meaning of $x^{\leftarrow}_p$, respectively $x^{\rightarrow}_p$ is that S contains $p^{\leftarrow}_S$, respectively $p^{\rightarrow}_S$. Then the geodetic set of our interest will be given by $X = T \cup \bigcup_{p \in E_1 \cup E_2} \{p^{\leftarrow}_S, p^{\rightarrow}_S\}$. For each $p \in \mathcal{E}$ we add the following constraints:

$$\begin{cases} x^{\leftarrow}_p > 0, x^{\rightarrow}_p > 0, \text{and } x^{\leftarrow}_p + x^{\rightarrow}_p \leq h_p & \text{if } p \in E_1 \cup E_2, \\ x^{\leftarrow}_p + x^{\rightarrow}_p = h_p & \text{if } p \in E_1, \\ h_p - 2x^{\leftarrow}_p - 2x^{\rightarrow}_p \leq d_G(v^{\leftarrow}_p, v^{\rightarrow}_p) & \text{if } p \in E_2, \\ x^{\leftarrow}_p = p^{\leftarrow}_T \text{ and } x^{\rightarrow}_p = h_p - p^{\rightarrow}_T & \text{if } p \in E'. \end{cases} \tag{1}$$

Let $V^{\leftarrow}_p = \{p^1, \ldots, p^{x^{\leftarrow}_p - 1}\}$ and $V^{\rightarrow}_p = \{p^{h_p - x^{\rightarrow}_i + 1}, \ldots, p^{h_p - 1}\}$ for each $p \in \mathcal{E}$. We show that constraint (1) guarantees that the vertices between $p^{\leftarrow}_S$ and $p^{\rightarrow}_S$ are covered if $p \notin E_0$.

▶ **Lemma 15 (⋆).** *If constraint (1) is fulfilled, then $Q_p = V(P) \setminus (\{p^{\leftarrow}, p^{\rightarrow}\} \cup V^{\leftarrow}_p \cup V^{\rightarrow}_p) \subseteq I[S]$ holds for each $p \in \mathcal{E}$.*

Next, we introduce constraints to determine whether there is a shortest path between p^r_S and q^s_S visiting p^r and q^s, for each $p \neq q \in E(\widetilde{G})$ and $r, s \in \mathcal{R}$ (recall that $\mathcal{R} = \{\leftarrow, \rightarrow\}$). Using binary variables $a^{r,s}_{p,q}, b^{r,s}_{p,q}, c^{r,s}_{p,q}, z^{r,s}_{p,q}$, we add the following constraints for each $p \neq q \in \mathcal{E}$ and $r, s \in \mathcal{R}$. Informally, if $z^{r,s}_{p,q} = 1$, then there exists a shortest path as described above.

$$\begin{cases} (x^r_p + d_G(p^r, q^s) + x^s_q) - (x^r_p + d_G(p^r, q^{\overline{s}}) + h_q - x^s_q) \leq N(1 - a^{r,s}_{p,q}), \\ (x^r_p + d_G(p^r, q^s) + x^s_q) - (h_p - x^r_p + d_G(p^{\overline{r}}, q^s) + x^s_q) \leq N(1 - b^{r,s}_{p,q}), \\ (x^r_p + d_G(p^r, q^s) + x^s_q) - (h_p - x^r_p + d_G(p^{\overline{r}}, q^{\overline{s}}) + h_q - x^s_q) \leq N(1 - c^{r,s}_{p,q}), \\ 3 - a^{r,s}_{p,q} - b^{r,s}_{p,q} - c^{r,s}_{p,q} \leq 3 - 3z^{r,s}_{p,q}. \end{cases} \tag{2}$$

Here N is some sufficiently large number (i.e., $N = 100 \cdot |E(G)|$ will do).

▶ **Lemma 16** (⋆). *If constraint (2) is fulfilled with $z^{r,s}_{p,q} = 1$, then $I[p^r, q^s] \subseteq I[p^r_S, q^s_S]$.*

We add a similar constraint for shortest paths between $p^{\leftarrow}_S$ and $p^{\rightarrow}_S$ for each $p \in E(\widetilde{G})$. For each $p \in \mathcal{E}$ and $r \in \mathcal{R}$ we add the constraint

$$(x^r_p + d_G(p^r, p^{\overline{r}}) + x^{\overline{r}}_p) - (h_p - x^r_p - x^{\overline{r}}_p) \leq N(1 - z^{r,\overline{r}}_{p,p}). \tag{3}$$

Here $z^{r,\overline{r}}_{p,p}$ is a binary variable. It is easy to see that if $z^{r,\overline{r}}_{p,p} = 1$, then there is a shortest path from p^r_S to $p^{\overline{r}}_S$ going through p^r and $p^{\overline{r}}$.

Now we use constraints (2) and (3) to cover the remaining vertices. First we handle the paths without any solution vertex. For each $\ell \in E_0$, we add the following constraint to guarantee that there are $p, q \in E(\widetilde{G})$ and $r, s \in \mathcal{R}$ such that $V(L) \subseteq I[p^r_S, q^s_S]$, where L is the path associated with ℓ:

$$\sum_{\substack{p,q \in \mathcal{E},\, r,s \in \mathcal{R}, (p,r) \neq (q,s) \\ d(p^r, \ell^{\leftarrow}) + h_\ell + d(\ell^{\rightarrow}, q^s) = d(p^r, q^s)}} z^{r,s}_{p,q} \geq 1. \tag{4}$$

To ensure that every vertex $v \in V(\widetilde{G}) \setminus \widetilde{S}$ is covered, we add constraint (4), where L is a path of length zero with endpoint v, that is, $h_\ell = 0$ and $\ell^{\leftarrow} = \ell^{\rightarrow} = v$.

Finally, we deal with the vertices in $V^{\leftarrow}_p$ and $V^{\rightarrow}_p$. Note that for each $p \in E(\widetilde{G})$ and $r \in \mathcal{R}$, the vertices in V^r_p are covered if

- it holds that $x^r_p \leq 1$ (that is, $V^r_p = \emptyset$), or
- there is $q \in E(\widetilde{G})$ and $s \in \mathcal{R}$ such that a shortest p^r_S–q^s_S-path visits p^r.

For each $p \in \mathcal{E}$ and $r \in \mathcal{R}$, let y^r_p be a binary variable and add the following constraint:

$$x^r_p - 1 \leq N(1 - y^r_p) \quad \text{and} \quad y^r_p + \sum_{q \in E(\widetilde{G}), s \in \mathcal{R}} z^{r,s}_{p,q} \geq 1. \tag{5}$$

It is easy to verify that if $y^r_p = 1$, then $x^r_p \leq 1$ must hold. This concludes the ILP formulation. We show that our ILP formulation finds a minimum geodetic set.

▶ **Theorem 17.** Geodetic Set *can be solved in $O^*(2^{O(\mathrm{fen}(G)^2)})$ time.*[4]

Proof. We prove that there is a geodetic set $S \supseteq T$ satisfying Lemmas 13 and 14 if and only if one of our ILP instances is a yes-instance. The forward direction is clearly correct. The correctness of the other direction is due to the following observations.

- The vertices in P for $p \in E_0$ as well as the vertices in $V(\widetilde{G}) \setminus \widetilde{S}$ are covered because of constraint (4).
- For each $p \in \mathcal{E}$, $V^{\leftarrow}_i$ and $V^{\rightarrow}_i$ are covered due to constraint (5). The remaining vertices are covered due to Lemma 16.

Note that we construct $2^{O(\mathrm{fen}(G))}$ instances of ILP. Each ILP instance uses $O(\mathrm{fen}(G)^2)$ binary variables and $O(\mathrm{fen}(G))$ variables which are not necessarily binary. To solve one ILP instance, we first try every assignment to binary variables (note that there are $2^{O(\mathrm{fen}(G)^2)}$ assignments). Then, we solve an ILP instance with $O(\mathrm{fen}(G))$ variables, which requires $O^*(\mathrm{fen}(G)^{O(\mathrm{fen}(G))})$ time [19]. This results in an algorithm whose running time is $O^*(2^{O(\mathrm{fen}(G)^2)})$. ◀

[4] The $O^*(\cdot)$ notation hides factors that are polynomial in the input size.

5 Fixed-Parameter Tractability for Clique-Width with Diameter

In this section we obtain fixed-parameter tractability results for clique-width combined with diameter, and for tree-depth. Our algorithm is based on a theorem by Courcelle et al. [10]: If a graph property π can be expressed as a formula φ in MSO_1 logic, then whether a graph G has π can be determined in $O(f(\mathrm{cw}(G)+|\varphi|)\cdot(|V(G)|+|E(G)|))$ time for some function f.

▶ **Theorem 18.** Geodetic Set *is fixed-parameter tractable with respect to* $\mathrm{cw}(G)+\mathrm{diam}(G)$.

Proof. We describe how to express Geodetic Set in MSO_1 logic. We define

$$\varphi = \exists S\ (\forall v\ [\exists u, w\ (u \in S \wedge w \in S \wedge \mathrm{Visit}(u,v,w))])\,,$$

where $\mathrm{Visit}(u,v,w)$ is true if and only if there is a shortest path u–w visiting v. It remains to construct $\mathrm{Visit}(u,v,w)$. First, let us define a formula $\mathrm{Path}(v_1,\ldots,v_i)$ which evaluates to true if and only if $(v_1,\ldots,v_i)$ is a path:

$$\mathrm{Path}(v_1,\ldots,v_\delta) = \bigwedge_{j\in[i-1]} v_jv_{j+1} \in E(G).$$

We then define $\mathrm{Dist}_i(u,w)$ which is true if and only if $d_G(u,w)=i$.

$$\begin{aligned}\mathrm{Dist}_i(u,w) =& \exists v_2,\ldots,v_{i-1}\,(\mathrm{Path}(u,v_2,\ldots,v_{i-1},w))\\ &\wedge \bigwedge_{j\in[i-1]} \nexists v_2,\ldots,v_{j-1}(\mathrm{Path}(u,v_2,\ldots,v_{j-1},w)).\end{aligned}$$

Finally, we define $\mathrm{Visit}(u,v,w)$:

$$\mathrm{Visit}(u,v,w) = \bigvee_{i\in[\mathrm{diam}(G)]} \left(\mathrm{Dist}_i(u,w) \wedge \left[\bigvee_{j\in[i-1]} \mathrm{Dist}_j(u,v)\wedge \mathrm{Dist}_{j-i}(v,w)\right]\right).$$

Note that $|\varphi| \in \mathrm{diam}(G)^{O(1)}$. Thus, fixed-parameter tractability for $\mathrm{cw}(G)+\mathrm{diam}(G)$ follows from Courcelle's theorem. ◀

Note that $\mathrm{cw}(G)\le 2$ and $\mathrm{diam}(G)\le 2$ for any cograph G. Thus, our result extends polynomial-time solvability on cographs proven by Dourado et al. [11].

We also obtain fixed-parameter tractability for tree-depth as well as for modular-width from Theorem 18. The tree-depth of a graph G can be roughly approximated by $\log h \le \mathrm{td}(G) \le h$, where h is the height of a depth-first search tree of G [21]. Hence, the length of all paths in G, specifically the diameter of G, is at most $2^{\mathrm{td}(G)}$. Moreover, $\mathrm{cw}(G) \le 3\cdot 2^{\mathrm{tw}(G)-1}$ [9] and $\mathrm{tw}(G)\le \mathrm{td}(G)-1$. Similarly, $\mathrm{cw}(G)\le \mathrm{mw}(G)$ (by definition) and $\mathrm{diam}(G)\le \max\{2,\mathrm{mw}(G)\}$ [18]. Consequently, we obtain the following.

▶ **Corollary 19.** Geodetic Set *is fixed-parameter tractable with respect to tree-depth and with respect to modular-width.*

6 Conclusion

We initiated a parameterized complexity study of Geodetic Set for parameters measuring tree-likeness. We conclude this work by suggesting some future research directions. None of the fixed-parameter algorithms presented in this work are practical. Are there more efficient fixed-parameter algorithms with respect to feedback edge number, tree-depth or

modular-width? Further, while we can quite surely exclude fixed-parameter tractability for feedback vertex number and path-width, it is still open whether Geodetic Set is in XP with any (combination) of these parameters. Recall that the related Geodetic Hull problem is in XP with respect to tree-width [17], but for Geodetic Set, even the complexity on series-parallel graphs (which have tree-width two) is unknown.

Going to related problems and parameters, it is open whether Metric Dimension is fixed-parameter tractable with respect to the feedback edge number [13]. This is especially interesting since the problem behaves similarly to Geodetic Set in terms of complexity: Metric Dimension is fixed-parameter tractable with respect to tree-depth [22] and with respect to modular-width [3], but W[1]-hard with respect to path-width [5] and W[2]-hard with respect to the solution size [16]. We are optimistic that the method presented in Section 4 can be used to answer this question positively, especially since Epstein et al. [14] showed that the number of solution vertices on a path of degree-two vertices (cf. Lemma 14) is bounded by a constant.

References

1 Júlio Araújo, Grégory Morel, Leonardo Sampaio, Ronan Pardo Soares, and Valentin Weber. Hull number: P_5-free graphs and reduction rules. *Discrete Applied Mathematics*, 210:171–175, 2016. 2

2 Mustafa Atici. Computational complexity of geodetic set. *International Journal of Computer Mathematics*, 79(5):587–591, 2002. 1

3 Rémy Belmonte, Fedor V. Fomin, Petr A. Golovach, and M. S. Ramanujan. Metric dimension of bounded tree-length graphs. *SIAM Journal on Discrete Mathematics*, 31(2):1217–1243, 2017. 13

4 Stéphane Bessy, Mitre Costa Dourado, Lucia Draque Penso, and Dieter Rautenbach. The geodetic hull number is hard for chordal graphs. *SIAM Journal on Discrete Mathematics*, 32(1):543–547, 2018. 2

5 Édouard Bonnet and Nidhi Purohit. Metric dimension parameterized by treewidth. In *14th International Symposium on Parameterized and Exact Computation (IPEC '19)*, pages 5:1–5:15, 2019. 13

6 Bostjan Bresar, Sandi Klavzar, and Aleksandra Tepeh Horvat. On the geodetic number and related metric sets in Cartesian product graphs. *Discrete Mathematics*, 308(23):5555–5561, 2008. 2

7 Letícia Rodrigues Bueno, Lucia Draque Penso, Fábio Protti, Victor R. Ramos, Dieter Rautenbach, and Uéverton S. Souza. On the hardness of finding the geodetic number of a subcubic graph. *Information Processing Letters*, 135:22–27, 2018. 1

8 Dibyayan Chakraborty, Florent Foucaud, Harmender Gahlawat, Subir Kumar Ghosh, and Bodhayan Roy. Hardness and approximation for the geodetic set problem in some graph classes. In *6th International Conference on Algorithms and Discrete Applied Mathematics (CALDAM '20)*, pages 102–115, 2020. 1

9 Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM Journal on Computing*, 34(4):825–847, 2005. 12

10 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000. 12

11 Mitre Costa Dourado, Fábio Protti, Dieter Rautenbach, and Jayme Luiz Szwarcfiter. Some remarks on the geodetic number of a graph. *Discrete Mathematics*, 310(4):832–837, 2010. 1, 2, 12

12 Mitre Costa Dourado, Lucia Draque Penso Rautenbach, and Dieter Rautenbach. On the geodetic hull number of P_k-free graphs. *Theoretical Computer Science*, 640:52–60, 2016. 2

13 David Eppstein. Metric dimension parameterized by max leaf number. *Journal of Graph Algorithms and Applications*, 19(1):313–323, 2015. 8, 13

14 Leah Epstein, Asaf Levin, and Gerhard J. Woeginger. The (weighted) metric dimension of graphs: Hard and easy cases. *Algorithmica*, 72(4):1130–1171, 2015. 8, 13

15 Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010. 3

16 Sepp Hartung and André Nichterlein. On the parameterized and approximation hardness of metric dimension. In *Proceedings of the 28th IEEE Conference on Computational Complexity (CCC '13)*, pages 266–276. IEEE, 2013. 13

17 Mamadou Moustapha Kanté, Thiago Braga Marcilon, and Rudini M. Sampaio. On the parameterized complexity of the geodesic hull number. *Theoretical Computer Science*, 791:10–27, 2019. 2, 13

18 Minki Kim, Bernard Lidický, Tomás Masarík, and Florian Pfender. Notes on complexity of packing coloring. *Information Processing Letters*, 137:6–10, 2018. 12

19 Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983. 11

20 Dániel Marx. On the optimality of planar and geometric approximation schemes. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS '07)*, pages 338–348, 2007. 2, 4

21 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012. 3, 12

22 Sanchez Villaamil. *About Treedepth and Related Notions*. PhD thesis, RWTH Aachen, 2017. 13

Parameterized Complexity of Graph Burning

Yasuaki Kobayashi
Kyoto University, Japan
kobayashi@iip.ist.i.kyoto-u.ac.jp

Yota Otachi
Nagoya University, Japan
otachi@nagoya-u.jp

Abstract

Graph Burning asks, given a graph $G = (V, E)$ and an integer k, whether there exists $(b_0, \ldots, b_{k-1}) \in V^k$ such that every vertex in G has distance at most i from some b_i. This problem is known to be NP-complete even on connected caterpillars of maximum degree 3. We study the parameterized complexity of this problem and answer all questions arose by Kare and Reddy [IWOCA 2019] about parameterized complexity of the problem. We show that the problem is W[2]-complete parameterized by k and that it does not admit a polynomial kernel parameterized by vertex cover number unless NP $\subseteq$ coNP/poly. We also show that the problem is fixed-parameter tractable parameterized by clique-width plus the maximum diameter among all connected components. This implies the fixed-parameter tractability parameterized by modular-width, by treedepth, and by distance to cographs. Although the parameterization by distance to split graphs cannot be handled with the clique-width argument, we show that this is also tractable by a reduction to a generalized problem with a smaller solution size.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Graph burning, parameterized complexity, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.21

Related Version A full version of the paper is available at `https://arxiv.org/abs/2007.08811`.

Funding *Yasuaki Kobayashi*: JSPS KAKENHI Grant Number JP20K19742.
Yota Otachi: JSPS KAKENHI Grant Numbers JP18K11168, JP18K11169, JP18H04091.

Acknowledgements The authors would like to thank Michael Lampis for bringing the Graph Burning problem to their attention.

1 Introduction

Bonato, Janssen, and Roshanbin [7, 8] introduced Graph Burning as a model of information spreading. This problem asks to burn all the vertices in a graph in the following way: we first pick a vertex and set fire to the vertex; at the beginning of each round, the fire spreads one step along edges; at the end of each round, we pick a vertex and set fire to it; the process finishes when all vertices are burned. The objective in the problem is to minimize the number of rounds (including the first one for just picking the first vertex) to burn all the vertices. The minimum number of rounds that can burn a graph G in such a process is the *burning number* of G, which is denoted by $\mathsf{b}(G)$. Given a graph G and an integer k, Graph Burning asks whether $\mathsf{b}(G) \leq k$.

In other words, the burning number of G can be defined as the minimum length k of a sequence $(b_0, \ldots, b_{k-1})$ of vertices of G such that every vertex in G has distance at most i from some b_i. We call such a sequence a *burning sequence*. Note that in this definition, b_i is the vertex we set fire in the $(k-i)$th round. Note also that we do not ask the vertices $b_0, \ldots, b_{k-1}$ to be distinct. It is also useful to introduce the generalized neighborhood of

vertices. For a vertex v of a graph G, let $N_d[v]$ be the set of vertices with distance at most d in G. For example, $N_0[v]$ contains only v, $N_1[v]$ is just the closed neighborhood of v, and $N_2[v] = \bigcup_{u \in N[v]} N[u]$. With this terminology, a sequence $(b_0, \ldots, b_{k-1})$ of vertices of $G = (V, E)$ is a burning sequence of $G = (V, E)$ if and only if $\bigcup_{0 \leq i \leq k-1} N_i[b_i] = V$.

1.1 Previous work

As a model of information spreading, it is important to know how fast the information can spread under the model in the worst case. This question can be answered by finding the maximum burning number of graphs of n vertices. The literature is rich in this direction. In the very first paper [7, 8], it is shown that $\mathsf{b}(G) \leq 2\lceil\sqrt{n}\rceil - 1$ for every connected graph G of order n and conjectured that $\mathsf{b}(G) \leq \lceil\sqrt{n}\rceil$ holds. Note that the connectivity requirement is essential here as an edgeless graph of order n needs n rounds. Some improvements of the general upper bound and studies on special cases are done [29, 33, 19, 2, 14, 34, 5, 10, 32, 25, 6, 30, 31, 39, 18], but the conjecture of $\mathsf{b}(G) \leq \lceil\sqrt{n}\rceil$ for general connected graphs remains unsettled. The current best upper bound is $\lceil(-3 + \sqrt{24n + 33})/4\rceil \approx \sqrt{1.5n}$ [29].

The computational complexity of Graph Burning has been studied intensively as well. It is shown that Graph Burning is NP-complete on trees of maximum degree 3, spiders, and linear forests [1]. The NP-completeness result is further extended to connected caterpillars of maximum degree 3 [30], which form subclasses of connected interval graphs, connected permutation graphs, and connected unit disk graphs. On the other hand, Graph Burning admits a 3-approximation algorithm for general graphs [9]; that is, given a graph G, the algorithm finds a burning sequence of G of length at most $3 \cdot \mathsf{b}(G)$ in polynomial time. Algorithms with approximation factors parameterized by path-length and tree-length are known as well [27]. Bonato and Kamali [9] asked whether the problem is APX-hard. Recently, this question has been answered in the affirmative [35].

Kare and Reddy [28] initiated the study on parameterized complexity of Graph Burning. They showed that Graph Burning on connected graphs is fixed-parameter tractable parameterized by distance to cluster graphs (disjoint unions of complete graphs) and by neighborhood diversity. The parameterized complexity with respect to the natural parameter k, the burning number, remained open. Recently, Janssen [26] has generalized the problem to directed graphs and has shown that the directed version is W[2]-complete parameterized by k even on directed acyclic graphs. It was mentioned in [26] that the original undirected version parameterized by k was still open.

For further information about the previous studies, see the recent comprehensive survey by Bonato [4].

1.2 Our results

In the literature, the input graph of Graph Burning is sometimes assumed to be connected (e.g. in [28]). In this paper, however, we do not generally assume the connectivity of input graphs. All positive results in this paper hold on possibly disconnected graphs, while all negative results hold even on connected graphs. Note that in Graph Burning, the disconnected case would be nontrivially more complex than the connected case. For example, while the burning number of a path of n vertices is $\lceil\sqrt{n}\rceil$ [7, 8], Graph Burning is NP-complete on graphs obtained as the disjoint union of paths [1].

Our study in this paper is inspired by Kare and Reddy [28] and Janssen [26]. We generalize the results in [28] and solve all open problems on parameterized complexity in [28]. In Section 2, we present our positive algorithmic results. We show that Graph Burning is fixed-parameter tractable parameterized by clique-width plus the maximum diameter among all connected components. This implies that Graph Burning is fixed-parameter

tractable parameterized by modular-width, by treedepth, and by distance to cographs. We also show that Graph Burning is fixed-parameter tractable parameterized by distance to split graphs. The complexity parameterized by distance to cographs and by distance to split graphs are explicitly asked in [28]. The fixed-parameter tractability parameterized by modular-width generalizes the one parameterized by neighborhood diversity in [28]. In Section 3, we present some negative results. We show that Graph Burning parameterized by the natural parameter k is W[2]-complete. This settles the main open problem in this line of research [28, 26]. As a byproduct, we also show that Graph Burning parameterized by vertex cover number does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. This also answers a question in [28]. See Figure 1 for a summary of the results.

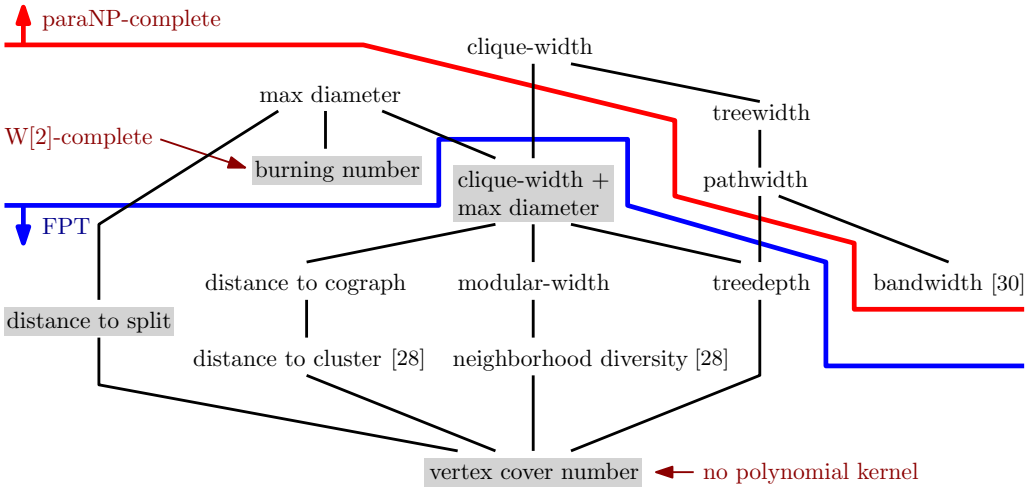


Figure 1 Graph parameters and the complexity of Graph Burning. The results on the parameters with dark background are the main results of the paper. Connections between two parameters imply the existence of a function in the one above (being in this sense more general) that lowerbounds the one below. "The maximum diameter among all connected components" is shortened as "max diameter." The paraNP-completeness parameterized by bandwidth follows from the result by Liu et al. [30] who showed that the problem is NP-complete on caterpillars of maximum degree 3, which have bandwidth at most 2.

We assume that the readers are familiar with the basic terms and concepts in the parameterized complexity theory. See some textbooks in the field (e.g., [17, 13]) for definitions. We omit the definitions of most of the graph parameters in this paper as we do not explicitly need them. We only need the definition of the *vertex cover number* of a graph: it is the minimum integer k such that their exists a set of k vertices of the graph (called a *vertex cover*) such that each edge in the graph has at least one endpoint in the set. We refer the readers to [38] for the definitions of other graph parameters and the hierarchy among them.

2 Positive results

We first observe that Graph Burning is expressible as a first order logic (FO) formula of length depending only on k.

The syntax of FO of graphs includes (i) the logical connectives $\vee$, $\wedge$, $\neg$, $\Leftrightarrow$, $\Rightarrow$, (ii) variables for vertices, (iii) the quantifiers $\forall$ and $\exists$ applicable to these variables, and (iv) the following binary relations: equality of variables, and $\mathsf{adj}(u, v)$ for two vertex variables u and v, which means that u and v are adjacent. If G models an FO formula φ with no free variables, then we write $G \models \varphi$.

The following formula $\mathrm{dist}_{\leq d}(v,w)$ is true if and only if the distance between v and w is at most d:

$$\mathrm{dist}_{\leq d}(v,w) \coloneqq \exists u_0,\ldots,u_d\ (u_0=v)\wedge(u_d=w)\wedge\left(\bigwedge_{0\leq i<d}(u_i=u_{i+1})\vee \mathsf{adj}(u_i,u_{i+1})\right).$$

Clearly, $\mathrm{dist}_{\leq d}(v,w)$ has length depending only on d. Now we define the formula φ_k such that $G\models\varphi_k$ if and only if (G,k) is a yes instance of Graph Burning as follows:

$$\varphi_k \coloneqq \exists v_0,\ldots,v_{k-1}\ \forall v \bigvee_{0\leq i\leq k-1}\mathrm{dist}_{\leq i}(v,v_i).$$

It is known that on nowhere dense graph classes, testing an FO formula ψ is fixed-parameter tractable parameterized by $|\psi|$, where $|\psi|$ is the length of ψ [23]. (See [23] for the definition of nowhere dense graph classes.) Since φ_k is an FO formula of length depending only on k, the following holds.

▶ **Observation 2.1.** *Graph Burning on nowhere dense graphs parameterized by k is fixed-parameter tractable.*

For an n-vertex graph G of clique-width at most cw and for a one-sorted monadic-second order logic (MSO_1) formula ψ, one can check whether $G\models\psi$ in time $O(f(|\psi|,\mathsf{cw})\cdot n^3)$, where f is a computable function [12, 37]. Since an FO formula is an MSO_1 formula and the length of φ_k depends only on k, we can observe the following fact.

▶ **Observation 2.2.** *Graph Burning parameterized by clique-width $+\,k$ is fixed-parameter tractable.*

We extend this observation in a nontrivial way to show the main result of this section. To this end, it is useful to generalize φ_k as follows. For nonempty $I=\{i_1,\ldots,i_{|I|}\}\subseteq\{0,\ldots,k-1\}$, let φ_I be a formula that means that there are vertices $b_{i_1},\ldots,b_{i_{|I|}}$ such that $\bigcup_{i_j\in I}N_{i_j}[b_{i_j}]=V$, which can be expressed as follows:

$$\varphi_I \coloneqq \exists v_{i_1},\ldots,v_{i_{|I|}}\ \forall v \bigvee_{1\leq j\leq |I|}\mathrm{dist}_{\leq i_j}(v,v_{i_j}).$$

Clearly, checking $G\models\varphi_I$ is still fixed-parameter tractable parameterized by clique-width $+\,k$.

▶ **Theorem 2.3.** *Graph Burning is fixed-parameter tractable parameterized by the clique-width of the input graph plus the maximum diameter among all connected components.*

Proof. Let (G,k) be an instance of Graph Burning, where G has n vertices. Let $C_1,\ldots,C_p$ be the connected components of G and $d_{\max}$ be the maximum diameter of the components. We assume that $k\geq p$ since otherwise (G,k) is a trivial no instance. By Observation 2.2, we can also assume that $d_{\max}<k$.

Observe that if a component C_q contains b_i for some $i\geq d_{\max}$, then $N_i[b_i]=V(C_q)$. Hence the problem is equivalent to finding a sequence $(b_0,\ldots,b_{d_{\max}-1})$ that burns as many connected components as possible. That is, we want to find a maximum cardinality subset $\mathcal{C}\subseteq\{C_1,\ldots,C_p\}$ and a sequence $(b_0,\ldots,b_{d_{\max}-1})$ such that $\bigcup_{0\leq i\leq d_{\max}-1}N_i[b_i]=\bigcup_{C_q\in\mathcal{C}}V(C_q)$. It holds that $p-|\mathcal{C}|\leq k-d_{\max}$ if and only if (G,k) is a yes instance.

We reduce this problem to Disjoint Sets. Given a universe U, a subset family $\mathcal{S}\subseteq 2^U$, and an integer t, Disjoint Sets asks whether there are t pairwise-disjoint subsets in $\mathcal{S}$. Let $U=\{0,1,\ldots,d_{\max}-1\}\cup\{c_1,\ldots,c_p\}$ and $\mathcal{S}=\{I\cup\{c_q\}\mid I\subseteq\{0,\ldots,d_{\max}-1\},\ C_q\models$

φ_I, $1 \leq q \leq p\}$. Clearly, picking $I \cup \{c_q\}$ into the solution for the Disjoint Set instance corresponds to burning C_q with $\{b_i \mid i \in I\}$, and vice versa. We set $t = p - k + d_{\max}$. Note that $t \leq d_{\max}$ as $k \geq p$. Using the color-coding technique, it can be shown that Disjoint Sets is solvable in time $2^{O(t \cdot \max_{S \in \mathcal{S}} |S|)}(|\mathcal{S}| + |U|)^{O(1)}$ [17, Disjoint r-Subsets] (see also [15, Bounded Rank Disjoint Sets]). In our instance, $t \cdot \max_{S \in \mathcal{S}} |S| \leq d_{\max}(d_{\max} + 1)$ holds.

For each $q \in \{1, \ldots, p\}$ and for each $I \subseteq \{0, \ldots, d_{\max} - 1\}$, we can test whether $C_q \models \varphi_I$ in time $O(f(d_{\max} + \mathsf{cw}) \cdot n^3)$ for some computable function f, where cw is the clique-width of G, because $|\varphi_I|$ depends only on $d_{\max}$. Thus, $\mathcal{S}$ can be constructed in time $O(f(d_{\max} + \mathsf{cw}) \cdot n^3 \cdot p \cdot 2^{d_{\max}})$. Since $|\mathcal{S}| \leq 2^{d_{\max}} \cdot p$, the last step of solving Disjoint Sets can be done in time $2^{O(d_{\max}^2)}(2^{d_{\max}} \cdot p + (d_{\max} + p))^{O(1)}$. Since $p \leq n$, the total running time is $g(d_{\max} + \mathsf{cw}) \cdot n^{O(1)}$ for some computable function g. This completes the proof. ◀

The definition of modular-width implies that every connected component of a graph of modular-width at most w has both diameter and clique-width at most w [22]. It is known that every connected component of a graph of treedepth at most d has diameter at most 2^d [36] and its clique-width is bounded by a function of treedepth (or even smaller treewidth) [11]. Therefore, Theorem 2.3 implies the fixed-parameter tractability with respect to these parameters.

▶ **Corollary 2.4.** *Graph Burning is fixed-parameter tractable parameterized by modular-width.*

▶ **Corollary 2.5.** *Graph Burning is fixed-parameter tractable parameterized by treedepth.*

For a graph class $\mathcal{C}$ and a graph G, the *distance* from G to $\mathcal{C}$ is defined as the minimum integer k such that by removing at most k vertices from G, one can obtain a member of $\mathcal{C}$.

Let $\mathcal{C}$ be a graph class with constants c and d such that each graph in $\mathcal{C}$ has clique-width at most c and each connected component of each member of $\mathcal{C}$ has diameter at most d. Observe that a graph of distance at most k to $\mathcal{C}$ has clique-width at most $c \cdot 2^k$ since after a removal of a single vertex, the clique-width remains at least half of the original clique-width [24]. Observe also that each connected component of a graph of distance at most k to $\mathcal{C}$ has diameter at most $k+d$. Thus, by Theorem 2.3, Graph Burning is fixed-parameter tractable parameterized by distance to $\mathcal{C}$. This observation can be applied immediately to cographs that are known to be the P_4-free graphs and the graphs of clique-width at most 2.

▶ **Corollary 2.6.** *Graph Burning is fixed-parameter tractable parameterized by distance to cographs.*

Now we consider the distance to split graphs. A graph is a *split graph* if its vertex set can be partitioned into a clique and an independent set. For a graph $G = (V, E)$, which is not necessarily a split graph, a subset $S \subseteq V$ is a *split-deletion set* if $G - S$ is a split graph. Then the distance to split graphs from G is equal to the minimum size of a split-deletion set. It is known that the split graphs are exactly the $(2K_2, C_4, C_5)$-free graphs [20]. This characterization implies that when designing an algorithm parameterized by distance d to split graphs, we can assume that a split-deletion set of minimum size is given since a standard bounded-search tree algorithm finds such a set in time $5^d \cdot n^{O(1)}$, where the number 5 is the maximum order of the forbidden induced subgraphs.

▶ **Theorem 2.7.** *Graph Burning is fixed-parameter tractable parameterized by distance to split graphs.*

Proof. Let (G, k) be an instance of Graph Burning and S be a minimum split-deletion set of $G = (V, E)$. We denote $|S|$ by s. Let (K, I) be a partition of $V - S$, where K is a clique and I is an independent set. Such a partition can be found in linear time by greedily adding a vertex of minimum degree into I. We further partition I into I_K, I_S, and $I_\emptyset$ in such a way that $I_\emptyset$ is the set of degree-0 vertices in G, I_S is the set of vertices in $I \setminus I_\emptyset$ that have neighbors only in S, and $I_K = I \setminus (I_\emptyset \cup I_S)$. Note that each vertex in I_S has at least one neighbor in S and each vertex in I_K has at least one neighbor in K (and possibly some neighbors in S).

We first reduce the number of vertices in I_S. For a nonempty subset $S' \subseteq S$, let $J_{S'} \subseteq I_S$ be the set of vertices whose neighborhood is exactly S'. Since the pairwise distance between vertices in $J_{S'}$ is 2, a burning sequence does not pick four or more vertices in $J_{S'}$. Thus, we can remove all but three vertices in $J_{S'}$ and obtain an equivalent instance. We apply this reduction to all subsets $S' \subseteq S$ and denote the reduced subset of I_S by I_S^*. Note that $|I_S^*| < 3 \cdot 2^{|S|}$. If $k \geq 3 + |S| + |I_S^*| + |I_\emptyset| = 3 + s + 3 \cdot 2^s + |I_\emptyset|$, then (G, k) is a yes instance: we can take all the vertices in $S \cup I_S^* \cup I_\emptyset$ and three vertices in $K \cup I_K$ and then arbitrarily order them to construct a burning sequence of G. Hence, in the following, we assume that $k < 3 + s + 3 \cdot 2^s + |I_\emptyset|$.

We next remove all vertices in $I_\emptyset$ and obtain an equivalence instance of a slightly generalized problem. Observe that if (G, k) is a yes instance, then $k \geq |I_\emptyset|$ and there is a burning sequence $(b_0, \ldots, b_{k-1})$ of G such that the set of first $|I_\emptyset|$ vertices $\{b_0, \ldots, b_{|I_\emptyset|}\}$ is $I_\emptyset$: we need to take every isolated vertex into a burning sequence, but even b_0 is good enough to burn an isolated component. In the following, we only consider burning sequences with this restriction. Now the problem is reduced to the one for finding a sequence $(b_{|I_\emptyset|}, b_{|I_\emptyset|+1}, \ldots, b_{k-1})$ of vertices in $V \setminus I_\emptyset$ such that $\bigcup_{|I_\emptyset| \leq i \leq k-1} N_i[b_i] = V \setminus I_\emptyset$. We denote by $(G', k, |I_\emptyset|)$ the obtained instance of the new problem, where $G' = G[K \cup I_K \cup S \cup I_S^*]$.

To solve the reduced problem, we first guess which vertices in $S \cup I_S^*$ appear in $(b_{|I_\emptyset|}, b_{|I_\emptyset|+1}, \ldots, b_{k-1})$ and where they are placed in the sequence. The number of candidates of such a guess depends only on s as $|S \cup I_S^*|^{k-|I_\emptyset|} < (s + 3 \cdot 2^s)^{s+3\cdot 2^s+3}$. The vacant slots of $(b_{|I_\emptyset|}, b_{|I_\emptyset|+1}, \ldots, b_{k-1})$ after the guess tell us which b_i belongs to $K \cup I_K$. If there are at most three vertices in $K \cup I_K$ appear in $(b_{|I_\emptyset|}, b_{|I_\emptyset|+1}, \ldots, b_{k-1})$ then we try all $O(n^3)$ combinations to complete the sequence. Otherwise, we guess from $O(n)$ candidates the vertex in $K \cup I_K$ that appears in $(b_{|I_\emptyset|}, b_{|I_\emptyset|+1}, \ldots, b_{k-1})$ and has the largest index. Since the index of the guessed vertex in $(b_{|I_\emptyset|}, b_{|I_\emptyset|+1}, \ldots, b_{k-1})$ is at least 3 and $K \cup I_K$ induces a connected split graph, which has diameter at most 3, the guessed vertex in $K \cup I_K$ burns all vertices in $K \cup I_K$.

Finally we fill the positions in $(b_{|I_\emptyset|}, b_{|I_\emptyset|+1}, \ldots, b_{k-1})$ that still remain vacant. Let $X \subseteq \{|I_\emptyset|, \ldots, k-1\}$ be the set of indices i for which no vertex is guessed as b_i so far, and let $\overline{X} = \{|I_\emptyset|, \ldots, k-1\} \setminus X$. Let $U = V(G') \setminus (\bigcup_{i \in \overline{X}} N_i[b_i])$. Note that $U \subseteq S \cup I_S^*$. Our task is to find $\{b_i \mid i \in X\} \subseteq K \cup I_K$ such that $U \subseteq \bigcup_{i \in X} N_i[b_i]$. This task can be seen as an instance $(U', \mathcal{S}, p)$ of Set Cover, where $U' = U \cup X$, $\mathcal{S} = \{(N_i[v] \cap U) \cup \{i\} \mid v \in K \cup I_K, i \in X\}$, and $p = |X| < k - |I_\emptyset| \leq 3 + s + 3 \cdot 2^s$. Since Set Cover parameterized by $|U'|$ is fixed-parameter tractable [21, Lemma 2] and $|U'| \leq s + 3 \cdot 2^s + p$, the theorem follows. ◀

3 Negative results

This section is devoted to the proofs of the following theorems.

▶ **Theorem 3.1.** *Graph Burning is* W[2]*-complete parameterized by* k*.*

▶ **Theorem 3.2.** *Graph Burning does not admit a polynomial kernel parameterized by vertex cover number unless* NP $\subseteq$ coNP/poly*.*

We present a reduction from Set Cover to Graph Burning that proves both Theorems 3.1 and 3.2. Given a set $U = \{u_1, \ldots, u_n\}$, a family of nonempty subsets $\mathcal{S} = \{S_1, \ldots, S_m\} \subseteq 2^U \setminus \{\emptyset\}$, and a positive integer s, Set Cover asks whether there exists a subfamily $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'| \leq s$ and $\bigcup_{S \in \mathcal{S}'} S = U$.

Let $(U = \{u_1, \ldots, u_n\}, \mathcal{S} = \{S_1, \ldots, S_m\}, s)$ be an instance of Set Cover. We construct an equivalent instance $(G, k = s+2)$ of Graph Burning. (See Figure 2.) We first construct $s = k-2$ isomorphic graphs $G_2, \ldots, G_{k-1}$ as follows. For each $i \in \{2, 3, \ldots, k-1\}$, the vertex set of G_i is $U_i \cup V_i$, where $U_i = \{u_1^{(i)}, \ldots, u_n^{(i)}\}$ is a clique and $V_i = \{v_1^{(i)}, \ldots, v_m^{(i)}\}$ is an independent set. In G_i, $u_p^{(i)}$ and $v_q^{(i)}$ are adjacent if and only if $u_p \in S_q$. From each G_i, we construct H_i by adding $i+2$ copies of a path of i vertices and all possible edges between each vertex in V_i and one of the degree-1 vertices in each path. We then take the disjoint union of $H_2, H_3, \ldots, H_{k-1}$ and add U as a clique. For each $i \in \{2, 3, \ldots, k-1\}$ and $j \in \{1, 2, \ldots, m\}$, we connect $u_j^{(i)}$ and u_j with a path of length $i-1$ with $i-2$ new inner vertices. Finally, we attach a vertex w to a vertex in U, and a path (x, y, z) to the same vertex. We set $V_0 = \{w\}$ and $V_1 = \{x, y, z\}$. We denote the constructed graph by G.

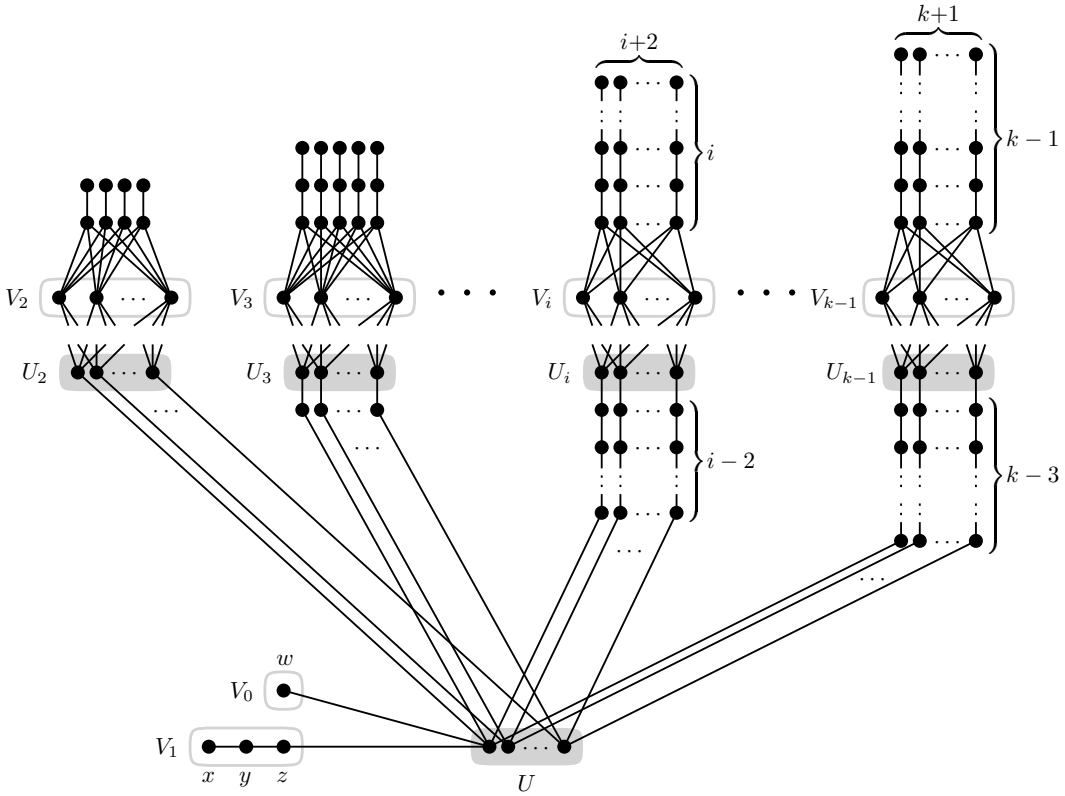


Figure 2 The reduction from Set Cover to Graph Burning. The edges in the cliques $U_2, \ldots, U_{k-1}$, and U are omitted.

▶ **Lemma 3.3.** *$(U, \mathcal{S}, s)$ is a yes instance of Set Cover if and only if (G, k) is a yes instance of Graph Burning.*

Proof. ($\Longrightarrow$) Assume that $(U, \mathcal{S}, s)$ is a yes instance of Set Cover and $\mathcal{S}' \subseteq \mathcal{S}$ is a certificate; that is, $|\mathcal{S}'| \leq s$ and $\bigcup_{S \in \mathcal{S}'} S = U$. We assume without loss of generality that $|\mathcal{S}'| = s$ and $\mathcal{S}' = \{S_2, S_3, \ldots, S_{s+1=k-1}\}$. We set $b_0 = w$, $b_1 = y$, and $b_i = v_i^{(i)}$ for $2 \leq i \leq k-1$. We show that $(b_0, \ldots, b_{k-1})$ is a burning sequence of G.

Clearly, $N_0[b_0] = V_0$ and $N_1[b_1] = V_1$. For $2 \leq i \leq k-1$, observe that $N_i[b_i] = N_i[v_i^{(i)}]$ includes all the vertices of H_i: the farthest vertices in i-vertex paths have distance exactly i from $v_i^{(i)}$; $\mathrm{dist}(v_i^{(i)}, v_j^{(i)}) = 2$ for each $j \neq i$ as $v_j^{(i)}$ and $v_i^{(i)}$ share a neighbor (an endpoint of a path of i vertices); $\mathrm{dist}(v_i^{(i)}, u_j^{(i)}) \leq 2$ for every j since $v_i^{(i)}$ has at least one neighbor in the clique U_i as $\emptyset \notin \mathcal{S}$. Moreover, $N_i[v_i^{(i)}]$ includes all inner vertices of the paths from U_i to U as $\mathrm{dist}(v_i^{(i)}, u_j^{(i)}) \leq 2$ for every j. Finally, $u_j \in N_i[v_i^{(i)}]$ if and only if $u_j \in S_i$: if $u_j \in S_i$, then $v_i^{(i)}$ and $u_j^{(i)}$ are adjacent, and thus $\mathrm{dist}(v_i^{(i)}, u_j) \leq 1 + \mathrm{dist}(u_j^{(i)}, u_j) = i$; otherwise, $v_i^{(i)}$ and $u_j^{(i)}$ are not adjacent and thus $\mathrm{dist}(v_i^{(i)}, u_j) \geq \min\{2 + \mathrm{dist}(u_j^{(i)}, u_j), 1 + \mathrm{dist}(u_{h \neq j}^{(i)}, u_j)\} > i$. This implies that $U \subseteq \bigcup_{2 \leq i \leq k-1} N_i[v_i^{(i)}]$ since $\bigcup_{S \in \mathcal{S}'} S = U$.

($\Longleftarrow$) Assume that (G, k) is a yes instance of Graph Burning and $(b_0, \ldots, b_{k-1})$ is a burning sequence of G.

We first show that $b_i \in V_i$ for all $0 \leq i \leq k-1$. Let $i \in \{2, \ldots, k-1\}$. Assume that we already know that $b_j \in V_j$ for $i+1 \leq j \leq k-1$. Since there are $i+2$ paths attached to V_i, at least one of them, say P, has no vertex in the remaining vertices $b_0, \ldots, b_i$. The degree-1 vertex in P has distance exactly i from every vertex in V_i and distance at least $i+1$ from every vertex not in $V(P) \cup V_i$. Hence, $b_i \in V_i$. Now we know that $b_i \in V_i$ for $2 \leq i \leq k-1$. Let $u_j \in U$ be the vertex where V_0 and V_1 are attached to. For $2 \leq i \leq k-1$, we have $\mathrm{dist}(b_i, u_j) \geq i$, and thus $N_i[b_i]$ contains no vertex in $V_0 \cup V_1$. Since $N_0[b_0]$ will cover only one vertex, $b_1 = y$ and $b_0 = w$ hold.

For $2 \leq i \leq k-1$, let $b_i = v_{h_i}^{(i)}$. Since $N_0[b_0] = \{w\}$ and $N_1[b_1] = \{x, y, z\}$, we have $U \subseteq \bigcup_{2 \leq i \leq k-1} N_i[v_{h_i}^{(i)}]$. As we saw in the only-if case, $u_j \in N_i[v_{h_i}^{(i)}]$ if and only if $u_j \in S_{h_i}$ for $1 \leq j \leq n$. This implies that $N_i[v_{h_i}^{(i)}] \cap U = S_{h_i}$, and thus $U = \bigcup_{2 \leq i \leq k-1} S_{h_i}$. Therefore, the subfamily $\{S_{h_2}, S_{h_3}, \ldots, S_{h_{k-1}}\} \subseteq \mathcal{S}$ of at most $k-2 = s$ subsets shows that $(U, \mathcal{S}, s)$ is a yes-instance of Set Cover. ◀

Proof of Theorem 3.1. By Lemma 3.3, the construction of (G, k) from $(U, \mathcal{S}, s)$ described above is a parameterized reduction from Set Cover parameterized by s to Graph Burning parameterized by $k = s + 2$. Since Set Cover is W[2]-complete parameterized by s [16], the W[2]-hardness follows.

The membership to W[2] can be shown by the following reduction to Set Cover. Let $(G = (V, E), k)$ be an instance of Graph Burning. We set $s = k$, $U = V \cup \{0, 1, \ldots, k-1\}$, and $\mathcal{S} = \{N_i[v] \cup \{i\} \mid v \in V, 0 \leq i \leq k-1\}$. This is just an undirected version of the proof by Janssen [26], who showed the membership to W[2] for Graph Burning on directed graphs, and the correctness can be shown in the same way. ◀

Proof of Theorem 3.2. The graph G constructed above has an independent set $\bigcup_{2 \leq i \leq k-1} V_i$. The vertices not belonging to this independent set form a vertex cover of size $4 + (k-1)|U| + \sum_{2 \leq i \leq k-1} (i+2)(2i-2)$, which is a polynomial in $k = s + 2$ and $|U|$. By Lemma 3.3, the construction of (G, k) from $(U, \mathcal{S}, s)$ described above is a polynomial parameter transformation [3] from Set Cover parameterized by $|U| + s$ to Graph Burning parameterized by vertex cover number. Since Set Cover parameterized by $|U| + s$ does not admit polynomial kernels unless NP $\subseteq$ coNP/poly [15], the theorem holds. ◀

The reduction above also shows the W[2]-hardness parameterized by diameter since the diameter of a connected graph is smaller than the square of its burning number [7, 8]. We further observe that the graph G in the reduction is $P_{(4k-3)}$-free. That is, G does not contain a path of $4k-3$ vertices as an induced subgraph. Let P be an induced path in G. For $i \in \{2, \ldots, k-1\}$, let H_i' be the graph consists of H_i and the paths from U_i to U. Since U

is a clique, there are at most two indices i such that P intersects $H'_i - U$. We can see that $|V(P) \cap V(H'_i)| \leq 2i + 1$ for each i, and thus $|V(P)| \leq 2(k-1) + 1 + 2(k-2) + 1 = 4k - 4$. This implies the following W[2]-hardness.

▶ **Corollary 3.4.** *Graph Burning on P_q-free graphs is* W[2]*-hard parameterized by q.*

References

1 Stéphane Bessy, Anthony Bonato, Jeannette C. M. Janssen, Dieter Rautenbach, and Elham Roshanbin. Burning a graph is hard. *Discret. Appl. Math.*, 232:73–87, 2017. doi:10.1016/j.dam.2017.07.016.

2 Stéphane Bessy, Anthony Bonato, Jeannette C. M. Janssen, Dieter Rautenbach, and Elham Roshanbin. Bounds on the burning number. *Discret. Appl. Math.*, 235:16–22, 2018. doi:10.1016/j.dam.2017.09.012.

3 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011. doi:10.1016/j.tcs.2011.04.039.

4 Anthony Bonato. A survey of graph burning. *CoRR*, abs/2009.10642, 2020. arXiv:2009.10642.

5 Anthony Bonato, Sean English, Bill Kay, and Daniel Moghbel. Improved bounds for burning fence graphs. *CoRR*, abs/1911.01342, 2019. arXiv:1911.01342.

6 Anthony Bonato, Karen Gunderson, and Amy Shaw. Burning the plane. *Graphs Comb.*, 36:1311–1335, 2020. doi:10.1007/s00373-020-02182-9.

7 Anthony Bonato, Jeannette C. M. Janssen, and Elham Roshanbin. Burning a graph as a model of social contagion. In *WAW 2014*, volume 8882 of *Lecture Notes in Computer Science*, pages 13–22, 2014. doi:10.1007/978-3-319-13123-8_2.

8 Anthony Bonato, Jeannette C. M. Janssen, and Elham Roshanbin. How to burn a graph. *Internet Math.*, 12(1-2):85–100, 2016. doi:10.1080/15427951.2015.1103339.

9 Anthony Bonato and Shahin Kamali. Approximation algorithms for graph burning. In *TAMC 2019*, volume 11436 of *Lecture Notes in Computer Science*, pages 74–92, 2019. doi:10.1007/978-3-030-14812-6_6.

10 Anthony Bonato and Thomas Lidbetter. Bounds on the burning numbers of spiders and path-forests. *Theor. Comput. Sci.*, 794:12–19, 2019. doi:10.1016/j.tcs.2018.05.035.

11 Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005. doi:10.1137/S0097539701385351.

12 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

13 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

14 Sandip Das, Subhadeep Ranjan Dev, Arpan Sadhukhan, Uma Kant Sahoo, and Sagnik Sen. Burning spiders. In *CALDAM 2018*, volume 10743 of *Lecture Notes in Computer Science*, pages 155–163, 2018. doi:10.1007/978-3-319-74180-2_13.

15 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and IDs. *ACM Trans. Algorithms*, 11(2):13:1–13:20, 2014. doi:10.1145/2650261.

16 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.

17 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

18 Zahra Rezai Farokh, Maryam Tahmasbi, Zahra Haj Rajab Ali Tehrani, and Yousof Buali. New heuristics for burning graphs. *CoRR*, abs/2003.09314, 2020. arXiv:2003.09314.

19 Shannon L. Fitzpatrick and Leif Wilm. Burning circulant graphs. *CoRR*, abs/1706.03106, 2017. arXiv:1706.03106.

20 Stephane Foldes and Peter L. Hammer. Split graphs. In *the Eighth Southeastern Conference on Combinatorics, Graph Theory and Computing*, volume 19 of *Congressus Numerantium*, pages 311–315, 1977.

21 Fedor V. Fomin, Dieter Kratsch, and Gerhard J. Woeginger. Exact (exponential) algorithms for the dominating set problem. In *WG 2004*, volume 3353 of *Lecture Notes in Computer Science*, pages 245–256, 2004. doi:10.1007/978-3-540-30559-0_21.

22 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In *IPEC 2013*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176, 2013. doi:10.1007/978-3-319-03898-8_15.

23 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

24 Frank Gurski. The behavior of clique-width under graph operations and graph transformations. *Theory Comput. Syst.*, 60(2):346–376, 2017. doi:10.1007/s00224-016-9685-1.

25 Michaela Hiller, Eberhard Triesch, and Arie M. C. A. Koster. On the burning number of p-caterpillars. *CoRR*, abs/1912.10897, 2019. arXiv:1912.10897.

26 Remie Janssen. The burning number of directed graphs: Bounds and computational complexity. *CoRR*, abs/2001.03381, 2020. arXiv:2001.03381.

27 Shahin Kamali, Avery Miller, and Kenny Zhang. Burning two worlds. In *SOFSEM 2020*, volume 12011, pages 113–124. Springer, 2020. doi:10.1007/978-3-030-38919-2_10.

28 Anjeneya Swami Kare and I. Vinod Reddy. Parameterized algorithms for graph burning problem. In *IWOCA 2019*, volume 11638 of *Lecture Notes in Computer Science*, pages 304–314, 2019. doi:10.1007/978-3-030-25005-8_25.

29 Max R. Land and Linyuan Lu. An upper bound on the burning number of graphs. In *WAW 2016*, volume 10088 of *Lecture Notes in Computer Science*, pages 1–8, 2016. doi:10.1007/978-3-319-49787-7_1.

30 Huiqing Liu, Xuejiao Hu, and Xiaolan Hu. Burning number of caterpillars. *Discret. Appl. Math.*, 284:332–340, 2020. doi:10.1016/j.dam.2020.03.062.

31 Huiqing Liu, Xuejiao Hu, and Xiaolan Hu. Burning numbers of path forests and spiders. *Bull. Malays. Math. Sci. Soc.*, 2020. to appear. doi:10.1007/s40840-020-00969-w.

32 Huiqing Liu, Ruiting Zhang, and Xiaolan Hu. Burning number of theta graphs. *Appl. Math. Comput.*, 361:246–257, 2019. doi:10.1016/j.amc.2019.05.031.

33 Dieter Mitsche, Pawel Pralat, and Elham Roshanbin. Burning graphs: A probabilistic perspective. *Graphs Comb.*, 33(2):449–471, 2017. doi:10.1007/s00373-017-1768-5.

34 Dieter Mitsche, Pawel Pralat, and Elham Roshanbin. Burning number of graph products. *Theor. Comput. Sci.*, 746:124–135, 2018. doi:10.1016/j.tcs.2018.06.036.

35 Debajyoti Mondal, N. Parthiban, V. Kavitha, and Indra Rajasingh. APX-hardness and approximation for the k-burning number problem. *CoRR*, abs/2006.14733, 2020. arXiv:2006.14733.

36 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*. Algorithms and combinatorics. Springer, 2012. doi:10.1007/978-3-642-27875-4.

37 Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):10:1–10:20, 2008. doi:10.1145/1435375.1435385.

38 Manuel Sorge and Mathias Weller. The graph parameter hierarchy, 2019. URL: https://manyu.pro/assets/parameter-hierarchy.pdf.

39 Ta Sheng Tan and Wen Chean Teh. Graph burning: Tight bounds on the burning numbers of path forests and spiders. *Appl. Math. Comput.*, 385:125447, 2020. doi:10.1016/j.amc.2020.125447.

Finding Optimal Triangulations Parameterized by Edge Clique Cover

Tuukka Korhonen
Department of Computer Science, University of Helsinki, Finland
https://tuukkakorhonen.com
tuukka.m.korhonen@helsinki.fi

Abstract

Many graph problems can be formulated as a task of finding an optimal triangulation of a given graph with respect to some notion of optimality. In this paper we give algorithms to such problems parameterized by the size of a minimum edge clique cover ($\mathtt{cc}$) of the graph. The parameter $\mathtt{cc}$ is both natural and well-motivated in many problems on this setting. For example, in the perfect phylogeny problem $\mathtt{cc}$ is at most the number of taxa, in fractional hypertreewidth $\mathtt{cc}$ is at most the number of hyperedges, and in treewidth of Bayesian networks $\mathtt{cc}$ is at most the number of non-root nodes of the Bayesian network.

Our results are based on the framework of potential maximal cliques. We show that the number of minimal separators of graphs is at most $2^{\mathtt{cc}}$ and the number of potential maximal cliques is at most $3^{\mathtt{cc}}$. Furthermore, these objects can be listed in times $O^*(2^{\mathtt{cc}})$ and $O^*(3^{\mathtt{cc}})$, respectively, even when no edge clique cover is given as input; the $O^*(\cdot)$ notation omits factors polynomial in the input size. Using these enumeration algorithms we obtain $O^*(3^{\mathtt{cc}})$ time algorithms for problems in the potential maximal clique framework, including for example treewidth, minimum fill-in, and feedback vertex set. We also obtain an $O^*(3^m)$ time algorithm for fractional hypertreewidth, where m is the number of hyperedges. In the case when an edge clique cover of size $\mathtt{cc}'$ is given as an input we further improve the time complexity to $O^*(2^{\mathtt{cc}'})$ for treewidth, minimum fill-in, and chordal sandwich. This implies an $O^*(2^n)$ time algorithm for perfect phylogeny, where n is the number of taxa. We also give polynomial space algorithms with time complexities $O^*(9^{\mathtt{cc}'})$ and $O^*(9^{\mathtt{cc}+O(\log^2 \mathtt{cc})})$ for problems in this framework.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Treewidth, Minimum fill-in, Perfect phylogeny, Fractional hypertreewidth, Potential maximal cliques, Edge clique cover

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.22

Related Version A full version of the paper is available at https://arxiv.org/abs/1912.10989.

Funding This work has been financially supported by Academy of Finland (grant 322869).

Acknowledgements I wish to thank Matti Järvisalo, Mikko Koivisto, Andreas Niskanen, and Juha Harviainen for helpful comments.

1 Introduction

In this paper, we give algorithms to problems that can be formulated as a task of finding an optimal triangulation of a given graph with respect to some notion of optimality. A triangulation of a graph G is a chordal graph H with $E(G) \subseteq E(H)$. For example, computing the graph parameter treewidth corresponds to finding a triangulation with the minimum possible size of a maximum clique, and minimum fill-in corresponds to finding a triangulation with the least number of edges. In particular, we consider optimal triangulation problems parameterized by the size of a minimum edge clique cover of the input graph, denoted by $\mathtt{cc}$. An edge clique cover of a graph is a set of cliques of the graph that covers all edges of the graph. Our algorithms are based on the framework of potential maximal cliques [5, 14].

1.1 Motivation

While in general the parameter `cc` could be considered non-standard, it has natural interpretations in at least three settings on which algorithms for finding optimal triangulations are applied: hypergraph parameters, phylogenetics, and probabilistic inference. The reason that `cc` is a natural choice in these settings is that the input graph is constructed as a union of cliques, with the goal of each clique W representing a constraint of type "the triangulation must contain a maximal clique Ω with $W \subseteq \Omega$". Maximal cliques of a triangulation in turn correspond to bags of a tree decomposition. Next we discuss the three settings in more detail.

The hypergraph parameter fractional hypertreewidth is a central structural parameter of constraint satisfaction problems (CSPs) [18]. The computation of fractional hypertreewidth can be formulated as a task of finding a triangulation H of the primal graph of the hypergraph, minimizing the quantity $\max_{\Omega \in \mathrm{MC}(H)} \mathrm{FCOV}(\Omega)$, where $\mathrm{MC}(H)$ denotes the set of maximal cliques of H and $\mathrm{FCOV}(\Omega)$ denotes the minimum size of a so-called fractional edge cover of the maximal clique Ω [30]. The primal graph of a hypergraph is constructed by inducing a clique on each hyperedge, and therefore the size of its minimum edge clique cover is at most m, the number of hyperedges.

In phylogenetics a central problem is to construct an evolutionary tree of a set of n taxa (i.e. species) based on k characters (i.e. attributes) describing them [34]. For example, when the character data is drawn from molecular sequences, the number of characters k can be much larger than the number of taxa n [24]. Deciding if the taxa admit a perfect phylogeny can be reduced to the chordal sandwich problem on the partition intersection graph of the characters [19, 34]. Moreover, the optimization version of perfect phylogeny, called the maximum compatibility problem of phylogenetic characters, can be reduced to the weighted minimum fill-in problem on the partition intersection graph if the characters are binary [4, 19]. The partition intersection graph has a vertex for each character-state pair, and its edges are constructed by inducing a clique corresponding to each taxon [34]. Hence the size of its minimum edge clique cover is at most n, the number of taxa.

A third setting in which parameterization by `cc` is motivated is probabilistic inference. Given a Bayesian network, the first step of efficient probabilistic inference algorithms is to compute a tree decomposition of small width of the moral graph of the Bayesian network [22, 23]. The moral graph is constructed as a union of n' cliques, where n' is the number of non-root nodes of the Bayesian network [23], and thus the size of its minimum edge clique cover is at most n'.

This paper is also directly motivated by observations in practice. Starting from the Second Parameterized Algorithms and Computational Experiments challenge (PACE 2017) [10], algorithm implementations based on potential maximal cliques have been observed to outperform other exact algorithm implementations on problems formulated as finding optimal triangulations [25, 26, 27, 31, 36, 37]. In particular, this paper is motivated by experimental observations of the usefulness of potential maximal cliques in computing hypergraph parameters [25, 26] and in phylogenetics [25, 27].

Our parameterization can be justified by real-world instances with small edge clique covers. In the context of fractional hypertreewidth, 708 of the 3072 hypergraphs in the standard HyperBench library [11] have $m < n/2$, where n is the number of vertices and m is the number of hyperedges. In the context of phylogenetics, an instance describing mammal mitochondrial sequences [20] has 7 taxa while its partition intersection graph has 245 vertices and an instance describing Indo-European languages [32] has 24 taxa while its partition intersection graph has 864 vertices. In the context of Bayesian networks, the Bayesian network "Andes" [8], accessed from the standard BNlearn repository [33], has 223 nodes of which 134 are non-root.

1.2 Techniques

The algorithms that we design in this paper are based on the framework of potential maximal cliques (PMCs) [5, 14]. Algorithms in this framework typically consist of two phases. In the first phase the set $\Pi(G)$ of PMCs of the input graph G is enumerated, and in the second phase dynamic programming over the PMCs is performed in time $O^*(|\Pi(G)|)$. The second phase of the PMC framework has already been formulated for all of the problems that we consider [14, 16, 17, 19, 30], so our $O^*(3^{\mathtt{cc}})$ time algorithms follow from an $O^*(3^{\mathtt{cc}})$ time PMC enumeration algorithm that we give. This algorithm is based on the Bouchitté–Todinca algorithm [6]. We achieve the $O^*(3^{\mathtt{cc}})$ bound by novel characterizations of minimal separators and PMCs with respect to an edge clique cover. In particular, we show that minimal separators correspond to bipartitions of an edge clique cover and almost all potential maximal cliques correspond to tripartitions of an edge clique cover.

On some of the problems we improve the time complexity to $O^*(2^{\mathtt{cc}'})$, where $\mathtt{cc}'$ is the size of an edge clique cover given as an input. The $O^*(2^{\mathtt{cc}'})$ time algorithms use the same dynamic programming states as the standard PMC framework, but instead of using PMCs for transitions we use fast subset convolution [2]. The application of fast subset convolution requires ad-hoc techniques for each problem to take into account the cost caused by the PMC implicitly selected by the convolution.

We also give algorithms that work in polynomial space and in times $O^*(9^{\mathtt{cc}'})$ and $O^*(9^{\mathtt{cc}+O(\log^2 \mathtt{cc})})$. These algorithms are based on a polynomial space and $O^*(9^{\mathtt{cc}})$ time algorithm for enumerating PMCs and on a lemma asserting that every minimal triangulation of a graph G has a maximal clique that is in a sense a balanced separator with respect to an edge clique cover of G.

1.3 Contributions

We start by giving bounds for the numbers of minimal separators and PMCs.

▶ **Theorem 1.** *If G is a graph with an edge clique cover of size $\mathtt{cc}$, then the number of minimal separators of G is at most $2^{\mathtt{cc}}$ and the number of potential maximal cliques of G is at most $3^{\mathtt{cc}}$.*

There are $O^*(|\Delta(G)|)$ time algorithms for enumerating the minimal separators $\Delta(G)$ of a graph G [1, 35], so it follows that the minimal separators of a graph can be enumerated in $O^*(2^{\mathtt{cc}})$ time. For enumerating PMCs no algorithms that are linear in the size of the output and polynomial in the size of the input are known[1]. Despite that, we are able to design an efficient algorithm for enumerating PMCs parameterized by $\mathtt{cc}$, even when no edge clique cover is given as input. The fact that the algorithm works even when no edge clique cover is given as input is crucial because there is no $O^*(2^{2^{o(\mathtt{cc})}})$ time parameterized algorithm for minimum edge clique cover assuming the exponential time hypothesis [9].

▶ **Theorem 2.** *There is an algorithm that given a graph G whose minimum edge clique cover has size $\mathtt{cc}$ enumerates the potential maximal cliques of G in $O^*(3^{\mathtt{cc}})$ time.*

It follows that all problems that can be solved in $O^*(|\Pi(G)|)$ time when the set $\Pi(G)$ of PMCs of the input graph G is given can be solved in $O^*(3^{\mathtt{cc}})$ time, even when no edge clique cover is given as input. We remark that in addition to the problems mentioned earlier, the set

[1] Obtaining an output-linear input-polynomial time algorithm for enumerating PMCs has been explicitly stated as an open problem in 2006 [3] and to the best of our knowledge is still open.

of such problems includes all problems in the framework called *maximum induced subgraph of bounded treewidth* [16], including for example the problems maximum independent set, minimum feedback vertex set, and longest induced path [16].

▶ **Corollary 3.** *Treewidth, weighted minimum fill-in, and all instances of maximum induced subgraph of bounded treewidth can be solved in $O^*(3^{\mathtt{cc}})$ time, where $\mathtt{cc}$ is the size of a minimum edge clique cover of the input graph. Fractional hypertreewidth can be solved in $O^*(3^m)$ time, where m is the number of hyperedges. The maximum compatibility problem of binary phylogenetic characters can be solved in $O^*(3^n)$ time, where n is the number of taxa.*

When an edge clique cover of size $\mathtt{cc}'$ is given as an input, some of the algorithms can be optimized to $O^*(2^{\mathtt{cc}'})$ time.

▶ **Theorem 4.** *Treewidth, minimum fill-in, and chordal sandwich can be solved in $O^*(2^{\mathtt{cc}'})$ time, where $\mathtt{cc}'$ is the size of an edge clique cover given as an input.*

▶ **Corollary 5.** *The perfect phylogeny problem can be solved in $O^*(2^n)$ time, where n is the number of taxa.*

A previous parameterized algorithm for perfect phylogeny works in time $O^*(4^r)$, where $r \leq n$ is the arity of characters [24]. Our $O^*(2^n)$ time algorithm improves over it in the case when $r > n/2$. This case is motivated by the fact that the $O^*(4^r)$ algorithm works for partial characters only via a reduction that sets $r \geq nf$ for a fraction f of missing data [34].

We also give polynomial space algorithms for some of the problems. The algorithms work in $O^*(9^{\mathtt{cc}'})$ time when an edge clique cover of size $\mathtt{cc}'$ is given as an input and in $O^*(9^{\mathtt{cc}+O(\log^2 \mathtt{cc})})$ time when the parameter $\mathtt{cc}$ is given as an input.

▶ **Theorem 6.** *Treewidth and weighted minimum fill-in can be solved in polynomial space and $O^*(9^{\mathtt{cc}'})$ time, where $\mathtt{cc}'$ is the size of an edge clique cover given as an input. Furthermore, fractional hypertreewidth can be solved in polynomial space and $O^*(9^m)$ time.*

▶ **Corollary 7.** *The perfect phylogeny problem and the maximum compatibility problem of binary phylogenetic characters can be solved in polynomial space and $O^*(9^n)$ time, where n is the number of taxa.*

▶ **Theorem 8.** *Treewidth and weighted minimum fill-in can be solved in polynomial space and $O^*(9^{\mathtt{cc}+O(\log^2 \mathtt{cc})})$ time, where $\mathtt{cc}$ is an integer given as an input that is at least the size of a minimum edge clique cover of the input graph.*

Finally, we demonstrate the tightness of the bounds on the numbers of minimal separators and potential maximal cliques.

▶ **Theorem 9.** *There is a family of graphs, containing for each positive integer $\mathtt{cc}$ a graph with edge clique cover of size $\mathtt{cc}$, $O(\mathtt{cc}^2)$ vertices, $\Theta(2^{\mathtt{cc}})$ minimal separators, and $\Theta(3^{\mathtt{cc}})$ potential maximal cliques.*

Furthermore, the parameter edge clique cover cannot be relaxed to vertex clique cover while retaining FPT, or even XP, bounds.

▶ **Theorem 10.** *There is a family of graphs, containing for each positive integer n a graph with n vertices, vertex clique cover of size 2, and $\Theta(2^{n/2})$ minimal separators.*

1.4 Related Work

The prior FPT algorithms for enumerating PMCs include an $O^*(4^{\mathtt{vc}})$ time algorithm, where $\mathtt{vc}$ is the size of a minimum vertex cover, and an $O^*(1.7347^{\mathtt{mw}})$ time algorithm, where $\mathtt{mw}$ is the modular width [15], extending the $O(1.7347^n)$ time algorithm, where n is the number of vertices [16]. One can see that edge clique cover and vertex cover are orthogonal parameters by considering complete graphs and star graphs. For modular width, the relation $\mathtt{mw} \leq 2^{\mathtt{cc}}$ holds, but there are graphs with $\mathtt{mw} = 2^{\mathtt{cc}} - 2$. In the conclusion of [15] the authors mentioned that they are not aware of other FPT parameters than $\mathtt{vc}$ and $\mathtt{mw}$ for PMCs and asked whether more parameterizations could be obtained.

Other parameterized approaches on the PMC framework include an FPT modulator parameter [28] and an XP parameterization for minimal separators in H-graphs [13]. The modulator parameter is orthogonal to edge clique cover. On H-graphs, the graphs with edge clique cover of size $\mathtt{cc}$ are $K_{\mathtt{cc}}$-graphs, so the H-graph parameterization implies an $n^{O(\mathtt{cc}^2)}$ time algorithm for enumerating PMCs.

In addition to the already discussed $O^*(4^r)$ time algorithm for perfect phylogeny [24], we are not aware of prior single-exponential FPT algorithms with the same parameters as our algorithms. For fractional hypertreewidth there are parameterized algorithms whose parameters depend on the sizes of intersections of hyperedges [12]. For treewidth and chordal sandwich, different techniques have been used to obtain an $O^*(3^{\mathtt{vc}})$ time algorithm for treewidth [7] and an $O^*(2^{\mathtt{vc}'})$ time algorithm for chordal sandwich, where $\mathtt{vc}'$ is the size of a minimum vertex cover of the admissible edge set [21].

1.5 Organization of the Paper

The proofs of lemmas marked with $\star$ are omitted and can be found in the full version of the paper. In Section 2 we give necessary definitions and background on minimal triangulations and PMCs. In Section 3 we characterize minimal separators and PMCs based on edge clique cover, proving Theorem 1. In Section 4 we give enumeration algorithms for PMCs, proving Theorem 2. In Section 5 we give faster algorithms for the case when an edge clique cover is given as an input, proving Theorem 4. In Section 6 we give polynomial space algorithms, proving Theorems 6 and 8. In Section 7 we demonstrate the tightness of our results, proving Theorems 9 and 10. Proofs for the relation of modular width and edge clique cover claimed in Section 1.4 can be found in the full version of the paper. We conclude in Section 8.

2 Preliminaries

We recall the standard graph notation that we use and preliminaries on minimal triangulations. We also give formal definitions of the problems that we consider and introduce our notation related to edge clique cover.

2.1 Notation on Graphs

We consider graphs that are finite, simple, and undirected. We assume that the graphs given as input are connected. For graphs with multiple connected components, the algorithms can be applied to each connected component independently. The sets of vertices and edges of a graph G are denoted by $V(G)$ and $E(G)$, respectively. The set of edges of a complete graph with vertex set X is X^2. The subgraph $G[X]$ induced by $X \subseteq V(G)$ has $V(G[X]) = X$ and $E(G[X]) = E(G) \cap X^2$. We also use the notation $G \setminus X = G[V(G) \setminus X]$. The vertex sets of connected components of a graph G are $\mathcal{C}(G)$. The set of neighbors of a vertex v is denoted

by $N(v)$ and the set of neighbors of a vertex set X by $N(X) = \bigcup_{v \in X} N(v) \setminus X$. The closed neighborhood of a vertex v is $N[v] = N(v) \cup \{v\}$ and the closed neighborhood of a vertex set X is $N[X] = N(X) \cup X$. A clique of a graph G is a vertex set X such that $G[X]$ is complete. The set of inclusion maximal cliques of G is denoted by $\mathrm{MC}(G)$.

2.2 Minimal Triangulations

A graph is *chordal* if it has no induced cycle of four or more vertices. A chordal graph H is a *triangulation* of a graph G if $V(G) = V(H)$ and $E(G) \subseteq E(H)$. A triangulation H of G is a *minimal triangulation* of G if there is no triangulation H' of G with $E(H') \subsetneq E(H)$. The edges in $E(H) \setminus E(G)$ are called *fill-edges*. A vertex set $\Omega \subseteq V(G)$ is a *potential maximal clique* (PMC) of G if there is a minimal triangulation H of G such that $\Omega \in \mathrm{MC}(H)$. The set of PMCs of G is denoted by $\Pi(G)$.

A vertex set S is a *minimal a, b-separator* of graph G if the vertices a and b are in different components of $G \setminus S$, and S is inclusion minimal in this regard. A *full component* of a vertex set X is a component $C \in \mathcal{C}(G \setminus X)$ with $N(C) = X$. We note that S is a minimal a, b-separator if and only if S has distinct full components containing a and b. A vertex set S is a *minimal separator* if it is a minimal a, b-separator for some pair a, b, i.e., it has at least two full components. We denote the set of minimal separators of G with $\Delta(G)$.

A *block* of a graph G is a vertex set $C \subseteq V(G)$ such that $N(C) \in \Delta(G)$. We remark that a common notation is to call such a pair $(N(C), C)$ a full block [5]. In modern formulations of the PMC framework the concept of non-full blocks is not needed [16], so we simplify the notation by identifying the block with only the vertex set C.

Next we recall a couple of required propositions on the structure of PMCs.

▶ **Proposition 11** ([5]). *A vertex set $\Omega \subseteq V(G)$ is a PMC of a graph G if and only if*

1. *$N(C) \subsetneq \Omega$ for all $C \in \mathcal{C}(G \setminus \Omega)$, i.e., no component of Ω is full, and*
2. *for all pairs of distinct vertices $u, v \in \Omega$, either $\{u, v\} \in E(G)$ or there is a component $C \in \mathcal{C}(G \setminus \Omega)$ with $\{u, v\} \subseteq N(C)$.*

We will refer to condition 1 of Proposition 11 as the *no full component condition* and to condition 2 as the *cliquish condition*. Note that Proposition 11 implies an $O(nm)$ time algorithm for testing if a vertex set is a PMC [5].

▶ **Proposition 12** ([5]). *If Ω is a PMC of a graph G then all components $C \in \mathcal{C}(G \setminus \Omega)$ are blocks of G.*

We call the components $C \in \mathcal{C}(G \setminus \Omega)$ the blocks of Ω. Note that $N(C) \subsetneq \Omega$, i.e., the minimal separators of the blocks $C \in \mathcal{C}(G \setminus \Omega)$ are strict subsets of the PMC. We also need the following proposition connecting PMCs and blocks.

▶ **Proposition 13** ([5]). *If Ω is a PMC of a graph G and C is a block of Ω, then there is a full component C' of $N(C)$ such that $\Omega \subseteq N[C']$.*

The following lemma, which follows from Proposition 11, simplifies some of our proofs.

▶ **Lemma 14** ($\star$). *If Ω is a PMC of a graph G and contains a vertex $v \in \Omega$ such that no block $C \in \mathcal{C}(G \setminus \Omega)$ has $v \in N(C)$, then $\Omega = N[v]$.*

2.3 Definitions of Problems

Let $\mathrm{Tr}(G)$ denote the set of triangulations of a graph G. The treewidth of a graph G is $\min_{H\in\mathrm{Tr}(G)}\max_{\Omega\in\mathrm{MC}(H)}|\Omega|-1$. The minimum fill-in of G is $\min_{H\in\mathrm{Tr}(G)}|E(H)\setminus E(G)|$. Given a weight function $w : V(G)^2 \to \mathbb{R}_{\geq 0}$, the weighted minimum fill-in of G with respect to w is $\min_{H\in\mathrm{Tr}(G)}\sum_{e\in(E(H)\setminus E(G))} w(e)$. Given a graph G and a set of admissible edges $F \subseteq V(G)^2 \setminus E(G)$, the chordal sandwich problem is to determine if there is a triangulation H of G with $E(H) \subseteq E(G) \cup F$. Note that chordal sandwich can be reduced to weighted minimum fill-in on the same graph G.

A hypergraph $\mathcal{G}$ has a set of vertices $V(\mathcal{G})$ and a set of hyperedges $E(\mathcal{G})$ that are arbitrary subsets of vertices. The primal graph $P(\mathcal{G})$ of $\mathcal{G}$ has vertices $V(P(\mathcal{G})) = V(\mathcal{G})$ and edges $E(P(\mathcal{G})) = \bigcup_{e\in E(\mathcal{G})} e^2$. A fractional edge cover of a set $X \subseteq V(\mathcal{G})$ is an assignment $c : E(\mathcal{G}) \to \mathbb{R}_{\geq 0}$ so that for each $v \in X$ it holds that $\sum_{v\in e\in E(\mathcal{G})} c(e) \geq 1$. The size of a fractional edge cover c is $\sum_{e\in E(\mathcal{G})} c(e)$. The minimum size of a fractional edge cover of a set $X \subseteq V(\mathcal{G})$ is denoted by $\mathrm{FCOV}(X)$. Note that $\mathrm{FCOV}(X)$ can be computed in polynomial time by linear programming [18]. The fractional hypertreewidth of a hypergraph $\mathcal{G}$ is $\min_{H\in\mathrm{Tr}(P(\mathcal{G}))}\max_{\Omega\in\mathrm{MC}(H)}\mathrm{FCOV}(\Omega)$ [18, 30].

For all of the aforementioned problems there is an optimal solution corresponding to a minimal triangulation. Furthermore, all of the problems can be solved in $O^*(|\Pi(G)|)$ time when $\Pi(G)$ is given as an input [14, 17, 19, 29, 30].

2.4 Notation on Edge Clique Cover

An edge clique cover of a graph G is a collection $\mathcal{W}$ of cliques of G so that $\bigcup_{W\in\mathcal{W}} W^2 = E(G)$. We often manipulate vertex sets based on an edge clique cover $\mathcal{W}$. For a vertex $v \in V(G)$, we denote by $\mathcal{W}[v] = \{W \in \mathcal{W} \mid v \in W\}$ the set of cliques in $\mathcal{W}$ that contain v. Similarly, for a vertex set X we denote by $\mathcal{W}[X] = \bigcup_{v\in X}\mathcal{W}[v]$ the set of cliques in $\mathcal{W}$ that intersect X.

A non-empty subset $\mathcal{W}' \subseteq \mathcal{W}$ of an edge clique cover $\mathcal{W}$ is called a part of the edge clique cover. The vertices in $V(G,\mathcal{W}') = \{v \in V(G) \mid \mathcal{W}[v] \subseteq \mathcal{W}'\}$ are called the vertices of the part $\mathcal{W}'$. We use a shorthand $V(G,\mathcal{W}_1,\ldots,\mathcal{W}_p) = V(G,\mathcal{W}_1)\cup\ldots\cup V(G,\mathcal{W}_p)$ to denote the union of vertices of multiple parts. The components of a part are $\mathcal{C}(\mathcal{W}') = \mathcal{C}(G[V(G,\mathcal{W}')])$. A part is called good if all of its components are blocks, in which case the components of the part may be called the blocks of the part. Note that a part $\mathcal{W}'$ with $V(G,\mathcal{W}') = \emptyset$ is good. Two disjoint parts $\mathcal{W}_1, \mathcal{W}_2$ are called compatible if $V(G,\mathcal{W}_1,\mathcal{W}_2) = V(G,\mathcal{W}_1\cup\mathcal{W}_2)$.

3 Characterization of the Central Combinatorial Objects

In this section we show that if a graph has an edge clique cover of size $\mathtt{cc}$, then the number of blocks and minimal separators of the graph is at most $2^{\mathtt{cc}}$ and the number of potential maximal cliques is at most $3^{\mathtt{cc}}$. The characterizations of these objects will be later used in the design of the algorithms.

We start by showing that blocks correspond to parts of an edge clique cover.

▶ **Lemma 15.** *Let G be a graph and $\mathcal{W}$ an edge clique cover of G. If C is a block of G then $V(G,\mathcal{W}[C]) = C$.*

Proof. Clearly $C \subseteq V(G,\mathcal{W}[C])$. Note that $V(G,\mathcal{W}[C]) \subseteq N[C]$ because any vertex v intersecting a common clique with a vertex $u \in C$ must be a neighbor of u. Suppose there is a vertex $v \in (V(G,\mathcal{W}[C]) \cap N(C))$. Let C' be a full component of $N(C)$ distinct from C, implying that $v \in N(C')$. All the cliques that v intersects also intersect with C, and therefore there must be a vertex in C' that is in a clique intersecting with C which is a contradiction to the fact that $N(C)$ separates C and C'. ◀

By Lemma 15, any block C of G can be uniquely identified with a set $\mathcal{W}[C] \subseteq \mathcal{W}$. Therefore, the number of blocks is at most $2^{\mathtt{cc}}$. Any minimal separator is identified as $N(C)$ of at least two blocks C, so the number of minimal separators is at most $2^{\mathtt{cc}-1}$.

Next we show that each PMC is either a closed neighborhood of a vertex or can be represented by a tripartition of edge clique cover.

▶ **Lemma 16.** *Let G be a graph and $\mathcal{W}$ an edge clique cover of G. If Ω is a potential maximal clique of G, then either (1) $\Omega = N[v]$ for a vertex $v \in V(G)$ or (2) $\mathcal{C}(G \setminus \Omega) = \mathcal{C}(\mathcal{W}_1) \cup \mathcal{C}(\mathcal{W}_2) \cup \mathcal{C}(\mathcal{W}_3)$, where $\{\mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_3\}$ is a partition of $\mathcal{W}$ into good parts.*

Proof. Suppose that case 1 does not apply, i.e., Ω is not equal to $N[v]$ for any $v \in V(G)$. By Proposition 12 the components $C \in \mathcal{C}(G \setminus \Omega)$ are blocks and therefore by Lemma 15 they define a collection $P = \{\mathcal{W}[C] \mid C \in \mathcal{C}(G \setminus \Omega)\}$ of disjoint good parts of $\mathcal{W}$. If the collection P is not a partition of $\mathcal{W}$, add an additional part $\mathcal{W}' = \mathcal{W} \setminus (\bigcup_{\mathcal{W}_i \in P} \mathcal{W}_i)$ to the collection to make it a partition of $\mathcal{W}$. We have that $V(G, \mathcal{W}') = \emptyset$ because if there would be a vertex v with $\mathcal{W}[v] \subseteq \mathcal{W}'$, then v would be in Ω because it is not in any block, and also v would not be in the neighborhood of any block, so by Lemma 14 we would have $\Omega = N[v]$. Now we have a partition P of $\mathcal{W}$ into good parts with $\bigcup_{\mathcal{W}_i \in P} \mathcal{C}(\mathcal{W}_i) = \mathcal{C}(G \setminus \Omega)$.

The partition P has at least two parts because otherwise Ω would be empty. If the number of parts is two, i.e. $P = \{\mathcal{W}_1, \mathcal{W}_2\}$, let $\mathcal{W}_1$ be a part such that $V(G, \mathcal{W}_1)$ is not empty. Such a part exists because otherwise $\Omega = V(G)$ and case 1 would apply. No vertex of Ω is in $V(G, \mathcal{W}_2)$, so $V(G, \mathcal{W}_1)$ is a full component of Ω, which is a contradiction to the no full component condition.

If the number of parts is at least three, merge arbitrary compatible pairs of parts until the number of parts is three or no pairs of parts can be merged anymore. If we end up with three parts, we are done. If we end up with more than three parts, i.e. $P = \{\mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_3, \mathcal{W}_4, \ldots\}$, then take a vertex $u \in (V(G, \mathcal{W}_1 \cup \mathcal{W}_2) \setminus V(G, \mathcal{W}_1, \mathcal{W}_2))$ and a vertex $v \in (V(G, \mathcal{W}_3 \cup \mathcal{W}_4) \setminus V(G, \mathcal{W}_3, \mathcal{W}_4))$. Because of our assumption that we cannot continue the merging process anymore both of these vertices exist and are in Ω. However, there is no edge between u and v and there is no common component in whose neighborhood u and v are, which is a contradiction to the cliquish condition. ◀

We call the PMCs corresponding to case 1 of Lemma 16 type 1 PMCs and the PMCs corresponding to case 2 type 2 PMCs. The number of type 1 PMCs is at most n, the number of vertices. Another upper bound for the number of type 1 PMCs is $2^{\mathtt{cc}}$, because if $\mathcal{W}[v] = \mathcal{W}[u]$ for vertices v and u then $N[v] = N[u]$. The number of type 2 PMCs is at most $S(\mathtt{cc}, 3)$, where S denotes the Stirling numbers of the second kind. One can verify that $S(\mathtt{cc}, 3) + 2^{\mathtt{cc}} \leq 3^{\mathtt{cc}}$ and the bound for PMCs follows.

4 Enumeration Algorithms

In this section we modify the Bouchitté–Todinca algorithm [6] for enumerating PMCs to give an $O^*(3^{\mathtt{cc}})$ time PMC enumeration algorithm and a polynomial space $O^*(9^{\mathtt{cc}})$ time PMC enumeration algorithm.

The Bouchitté–Todinca algorithm is based on theorems characterizing PMCs based on minimal separators. We summarize the theorems in the following proposition.

▶ **Proposition 17** ([6]). *Let G be a connected graph with $|V(G)| > 1$ and v any vertex of G. If Ω is a PMC of G, one of the following holds.*

1. $\Omega \setminus \{v\} \in \Pi(G \setminus \{v\})$.
2. $\Omega \setminus \{v\} \in \Delta(G)$.
3. $\Omega = S \cup T$, *where* $S \in \Delta(G)$ *and* $T \in \Delta(G[C \cup \{x, y\}])$, *where C is a full component of S and x and y are non-adjacent vertices in S.*

The algorithm uses case 1 to generate n induced subgraphs of the input graph, from which PMCs are generated by cases 2 and 3. Note that the size of a minimum edge clique cover is monotone with respect to induced subgraphs. We need the following lemma, which follows from Proposition 11, to ensure that each PMC of each induced subgraph corresponds to at most one PMC of the original graph.

▶ **Lemma 18** ([6]). *Let G be a graph and $v \in V(G)$. If $\Omega \in \Pi(G \setminus \{v\})$, then at most one of Ω and $\Omega \cup \{v\}$ is a PMC of G.*

Now, we can just enumerate PMCs from cases 2 and 3 in each of the n induced subgraphs, and each time a PMC is found we use Lemma 18 to generate at most one PMC of the original graph in polynomial time.

Next we complete the description of the algorithm by showing that PMCs from cases 2 and 3 can be enumerated in $O^*(3^{\mathsf{cc}})$ time. The proof is based on Lemma 15 on the structure and the number of blocks.

▶ **Lemma 19.** *There is an algorithm that given a graph G whose minimum edge clique cover has size cc enumerates the PMCs of G, possibly with duplicates, in polynomial space and $O^*(3^{\mathsf{cc}})$ time.*

Proof. By Lemma 18, it is sufficient to enumerate PMCs corresponding to cases 2 and 3 of Proposition 17. For case 2, the bound follows from the 2^{cc} bound on minimal separators and a polynomial space $O^*(|\Delta(G)|)$ time minimal separator enumeration algorithm [35]. For case 3, we do polynomial space enumeration of minimal separators in the graph G, and every time we output a minimal separator S, we do polynomial space enumeration of minimal separators in the graph $G[C \cup \{x, y\}]$ for all full components C of S and all non-adjacent pairs $x, y \in S$.

We do the inner iteration $O(n^2)$ times for each block C of G. The complexity of the inner iteration depends on the size of a minimum edge clique cover of $G[C \cup \{x, y\}]$. Let $\mathcal{W}$ be a minimum edge clique cover of G. By Lemma 15, the block C corresponds to an unique subset $\mathcal{W}[C]$ of $\mathcal{W}$. The subset $\mathcal{W}[C]$ is an edge clique cover of $G[C \cup \{x, y\}]$ because all edges in it are adjacent to C because x and y are non-adjacent. Therefore, the time complexity of the inner iteration is $O^*(2^{|\mathcal{W}[C]|})$, and therefore, the time complexity of the algorithm is at most $\sum_{\mathcal{W}' \subseteq \mathcal{W}} O^*(2^{|\mathcal{W}'|}) = O^*(3^{\mathsf{cc}})$. ◀

Using for example sorting we can deduplicate the output of the algorithm of Lemma 19 and an $O^*(3^{\mathsf{cc}})$ time exponential space algorithm for enumerating PMCs without duplicates follows. For deduplication in polynomial space, we use a simple trick that is efficient enough for our purposes.

▶ **Lemma 20.** *There is an algorithm that given a graph G whose minimum edge clique cover has size cc enumerates the PMCs of G in polynomial space and $O^*(9^{\mathsf{cc}})$ time.*

Proof. Run the algorithm of Lemma 19 multiple times in succession, each time outputting the lexicographically smallest PMC that is lexicographically larger than the previous PMC outputted, until no such PMC is found. Now, using the algorithm at most 3^{cc} times we have outputted the PMCs of G in lexicographically strictly increasing order. ◀

5 Faster Algorithms When Edge Clique Cover is Given

We use fast subset convolution [2] to design $O^*(2^{\mathtt{cc}'})$ time algorithms for treewidth, minimum fill-in, and chordal sandwich, where $\mathtt{cc}'$ is the size of an edge clique cover given as an input. In particular, we make use of the following result.

▶ **Proposition 21** ([2]). *Let X be a set, $f : 2^X \to [M]$ and $g : 2^X \to [M]$ functions from the set 2^X of all subsets of X to the set of integers up to M. The function $(f * g)$ defined as $(f * g)(Y) = \min_{Y' \subseteq Y} f(Y') + g(Y \setminus Y')$ can be computed for all $Y \subseteq X$ in $O^*(2^{|X|}M)$ time.*

The algorithms we introduce are modifications of the dynamic programming phase of the PMC framework. In most of this section our presentation is general in the sense that it applies to each of the three problems. We use the term "optimal triangulation" to refer to a triangulation with the minimum size of a maximum clique in the context of treewidth, a triangulation with the least number of edges in the context of minimum fill-in, and to a triangulation that has no non-admissible fill-edges in the context of chordal sandwich (or to information that no such triangulation exists).

We start by recalling the dynamic programming phase of the PMC framework. The states of the dynamic programming correspond to *realizations* of blocks.

▶ **Definition 22** ([5]). *Let G be a graph and C a block of G. A realization $R(C)$ of C is a graph with $V(R(C)) = N[C]$ and $E(R(C)) = E(G[N[C]]) \cup N(C)^2$.*

The following proposition characterizes minimal triangulations of a realization of a block.

▶ **Proposition 23** ([5]). *Let G be a graph and C a block of G. The graph H is a minimal triangulation of $R(C)$ if and only if (i) $V(H) = N[C]$ and (ii) there is a PMC $\Omega \in \Pi(G)$ with $N(C) \subseteq \Omega \subseteq N[C]$ and*

$$E(H) = \Omega^2 \cup \bigcup_{C_i \in \mathcal{C}(R(C) \setminus \Omega)} E(H_i),$$

where H_i is any minimal triangulation of $R(C_i)$. It also holds that each C_i is a block of G.

Proposition 23 implies dynamic programming formulas for computing optimal triangulations of realizations of all blocks [5, 14, 29].

We use the following proposition for a base case.

▶ **Proposition 24** ([5]). *Let G be a graph that is not complete. The graph H is a minimal triangulation of G if and only if (i) $V(H) = V(G)$ and (ii) there is a minimal separator $S \in \Delta(G)$ with*

$$E(H) = \bigcup_{C_i \in \mathcal{C}(G \setminus S)} E(H_i),$$

where H_i is any minimal triangulation of $R(C_i)$. It also holds that each C_i is a block of G.

Once we have computed optimal triangulations of realizations of all blocks, we can compute an optimal triangulation of the graph via Proposition 24 in time $O^*(2^{\mathtt{cc}})$. For computing optimal triangulations of realizations, the bottleneck in implementing the recursion of Proposition 23 is in iterating over the PMCs.

The high-level idea of our algorithm is that we use Proposition 23 directly only with type 1 PMCs, i.e., PMCs $\Omega = N[v]$ for some vertex $v \in V(G)$. For type 2 PMCs, we simulate the iteration over PMCs with fast subset convolution. In particular, we show that each PMC

Ω of type 2 with $N(C) \subseteq \Omega \subseteq N[C]$ can be expressed in terms of two disjoint good parts $\mathcal{W}_1$ and $\mathcal{W}_2$ of $\mathcal{W}[C]$, where $\mathcal{W}$ is an edge clique cover of G. In the case of treewidth, minimum fill-in, and chordal sandwich, an optimal partition of every subset $\mathcal{W}' \subseteq \mathcal{W}$ into two good parts $\mathcal{W}_1$ and $\mathcal{W}_2$ can be computed with fast subset convolution, provided that we have first computed optimal triangulations of realizations of all blocks in $\mathcal{C}(\mathcal{W}_1)$ and $\mathcal{C}(\mathcal{W}_2)$.

We first show the direction that each PMC can be expressed in terms of $\mathcal{W}_1$ and $\mathcal{W}_2$.

▶ **Lemma 25.** *Let G be a graph, $\mathcal{W}$ an edge clique cover of G, and C a block of G. If a graph H is a minimal triangulation of $R(C)$, then either (1) there is a vertex $v \in V(G)$ and*

$$E(H) = N[v]^2 \cup \bigcup_{C_i \in \mathcal{C}(R(C) \setminus N[v])} E(H_i),$$

where $N[v] \in \Pi(G)$, $N(C) \subseteq N[v] \subseteq N[C]$, and H_i is a minimal triangulation of $R(C_i)$, or (2) there is a partition $\{\mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_o\}$ of $\mathcal{W}$ into good parts with $\mathcal{W}_1 \cup \mathcal{W}_2 \subseteq \mathcal{W}[C]$, a block $C' \in \mathcal{C}(\mathcal{W}_o)$ with $N(C) = N(C')$, and

$$E(H) = \Omega^2 \cup \bigcup_{C_i \in \mathcal{C}(\mathcal{W}_1) \cup \mathcal{C}(\mathcal{W}_2) \cup (\mathcal{C}(\mathcal{W}_o) \setminus \mathcal{C}(G \setminus N(C)))} E(H_i),$$

where $\Omega = V(G) \setminus V(G, \mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_o)$ and H_i is a minimal triangulation of $R(C_i)$.

Proof. Case 1 corresponds to Proposition 23 with PMCs of type 1. Next we prove that case 2 covers all PMCs of type 2.

Let Ω be any PMC of G with $N(C) \subseteq \Omega \subseteq N[C]$ such that there is no vertex v with $\Omega = N[v]$. Consider the part $\mathcal{W}'_o = \mathcal{W} \setminus \mathcal{W}[C]$ and let us prove that $\mathcal{W}'_o$ is a good part and $\mathcal{C}(\mathcal{W}'_o) = \mathcal{C}(G \setminus N(C)) \setminus \{C\}$. Observe that $\{C, N(C), V(G, \mathcal{W}'_o)\}$ is a partition of $V(G)$, and moreover there are no edges between C and $V(G, \mathcal{W}'_o)$. Now, for any component $C' \in \mathcal{C}(\mathcal{W}'_o)$, it must hold that $N(C') \subseteq N(C)$, and therefore $N(C')$ is a minimal separator and therefore $\mathcal{W}'_o$ is a good part whose blocks are the components of $G \setminus N(C)$ except C.

Similarly as in the proof of Lemma 16, consider the collection of disjoint good parts $P = \{\mathcal{W}[C_i] \mid C_i \in \mathcal{C}(G \setminus \Omega)\}$. All of the parts that intersect $\mathcal{W}'_o$ are subsets of $\mathcal{W}'_o$ because they do not intersect $\mathcal{W}[C]$, and therefore we replace the parts that intersect $\mathcal{W}'_o$ by the part $\mathcal{W}'_o$. Now by similar arguments as in Lemma 16 we can add one additional part to the collection to make it a partition of $\mathcal{W}$. Now we have a partition P of $\mathcal{W}$ with $|P| \geq 2$ and $\mathcal{W}'_o \in P$. Moreover, $\bigcup_{\mathcal{W}_i \in P} \mathcal{C}(\mathcal{W}_i) = \mathcal{C}(G \setminus \Omega)$. If $|P| = 2$, then it would hold that $P = \{\mathcal{W}'_o, \mathcal{W}[C]\}$, in which case $\Omega = N(C)$ would hold, which is a contradiction. If there are at least three parts, then merge compatible parts until we have three parts or cannot merge parts anymore. By the proof of Lemma 16, we will end up with three parts. Now let $\mathcal{W}_o$ be the part that contains $\mathcal{W}'_o$ and $\mathcal{W}_1$ and $\mathcal{W}_2$ the other two parts. Because $\mathcal{W}'_o = \mathcal{W} \setminus \mathcal{W}[C]$, we have that $\mathcal{W}_1 \cup \mathcal{W}_2 \subseteq \mathcal{W}[C]$. Moreover, because $N(C)$ has at least two full components, and all components of $N(C)$ except C are components of $\mathcal{W}'_o$, we have that there is a block $C' \in \mathcal{C}(\mathcal{W}_o)$ with $N(C') = N(C)$.

Finally, we need to show that $\mathcal{C}(R(C) \setminus \Omega) = \mathcal{C}(\mathcal{W}_1) \cup \mathcal{C}(\mathcal{W}_2) \cup (\mathcal{C}(\mathcal{W}_o) \setminus \mathcal{C}(G \setminus N(C)))$. By Proposition 13 we have that $\mathcal{C}(R(C) \setminus \Omega) = \mathcal{C}(G \setminus \Omega) \setminus \mathcal{C}(G \setminus N(C))$ and therefore $\mathcal{C}(R(C) \setminus \Omega) = \mathcal{C}(\mathcal{W}_1) \cup \mathcal{C}(\mathcal{W}_2) \cup \mathcal{C}(\mathcal{W}_o) \setminus \mathcal{C}(G \setminus N(C))$. All blocks of $\mathcal{W}_1$ and $\mathcal{W}_2$ are subsets of C because $\mathcal{W}_1 \cup \mathcal{W}_2 \subseteq \mathcal{W}[C]$. ◀

The following lemma guarantees that the characterization of PMCs of type 2 in Lemma 25 is sound in the sense that all graphs H that it defines are (not necessarily minimal) triangulations of G.

▶ **Lemma 26** (⋆). *Let G be a graph, $\mathcal{W}$ an edge clique cover of G, and C a block of G. Furthermore, let $\{\mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_o\}$ be a partition of $\mathcal{W}$ into good parts with $\mathcal{W}_1 \cup \mathcal{W}_2 \subseteq \mathcal{W}[C]$ and C' a block of G with $C' \in \mathcal{C}(\mathcal{W}_o)$ and $N(C) = N(C')$. Let H be any graph with (i) $V(H) = N[C]$ and (ii)*

$$E(H) = \Omega^2 \cup \bigcup_{C_i \in \mathcal{C}(\mathcal{W}_1) \cup \mathcal{C}(\mathcal{W}_2) \cup (\mathcal{C}(\mathcal{W}_o) \setminus \mathcal{C}(G \setminus N(C)))} E(H_i),$$

where $\Omega = V(G) \setminus V(G, \mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_o)$ and H_i is any triangulation of $R(C_i)$. The graph H is a triangulation of $R(C)$.

In Lemmas 25 and 26 we formulated a recursion that characterizes all minimal triangulations of a realization $R(C)$ of a block C in terms of minimal triangulations of realizations $R(C')$ of blocks $C' \subsetneq C$. What remains is to integrate the computation of the optimal cost of the triangulation into this characterization. The following lemma is used for treewidth and minimum fill-in. It is simple, but we state it as a warmup for what follows.

▶ **Lemma 27** (⋆). *Let G be a graph, $\mathcal{W}$ an edge clique cover of G, $\{\mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_o\}$ a partition of $\mathcal{W}$, and $\Omega = V(G) \setminus V(G, \mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_o)$. It holds that $|\Omega| = |V(G)| - |V(G, \mathcal{W}_1)| - |V(G, \mathcal{W}_2)| - |V(G, \mathcal{W}_o)|$.*

Therefore, the size of Ω can be computed as a sum that considers $\mathcal{W}_1$, $\mathcal{W}_2$, and $\mathcal{W}_o$ independently, and therefore we can integrate the computation of it into fast subset convolution. For treewidth, we only have to make sure that $|\Omega| \leq k + 1$, where k is the upper bound for treewidth in the decision problem. For minimum fill-in, we can compute the number of edges in the triangulation of $R(C)$ as $\binom{|\Omega|}{2} + \sum_{C_i \in \mathcal{C}(R(C) \setminus \Omega)} (|E(H_i)| - \binom{|N(C_i)|}{2})$, where H_i is an optimal triangulation of the realization $R(C_i)$.

A similar lemma is used for chordal sandwich.

▶ **Lemma 28** (⋆). *Let G be a graph, $\mathcal{W}$ an edge clique cover of G, $\{\mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_o\}$ a partition of $\mathcal{W}$ into good parts, and $\Omega = V(G) \setminus V(G, \mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_o)$. It holds that $\Omega^2 = \bigcup_{\mathcal{W}_i \in \{\mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_o\}} (V(G) \setminus V(G, \mathcal{W}_i, \mathcal{W} \setminus \mathcal{W}_i))^2$.*

Lemma 28 guarantees that each fill-edge caused by the PMC Ω can be "seen" from at least one of the parts $\mathcal{W}_1$, $\mathcal{W}_2$, $\mathcal{W}_o$, implying that it is sufficient to check each part independently to guarantee that Ω does not add any forbidden fill-edges. We remark that Lemma 28 appears to be difficult to generalize to count the exact number of fill-edges, which is the barrier why we are not able to give an $O^*(2^{\mathsf{cc}'})$ time algorithm for weighted minimum fill-in.

Algorithm 1 presents the full $O^*(2^{\mathsf{cc}'})$ time algorithm for treewidth. The algorithms for minimum fill-in and chordal sandwich are similar. The algorithm maintains a collection $\mathcal{B}$ of blocks C for which it is known that the treewidth of $R(C)$ is at most k. The invariant of the main loop of lines 3 to 19 is that after ith iteration, all blocks of size at most i and treewidth at most k have been added to $\mathcal{B}$. In each iteration of the main loop, the algorithm iterates over all good parts $\mathcal{W}'$ on lines 5 to 7, and if all realizations of blocks of the part have treewidth at most k adds the part to a collection $F_{|V(G,\mathcal{W}')|}$. These parts $\mathcal{W}'$ correspond to parts $\mathcal{W}_1$ and $\mathcal{W}_2$ of our lemmas. Then, fast subset convolution is applied on line 8 on the collections F to find for all combinations $(\mathcal{W}_1 \cup \mathcal{W}_2)$ of disjoint parts $\mathcal{W}_1$ and $\mathcal{W}_2$ the maximum number of vertices in $V(G, \mathcal{W}_1, \mathcal{W}_2)$. Then on lines 9 to 15 the algorithm iterates through all good parts $\mathcal{W}_o$, thus determining $(\mathcal{W}_1 \cup \mathcal{W}_2)$ and all other variables that need to be taken into account. In particular, note that each part $\mathcal{W}_o$ determines only polynomially many blocks C such that there is $C' \in \mathcal{C}(\mathcal{W}_o)$ with $N(C') = N(C)$.

Algorithm 1 Treewidth in $O^*(2^{\mathsf{cc}'})$ time.

Input : Connected graph G, an edge clique cover $\mathcal{W}$ of G, and an integer k
Output : Whether the treewidth of G is at most k

1 **if** G *is complete* **then return** $|V(G)| \leq k+1$
2 Let $\mathcal{B} \leftarrow \emptyset$ be a collection of blocks of G
3 **for** $i \leftarrow 1$ **to** n **do**
4 For each $0 \leq j \leq n$ let $F_j \leftarrow \emptyset$ be a collection of subsets of $\mathcal{W}$
5 **for** *each good part* $\mathcal{W}' \subseteq \mathcal{W}$ **do**
6 **if** $\mathcal{C}(\mathcal{W}') \subseteq \mathcal{B}$ **then**
7 $F_{|V(G,\mathcal{W}')|} \leftarrow F_{|V(G,\mathcal{W}')|} \cup \{\mathcal{W}'\}$
8 Use fast subset convolution to compute for each subset $\mathcal{W}' \subseteq \mathcal{W}$ the maximum value of $j+k$ such that there is $\mathcal{W}'' \subseteq \mathcal{W}'$ with $\mathcal{W}'' \in P_j$ and $(\mathcal{W}' \setminus \mathcal{W}'') \in P_k$
9 **for** *each good part* $\mathcal{W}_o \subseteq \mathcal{W}$ **do**
10 Let t be the value on $\mathcal{W} \setminus \mathcal{W}_o$ computed in line 8
11 **if** t *exists and* $n - t - |V(G, \mathcal{W}_o)| \leq k+1$ **then**
12 **for** *each minimal separator* $N(C')$ *with* $C' \in \mathcal{C}(\mathcal{W}_o)$ **do**
13 **if** *exists* $C \in \mathcal{C}(G \setminus N(C'))$ *with* $(\mathcal{W} \setminus \mathcal{W}_o) \subseteq \mathcal{W}[C]$ **then**
14 **if** $(\mathcal{C}(\mathcal{W}_o) \setminus \mathcal{C}(N(C))) \subseteq \mathcal{B}$ **then**
15 $\mathcal{B} \leftarrow \mathcal{B} \cup \{C\}$
16 **for** *each block* C *of* G **do**
17 **for** $v \in V(G) \mid N[v] \in \Pi(G)$ *and* $|N[v]| \leq k+1$ *and* $N(C) \subseteq N[v] \subseteq N[C]$ **do**
18 **if** $\mathcal{C}(R(C) \setminus N[v]) \subseteq \mathcal{B}$ **then**
19 $\mathcal{B} \leftarrow \mathcal{B} \cup \{C\}$
20 **for** $S \in \Delta(G)$ **do**
21 **if** $\mathcal{C}(G \setminus S) \subseteq \mathcal{B}$ **then**
22 **return** True
23 **return** False

The analysis of the algorithm focuses on transitions via PMCs of type 2 and proceeds by induction on the main loop invariant. The time complexity follows simply from fast subset convolution, the bound 2^{cc} on the number of blocks, and the fact that each iteration of the loop of the lines 9 to 15 takes polynomial time. The correctness is shown by combining the lemmas introduced in this section.

▶ **Lemma 29** (⋆). *There is an algorithm that given a graph G with an edge clique cover of size cc' determines the treewidth and minimum fill-in of G in time $O^*(2^{\mathsf{cc}'})$. Furthermore, if also a set $F \subseteq V(G)^2 \setminus E(G)$ is given the algorithm determines if there is a triangulation H of G with $E(H) \subseteq E(G) \cup F$, i.e., solves the chordal sandwich problem.*

6 Polynomial Space Algorithms

We give polynomial space algorithms for treewidth, weighted minimum fill-in, and fractional hypertreewidth. The algorithms are based on the following characterization of minimal triangulations.

▶ **Proposition 30** ([5]). *Let G be a graph, H a minimal triangulation of G, and Ω a maximal clique of H. For each $C_i \in \mathcal{C}(G \setminus \Omega)$ there exists a minimal triangulation H_i of $R(C_i)$ such that*

$$E(H) = \Omega^2 \cup \bigcup_{C_i \in \mathcal{C}(G \setminus \Omega)} E(H_i).$$

Note that by iterating over all $\Omega \in \Pi(G)$ in Proposition 30 we can indeed construct all minimal triangulations of G. Furthermore, all graphs H constructed in this manner are minimal triangulations [5].

The idea of the algorithm is to use the recursion of Proposition 30 directly, without dynamic programming. The following "balanced PMC" lemma guarantees that we can expect the size of an edge clique cover to roughly halve in each level of the recursion.

▶ **Lemma 31.** *Let G be a graph with an edge clique cover $\mathcal{W}$. Any minimal triangulation H of G has a maximal clique Ω so that all blocks $C \in \mathcal{C}(G \setminus \Omega)$ have $|\mathcal{W}[C]| \leq |\mathcal{W}|/2$.*

Proof. Note that for any PMC Ω there can be at most one component $C \in \mathcal{C}(G \setminus \Omega)$ so that $|\mathcal{W}[C]| > |\mathcal{W}|/2$ because the sets $\mathcal{W}[C_i]$ over $C_i \in \mathcal{C}(G \setminus \Omega)$ correspond to disjoint subsets of $\mathcal{W}$. Let H be any minimal triangulation of G and pick arbitrary maximal clique Ω of H. While there is a component $C \in \mathcal{C}(G \setminus \Omega)$ such that $|\mathcal{W}[C]| > |\mathcal{W}|/2$, pick a maximal clique Ω of H such that $N(C) \subseteq \Omega \subseteq N[C]$. If this process stops, we have found the desired maximal clique Ω. Suppose the process does not stop. It considers an infinite sequence of blocks $C_1, C_2, \ldots$ with an associated infinite sequence of PMCs $\Omega_1, \Omega_2, \ldots$ with $C_i \in \mathcal{C}(G \setminus \Omega_i)$. Consider two consecutive blocks C_i and C_{i+1} in this sequence such that C_{i+1} is not a subset of C_i, which exist because G is finite. Recall that $N(C_i) \subseteq \Omega_{i+1} \subseteq N[C_i]$. Because C_{i+1} is not a subset of C_i, we have that $N(C_{i+1}) \subseteq N(C_i)$, implying that C_i and C_{i+1} are two distinct components of $N(C_i)$. Therefore the sets $\mathcal{W}[C_i]$ and $\mathcal{W}[C_{i+1}]$ are disjoint, implying that either of them has to be of size at most $|\mathcal{W}|/2$, which is a contradiction. ◀

We combine Proposition 30 and Lemma 31 into the following lemma.

▶ **Lemma 32** ($\star$). *Let G be a graph and $\mathcal{W}$ an edge clique cover of G. A graph H is a minimal triangulation of G if and only if (1) $V(H) = V(G)$ and (2) there is a PMC $\Omega \in \Pi(G)$ with $|\mathcal{W}[C_i]| \leq |\mathcal{W}|/2$ for all $C_i \in \mathcal{C}(G \setminus \Omega)$ and*

$$E(H) = \Omega^2 \cup \bigcup_{C_i \in \mathcal{C}(G \setminus \Omega)} E(H_i),$$

where H_i is a minimal triangulation of $R(C_i)$.

Algorithm 2 presents a polynomial space $O^*(9^{\mathtt{cc}'})$ time algorithm for treewidth. The algorithms for other problems are similar. The algorithm implements the characterization of Lemma 32, with the observation that $\mathcal{W}[C] \cup \{N(C)\}$ is an edge clique cover of $R(C)$.

The time complexity analysis of the algorithm reduces to a recursion equation resembling $t(\mathtt{cc}') = 9^{\mathtt{cc}'} + 3^{\mathtt{cc}'} 2t(\mathtt{cc}'/2)$, with some polynomial factors that get cleaned up at the end by the $O^*(\cdot)$ notation.

▶ **Lemma 33** ($\star$). *There is an algorithm that given a graph G with an edge clique cover of size $\mathtt{cc}'$ determines the treewidth of G in polynomial space and $O^*(9^{\mathtt{cc}'})$ time. If also a weight function $w : V(G)^2 \to \mathbb{R}_{\geq 0}$ is given, the algorithm determines the weighted minimum fill-in of G with respect to w. There is also algorithm that given a hypergraph $\mathcal{G}$ with m hyperedges determines its fractional hypertreewidth in polynomial space and $O^*(9^m)$ time.*

Algorithm 2 Treewidth in polynomial space and $O^*(9^{\mathtt{cc}'})$ time.

Input : Connected graph G, an edge clique cover $\mathcal{W}$ of G, and an integer k
Output : Whether the treewidth of G is at most k

```
for Ω ∈ Π(G) do
    if |Ω| ≤ k+1 and |W[C_i]| ≤ |W|/2 for all C_i ∈ C(G \ Ω) then
        ok ← True
        for C ∈ C(G \ Ω) do
            if Treewidth(R(C), W[C] ∪ {N(C)}, k) = False then
                ok ← False
        if ok then return True
return False
```

The main idea to make the algorithm work when we do not know the edge clique cover is to just check if there are at most $3^{\mathtt{cc}}$ PMCs. The reason why the time complexity becomes a bit higher than in Lemma 33 is that we cannot assume that the worst case of branching from a PMC Ω results in only two subproblems. In particular, we cannot assume anything better than $\mathtt{cc}$ subproblems each with a minimum edge clique cover of size $\mathtt{cc}/2+1$.

▶ **Lemma 34** ($\star$). *There is an algorithm that given a graph G and an integer $\mathtt{cc}$ uses polynomial space and $O^*(9^{\mathtt{cc}+O(\log^2 \mathtt{cc})})$ time and returns an integer t that is at least the treewidth of G. If also a weight function $w : V(G)^2 \to \mathbb{R}_{\geq 0}$ is given, the algorithm also returns a number f that is at least the weighted minimum fill-in of G with respect to w. If $\mathtt{cc}$ is at least the size of a minimum edge clique cover of G, then t is the treewidth of G and f is the weighted minimum fill-in of G with respect to w.*

7 Tightness

We show that the bounds $O(2^{\mathtt{cc}})$ and $O(3^{\mathtt{cc}})$ for the numbers of minimal separators and PMCs are tight, and that the parameter edge clique cover cannot be relaxed to vertex clique cover.

Let us construct a graph $K_2^{\mathtt{cc}}$. Let $\mathcal{W}$ be a collection of size $\mathtt{cc}$ initially containing disjoint sets of size 1, i.e., $\mathcal{W} = \{W_1, \ldots, W_{\mathtt{cc}}\}$ with $W_i = \{v_i\}$. For each pair $1 \leq i < j \leq \mathtt{cc}$ we insert an element $v_{i,j}$ into the sets W_i and W_j. Now, the vertex set of $K_2^{\mathtt{cc}}$ is $V(K_2^{\mathtt{cc}}) = \bigcup_{W \in \mathcal{W}} W$ and the edge set of $K_2^{\mathtt{cc}}$ is $E(K_2^{\mathtt{cc}}) = \bigcup_{W \in \mathcal{W}} W^2$. The collection $\mathcal{W}$ is therefore an edge clique cover of $K_2^{\mathtt{cc}}$.

▶ **Lemma 35.** *The graph $K_2^{\mathtt{cc}}$ has $\Theta(2^{\mathtt{cc}})$ minimal separators and $\Theta(3^{\mathtt{cc}})$ PMCs.*

Proof. Upper bounds follow from Theorem 1. For distinct subsets $\mathcal{W}' \subseteq \mathcal{W}$ the vertex sets $V(K_2^{\mathtt{cc}}, \mathcal{W}')$ are distinct because they can be identified by the inclusion of the v_i vertices. Let $\mathcal{W}'$ be any non-empty strict subset of $\mathcal{W}$. Note that $K_2^{\mathtt{cc}}[V(K_2^{\mathtt{cc}}, \mathcal{W}')]$ is connected, and therefore also $K_2^{\mathtt{cc}}[V(K_2^{\mathtt{cc}}, \mathcal{W} \setminus \mathcal{W}')]$ is connected. Any vertex in $V(K_2^{\mathtt{cc}}) \setminus V(K_2^{\mathtt{cc}}, \mathcal{W}', \mathcal{W} \setminus \mathcal{W}')$ is of type $v_{i,j}$ and has a neighbor in both $V(K_2^{\mathtt{cc}}, \mathcal{W}')$ and $V(K_2^{\mathtt{cc}}, \mathcal{W} \setminus \mathcal{W}')$. Therefore, $V(K_2^{\mathtt{cc}}) \setminus V(K_2^{\mathtt{cc}}, \mathcal{W}', \mathcal{W} \setminus \mathcal{W}')$ is a minimal separator and $V(K_2^{\mathtt{cc}}, \mathcal{W}')$ and $V(K_2^{\mathtt{cc}}, \mathcal{W} \setminus \mathcal{W}')$ are blocks of it. Therefore, the number of minimal separators of $K_2^{\mathtt{cc}}$ is at least the number of bipartitions of $\mathcal{W}$, i.e., $\Omega(2^{\mathtt{cc}})$.

Take any tripartition $\{\mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_3\}$ of $\mathcal{W}$. Note that distinct tripartitions define distinct sets $V(K_2^{\mathtt{cc}}, \mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_3)$. We show that $\Omega = V(K_2^{\mathtt{cc}}) \setminus (K_2^{\mathtt{cc}}, \mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_3)$ is a PMC of $K_2^{\mathtt{cc}}$. Any vertex in Ω is of type $v_{i,j}$, with vertices v_i and v_j in different blocks $V(K_2^{\mathtt{cc}}, \mathcal{W}_p)$.

Therefore, for any pair of vertices in Ω there is a common block that they are adjacent to, and therefore Ω satisfies the cliquish condition. A component $V(K_2^{\mathtt{cc}}, \mathcal{W}_1)$ cannot be full because there is a vertex $v_{i,j}$ that intersects cliques $W_i \in \mathcal{W}_2$ and $W_j \in \mathcal{W}_3$ and therefore is in the PMC but not in the neighborhood of $V(K_2^{\mathtt{cc}}, \mathcal{W}_1)$. Therefore, Ω satisfies the no full component condition. Therefore, the number of PMCs of $K_2^{\mathtt{cc}}$ is at least the number of tripartitions of $\mathcal{W}$, i.e., $\Omega(3^{\mathtt{cc}})$. ◀

We use a simpler construction to show that there cannot be FPT, or even XP, bounds for minimal separators and potential maximal cliques parameterized by the size of a minimum vertex clique cover, i.e., the number of cliques to cover all vertices of a graph.

▶ **Lemma 36.** *A graph that is constructed as a disjoint union of two cliques of size $n/2$ connected by a matching of $n/2$ edges has $\Omega(2^{n/2})$ minimal separators.*

Proof. All ways of selecting one endpoint of each edge of the matching result in selecting a minimal separator, expect the two ways that select one of the cliques. ◀

Note that $|\Pi(G)| \geq |\Delta(G)|/n$ [6], so the graphs of Lemma 36 have also an exponential number of PMCs.

8 Conclusion

We bounded the number of minimal separators and PMCs by the size of a minimum edge clique cover, obtaining new FPT algorithms for problems in the PMC framework. The parameterization by edge clique cover is motivated by real applications of optimal triangulations, and our results provide theoretical corroboration on the observations of the efficiency of the PMC framework in practice. Prior to our work, only the recent paper of Fomin et al. [15] considers FPT bounds for PMCs. Our work answers to their proposal for finding further FPT parameterizations for PMCs.

We omitted polynomial factors in our analysis, but the author believes that the polynomial factors are reasonably low and all of the algorithms that we gave can be implemented in a practical manner when $\mathtt{cc}$ is small. We also remark that the techniques introduced in this paper can be used to obtain $O^*(2^{O(m)})$ time algorithms for generalized hypertreewidth via the results of [30]. We focused on fractional hypertreewidth because it is a more general parameter [18], and its computation is simpler in the sense that $\mathrm{FCOV}(\Omega)$ can be computed in polynomial time, unlike its counterpart in generalized hypertreewidth.

Our bounds and enumeration algorithms for minimal separators and PMCs are tight with respect to $\mathtt{cc}$ up to polynomial factors. Improving the algorithms for individual problems or proving (conditional) lower bounds remains as a problem for future work. Also, the question of finding further useful parameterizations of PMCs still remains for future work. Because of the naturality of $\mathtt{cc}$ in multiple settings related to optimal triangulations, we expect that more applications of our results could arise in future.

References

1 Anne Berry, Jean Paul Bordat, and Olivier Cogis. Generating all the minimal separators of a graph. *International Journal of Foundations of Computer Science*, 11(3):397–403, 2000. doi:10.1142/S0129054100000211.

2 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 67–74. ACM, 2007. doi:10.1145/1250790.1250801.

3 Hans L Bodlaender, Leizhen Cai, Jianer Chen, Michael R. Fellows, Jan Arne Telle, and Dániel Marx. Open problems in parameterized and exact computation – IWPEC 2006. Technical Report UU-CS-2006-052, Department of Information and Computing Sciences, Utrecht University, 2006. URL: https://dspace.library.uu.nl/handle/1874/22186.

4 Magnus Bordewich, Katharina T. Huber, and Charles Semple. Identifying phylogenetic trees. *Discrete Mathematics*, 300(1-3):30–43, 2005. doi:10.1016/j.disc.2005.06.015.

5 Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM Journal on Computing*, 31(1):212–232, 2001. doi:10.1137/S0097539799359683.

6 Vincent Bouchitté and Ioan Todinca. Listing all potential maximal cliques of a graph. *Theoretical Computer Science*, 276(1-2):17–32, 2002. doi:10.1016/S0304-3975(01)00007-X.

7 Mathieu Chapelle, Mathieu Liedloff, Ioan Todinca, and Yngve Villanger. Treewidth and pathwidth parameterized by the vertex cover number. *Discrete Applied Mathematics*, 216:114–129, 2017. doi:10.1016/j.dam.2014.12.012.

8 Cristina Conati, Abigail S. Gertner, Kurt VanLehn, and Marek J. Druzdzel. On-line student modeling for coached problem solving using Bayesian networks. In *User Modeling*, volume 383 of *CISM*, pages 231–242. Springer, 1997. doi:10.1007/978-3-7091-2670-7_24.

9 Marek Cygan, Marcin Pilipczuk, and Michal Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM Journal on Computing*, 45(1):67–83, 2016. doi:10.1137/130947076.

10 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 parameterized algorithms and computational experiments challenge: The second iteration. In *12th International Symposium on Parameterized and Exact Computation*, volume 89 of *LIPIcs*, pages 30:1–30:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.IPEC.2017.30.

11 Wolfgang Fischl, Georg Gottlob, Davide Mario Longo, and Reinhard Pichler. HyperBench: A benchmark and tool for hypergraphs and empirical findings. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 464–480, 2019. doi:10.1145/3294052.3319683.

12 Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. General and fractional hypertree decompositions: Hard and easy cases. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 17–32, 2018. doi:10.1145/3196959.3196962.

13 Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on H-graphs. *Algorithmica*, 2020. doi:10.1007/s00453-020-00692-9.

14 Fedor V. Fomin, Dieter Kratsch, Ioan Todinca, and Yngve Villanger. Exact algorithms for treewidth and minimum fill-in. *SIAM Journal on Computing*, 38(3):1058–1079, 2008. doi:10.1137/050643350.

15 Fedor V. Fomin, Mathieu Liedloff, Pedro Montealegre, and Ioan Todinca. Algorithms parameterized by vertex cover and modular width, through potential maximal cliques. *Algorithmica*, 80(4):1146–1169, 2018. doi:10.1007/s00453-017-0297-1.

16 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM Journal on Computing*, 44(1):54–87, 2015. doi:10.1137/140964801.

17 Masanobu Furuse and Koichi Yamazaki. A revisit of the scheme for computing treewidth and minimum fill-in. *Theoretical Computer Science*, 531:66–76, 2014. doi:10.1016/j.tcs.2014.03.013.

18 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms*, 11(1):4:1–4:20, 2014. doi:10.1145/2636918.

19 Rob Gysel. Minimal triangulation algorithms for perfect phylogeny problems. In *Language and Automata Theory and Applications - 8th International Conference*, volume 8370 of *LNCS*, pages 421–432. Springer, 2014. doi:10.1007/978-3-319-04921-2_34.

20 Masami Hasegawa, Hirohisa Kishino, and Taka-aki Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial dna. *Journal of Molecular Evolution*, 22(2):160–174, 1985.

21 Pinar Heggernes, Federico Mancini, Jesper Nederlof, and Yngve Villanger. A parameterized algorithm for chordal sandwich. In *Proceedings of 7th International Conference on Algorithms and Complexity*, volume 6078 of *LNCS*, pages 120–130. Springer, 2010. doi:10.1007/978-3-642-13073-1_12.

22 Finn V. Jensen and Frank Jensen. Optimal junction trees. In *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence*, pages 360–366. Morgan Kaufmann, 1994. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=524&proceeding_id=10.

23 Finn V. Jensen and Thomas D. Nielsen. *Bayesian networks and decision graphs*. Springer, 2007. doi:10.1007/978-0-387-68282-2.

24 Sampath Kannan and Tandy Warnow. A fast algorithm for the computation and enumeration of perfect phylogenies. *SIAM Journal on Computing*, 26(6):1749–1763, 1997. doi:10.1137/S0097539794279067.

25 Tuukka Korhonen. Finding optimal tree decompositions. Master's thesis, University of Helsinki, 2020. URL: http://urn.fi/URN:NBN:fi:hulib-202006173010.

26 Tuukka Korhonen, Jeremias Berg, and Matti Järvisalo. Solving graph problems via potential maximal cliques: An experimental evaluation of the Bouchitté-Todinca algorithm. *ACM Journal of Experimental Algorithmics*, 24(1):1.9:1–1.9:19, 2019. doi:10.1145/3301297.

27 Tuukka Korhonen and Matti Järvisalo. Finding most compatible phylogenetic trees over multi-state characters. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, pages 1544–1551. AAAI Press, 2020. doi:10.1609/aaai.v34i02.5514.

28 Mathieu Liedloff, Pedro Montealegre, and Ioan Todinca. Beyond classes of graphs with "few" minimal separators: FPT results through potential maximal cliques. *Algorithmica*, 81(3):986–1005, 2019. doi:10.1007/s00453-018-0453-2.

29 Daniel Lokshtanov. On the complexity of computing treelength. *Discrete Applied Mathematics*, 158(7):820–827, 2010. doi:10.1016/j.dam.2009.10.007.

30 Lukas Moll, Siamak Tazari, and Marc Thurley. Computing hypergraph width measures exactly. *Information Processing Letters*, 112(6):238–242, 2012. doi:10.1016/j.ipl.2011.12.002.

31 Noam Ravid, Dori Medini, and Benny Kimelfeld. Ranked enumeration of minimal triangulations. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 74–88. ACM, 2019. doi:10.1145/3294052.3319678.

32 Don Ringe, Tandy Warnow, and Ann Taylor. Indo-european and computational cladistics. *Transactions of the Philological Society*, 100(1):59–129, 2002. doi:10.1111/1467-968X.00091.

33 Marco Scutari. Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3):1–22, 2010. doi:10.18637/jss.v035.i03.

34 Charles Semple and Mike Steel. *Phylogenetics*. Oxford University Press, 2003.

35 Ken Takata. Space-optimal, backtracking algorithms to list the minimal vertex separators of a graph. *Discrete Applied Mathematics*, 158(15):1660–1667, 2010. doi:10.1016/j.dam.2010.05.013.

36 Hisao Tamaki. Computing treewidth via exact and heuristic lists of minimal separators. In *Proceedings of the Special Event on Analysis of Experimental Algorithms*, volume 11544 of *LNCS*, pages 219–236. Springer, 2019. doi:10.1007/978-3-030-34029-2_15.

37 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *Journal of Combinatorial Optimization*, 37(4):1283–1311, 2019. doi:10.1007/s10878-018-0353-z.

The Asymmetric Travelling Salesman Problem In Sparse Digraphs

Łukasz Kowalik
Institute of Informatics, University of Warsaw, Poland
kowalik@mimuw.edu.pl

Konrad Majewski
Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw, Poland
km371194@students.mimuw.edu.pl

Abstract

Asymmetric Travelling Salesman Problem (ATSP) and its special case Directed Hamiltonicity are among the most fundamental problems in computer science. The dynamic programming algorithm running in time $\mathcal{O}^*(2^n)$ developed almost 60 years ago by Bellman, Held and Karp, is still the state of the art for both of these problems.

In this work we focus on *sparse* digraphs.

First, we recall known approaches for Undirected Hamiltonicity and TSP in sparse graphs and we analyse their consequences for Directed Hamiltonicity and ATSP in sparse digraphs, either by adapting the algorithm, or by using reductions. In this way, we get a number of running time upper bounds for a few classes of sparse digraphs, including $\mathcal{O}^*(2^{n/3})$ for digraphs with both out- and indegree bounded by 2, and $\mathcal{O}^*(3^{n/2})$ for digraphs with outdegree bounded by 3.

Our main results are focused on digraphs of bounded *average* outdegree d. The baseline for ATSP here is a simple enumeration of cycle covers which can be done in time bounded by $\mathcal{O}^*(\mu(d)^n)$ for a function $\mu(d) \leq (\lceil d \rceil !)^{1/\lceil d \rceil}$. One can also observe that Directed Hamiltonicity can be solved in randomized time $\mathcal{O}^*((2-2^{-d})^n)$ and polynomial space, by adapting a recent result of Björklund [ISAAC 2018] stated originally for Undirected Hamiltonicity in sparse bipartite graphs. We present two new deterministic algorithms for ATSP: the first running in time $\mathcal{O}(2^{0.441(d-1)n})$ and polynomial space, and the second in exponential space with running time of $\mathcal{O}^*(\tau(d)^{n/2})$ for a function $\tau(d) \leq d$.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases asymmetric traveling salesman problem, Hamiltonian cycle, sparse graphs, exponential algorithm

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.23

Related Version A full version of the paper is available at [27], `https://arxiv.org/abs/2007.12120`.

Funding *Łukasz Kowalik*: Supported by ERC Starting Grant TOTAL (Grant Agreement No 677651).

Acknowledgements The authors thank the reviewers for careful reading and useful comments.

1 Introduction

In the Directed Hamiltonicity problem, given a directed graph (digraph) G one has to decide if G has a Hamiltonian cycle, i.e., a simple cycle that visits all vertices. In its weighted version, called ATSP, we additionally have integer weights on edges $w : E \to \mathbb{Z}$, and the goal is to find a minimum weight Hamiltonian cycle in G.

The ATSP problem has a dynamic programming algorithm running in time and space $\mathcal{O}^*(2^n)$ due to Bellman [2] and Held and Karp [23]. Gurevich and Shelah [22] obtained the best known *polynomial space* algorithm, running in time $\mathcal{O}(4^n n^{\log n})$. It is a major

open problem whether there is an algorithm in time $\mathcal{O}^*((2-\varepsilon)^n)$ for an $\varepsilon > 0$, even for the unweighted case of Directed Hamiltonicity. However, there has been a significant progress in answering this question in variants of Directed Hamiltonicity. Namely, Björklund and Husfeldt [6] showed that the parity of the number of Hamiltonian cycles in a digraph can be determined in time $O(1.619^n)$ and Cygan, Kratsch and Nederlof [13] solved the bipartite case of Directed Hamiltonicity in time $\mathcal{O}(1.888^n)$, which was later improved to $\mathcal{O}^*(3^{n/2}) = O(1.74^n)$ by Björklund, Kaski and Koutis [9].

Table 1 Running times (with polynomial factors omitted) of algorithms for *undirected* graphs. Rows marked with 💣 denote exponential space algorithms, rows marked with ⚄ denote Monte Carlo algorithms.

Graph class	Undirected Hamiltonicity		Travelling Salesman Problem	
general	1.66^n ⚄	[3]	2^n 💣 $4^n n^{\log n}$	[2, 23] [22]
bipartite	1.42^n ⚄	[3]	2^n 💣 4^n	[2, 23] [29]
$\Delta = 3$	1.16^n 💣⚄ 1.24^n	[13] [33]	1.22^n 💣 1.24^n	[11] [33]
$\Delta = 4$	1.51^n 💣⚄ 1.59^n ⚄	[13]+[17] [8]	1.63^n 💣 1.70^n	[11]+[17] [34]
$\Delta = 5$	1.63^n ⚄	[8]	1.88^n 💣 2.35^n	[11]+[17] [35]
any Δ			$(2-\varepsilon'_\Delta)^n$ 💣	[7]
avgdeg $\leq d$	1.12^{dn} 💣⚄	[13]+[25]	1.14^{dn} 💣 $2^{(1-\varepsilon_d)n}$ 💣	[11]+[25] [15]
bipartite avgdeg $\leq d$	$(2-2^{1-d})^{n/2}$ ⚄	[4]		
pathwidth	$3.42^{\mathtt{pw}}$ 💣⚄	[13]	$4.28^{\mathtt{pw}}$ 💣	[11]
treewidth	$4^{\mathtt{tw}}$ 💣⚄	[14]	$9.56^{\mathtt{tw}}$ 💣	[11]

Undirected graphs. Even more is known in the undirected setting, where the problems are called Undirected Hamiltonicity and TSP. Björklund [3] shows that Undirected Hamiltonicity can be solved in time $\mathcal{O}(1.66^n)$ in general and $\mathcal{O}^*(2^{n/2}) = O(1.42^n)$ in the bipartite case. Very recently, Nederlof [28] showed that the bipartite case of TSP admits an algorithm in time $\mathcal{O}(1.9999^n)$, assuming that square matrices can be multiplied in time $O(n^{2+o(1)})$. Finally, there is a number of results for Undirected Hamiltonicity and TSP restricted to graphs that are somewhat sparse. An early example is an algorithm of Eppstein [16] for TSP in graphs of maximum degree 3, running in time $\mathcal{O}^*(2^{n/3}) = O(1.26^n)$. This result has been later improved and generalized to larger values of maximum degree, we refer the reader to Table 1 for details (Δ denotes the maximum degree). Perhaps the most general measure of graph sparsity is the average degree d. Cygan and Pilipczuk [15] showed that whenever d is bounded, the 2^n barrier for TSP can be broken, although only slightly. More precisely, they proved the bound $\mathcal{O}^*(2^{(1-\varepsilon_d)n})$, where $\varepsilon_d = 1/(2^{2d+1} \cdot 20d \cdot e^{e^{20d}})$. We

note that although their result was stated for undirected graphs, the same reasoning can be made for digraphs of average *total degree* (sum of indegree and outdegree). For small values of d, more significant improvements are possible. Namely, by combining the algorithms for Undirected Hamiltonicity and TSP parameterized by pathwidth [11, 13] with a bound on pathwidth of sparse graphs [25] we get the upper bound of $\mathcal{O}(1.12^{dn})$ and $\mathcal{O}(1.14^{dn})$, respectively. For Undirected Hamiltonicity, if the input graph is additionally bipartite, Björklund [4] shows the $\mathcal{O}^*((2-2^{1-d})^{n/2})$ upper bound.

Table 2 Running times (with polynomial factors omitted) of the algorithms for *directed* graphs. We preserve the notation from Table 1. By Δ^+ we denote maximum outdegree and Δ denotes maximum total degree. Treewidth refers to the underlying undirected graph.

Graph class	Directed Hamiltonicity		Asymmetric Travelling Salesman Problem	
general	2^n	[1, 24, 26]	2^n $4^n n^{\log n}$	[2, 23] [22]
bipartite	1.74^n	[9]	2^n 4^n	[2, 23] [29]
$(2,2)$-graphs	1.26^n	(Corollary 2.6)	1.26^n	(Corollary 2.6)
$\Delta^+ = 3$	1.74^n	(Corollary 2.8)	1.74^n	(Corollary 2.8)
$\Delta = 3$	1.13^n	(Corollary 2.7)	1.13^n	(Corollary 2.7)
any Δ	$(2-2^{-\Delta/2})^n$	(Theorem 2.10)	$(2-\varepsilon'_\Delta)^n$	[7]
average outdeg $\leq d$	$\mu(d)^n$ $2^{0.441(d-1)n}$ $\sqrt{\tau(d)}^{\,n}$ $(2-2^{-d})^n$ $2^{(1-\Omega(1/d))n}$	(Corollary 2.4) (Theorem 1.1) (Theorem 1.2) (Theorem 2.10) [5]	$\mu(d)^n$ $2^{0.441(d-1)n}$ $\sqrt{\tau(d)}^{\,n}$ $2^{(1-\varepsilon_{2d})n}$	(Corollary 2.4) (Theorem 1.1) (Theorem 1.2) [15]
treewidth	$6^{\mathtt{tw}}$	[14]		

Directed sparse graphs: hidden results. The goal of this paper is to investigate Directed Hamiltonicity and ATSP in *sparse directed* graphs. Quite surprisingly, not much results in this topic are stated explicitly. In fact, we were able to find just a few references of this kind: Björklund, Husfeldt, Kaski and Koivisto [7] describe an algorithm for digraphs with total degree bounded by D that works in time $\mathcal{O}^*((2-\varepsilon'_D)^n)$, for $\varepsilon'_D = 2-(2^{D+1}-2D-2)^{1/(D+1)}$. Second, Cygan et al. [14] describe an algorithm for Directed Hamiltonicity running in time $6^t n^{O(1)}$, where t is the treewidth of the input graph. Finally, Björklund and Williams [10] show a deterministic algorithm which *counts* Hamiltonian cycles in directed graphs of average degree d in time $2^{n-\Omega(n/d)}$ and exponential space. Very recently, Björklund [5], using a different approach, obtained the same running time for the *decision* Directed Hamiltonicity problem, but lowering the space to polynomial, at the cost of using randomization. The authors of these two works have not put an effort to optimize the constants hidden in the Ω notation. By following the analysis in each of these papers as-is, we get the saving term in the exponent at least $n/(111d)$ (for a faster, randomized algorithm) and $n/(500d)$, respectively.

However, one cannot say that nothing more is known, because many results for undirected graphs imply some running time bounds in the directed setting. We devote the first part of this work to investigating such implications. In some cases, the implications are immediate.

For example, Gebauer [20, 21] shows an algorithm running in time $\mathcal{O}^*(3^{n/2}) = \mathcal{O}^*(1.74^n)$ that solves TSP in graphs of maximum degree 4. It uses the meet-in-the-middle approach and can be sketched as follows: guess two opposite vertices of the solution cycle, generate a family of paths of length $n/2$ from each of them (of size at most $3^{n/2}$) and store one of the families in a dictionary to enumerate all complementary pairs of paths in time $\mathcal{O}^*(3^{n/2})$. This algorithm, without a change, can be used for ATSP in digraphs of maximum outdegree 3, with the same running time bound (see Theorem 2.8).

The other implications that we found rely on a simple reduction from ATSP to a variant of TSP in *bipartite undirected* graphs (see Lemma 2.1): replace each vertex v of the input digraph G by two vertices v^{out}, v^{in} joined by an edge of weight 0, and for each edge $(u, v) \in E(G)$ create an edge $u^{\mathsf{out}}v^{\mathsf{in}}$ of the same weight. Then find a lightest Hamiltonian cycle that contains the matching $M = \{v^{\mathsf{out}}v^{\mathsf{in}} \mid v \in V(G)\}$. By applying this reduction to a digraph with both outdegrees and indegrees bounded by 2, which we call a $(2, 2)$-graph, and using Eppstein's algorithm [16] we get the running time of $\mathcal{O}^*(2^{n/3}) = \mathcal{O}^*(1.26^n)$, see Corollary 2.6. Another consequence is an algorithm running in time $\mathcal{O}^*(2^{n/6})$ for digraphs of maximum total degree 3, see Corollary 2.7. These two simple classes of digraphs were studied by Plesník [30], who showed that Directed Hamiltonicity remains NP-complete when restricted to them.

We can also apply the reduction to an arbitrary digraph of average outdegree d. A naive approach would be then to enumerate all perfect matchings in the bipartite graph induced by edges $\{u^{\mathsf{out}}v^{\mathsf{in}} \mid (u, v) \in E(G)\}$. Indeed, each such matching corresponds to a cycle cover in the input graph, so we basically enumerate cycle covers and filter-out the disconnected ones. Thanks to Bregman-Minc inequality [12] which bounds the permament in sparse matrices the resulting algorithm has running time $\mathcal{O}^*(\mu(d)^n)$, where

$$\mu(d) = (\lfloor d \rfloor !)^{\frac{\lfloor d \rfloor + 1 - d}{\lfloor d \rfloor}} (\lceil d \rceil !)^{\frac{d - \lfloor d \rfloor}{\lceil d \rceil}} \leq (\lceil d \rceil !)^{1/\lceil d \rceil}.$$

See Corollary 2.4 for details.

Yet another upper bound for digraphs of average outdegree d is obtained by using the reduction described above and next applying Björklund's algorithm for sparse bipartite graphs [4] with a slight modification to force the matching M in the Hamiltonian cycle (see Theorem 2.10). The resulting algorithm has running time $\mathcal{O}^*((2 - 2^{-d})^n)$.

Directed sparse graphs: main results. The simple consequences that we describe above are complemented by two more technical results.

The first algorithm runs in polynomial space and realizes the following idea. Assume $d < 3$. Then many of the vertices of the input graph have outdegree at most 2, and we can just branch on vertices of outdegree at least 3, and solve the resulting $(2, 2)$-graph using the fast $\mathcal{O}^*(2^{n/3})$-time algorithm mentioned before. This idea can be boosted a bit in the case when the initial branching is too costly, i.e., there are many vertices of high outdegree: then we observe that in such an unbalanced graph one can apply the simple cycle cover enumeration which then runs faster than in graphs of the same density but with balanced outdegrees. After a technical analysis of the running time we get the following theorem.

▶ **Theorem 1.1.** ATSP *restricted to digraphs of average outdegree at most d can be solved in time $\mathcal{O}^*(2^{\alpha(d-1)n})$ and polynomial space, where $\alpha = \frac{7}{12} - \frac{1}{12(\log_2 3 - 1)} < 0.44088$.*

The second algorithm generalizes Gebauer's meet-in-the-middle approach to digraphs of average outdegree d. (We note that it uses exponential space.)

▶ **Theorem 1.2.** ATSP *restricted to digraphs of average outdegree at most d can be solved in time $\mathcal{O}^*(\tau(d)^{n/2})$ and the same space, where*

$$\tau(d) = \lfloor d \rfloor^{\lfloor d \rfloor + 1 - d} (\lfloor d \rfloor + 1)^{d - \lfloor d \rfloor} \leq d$$

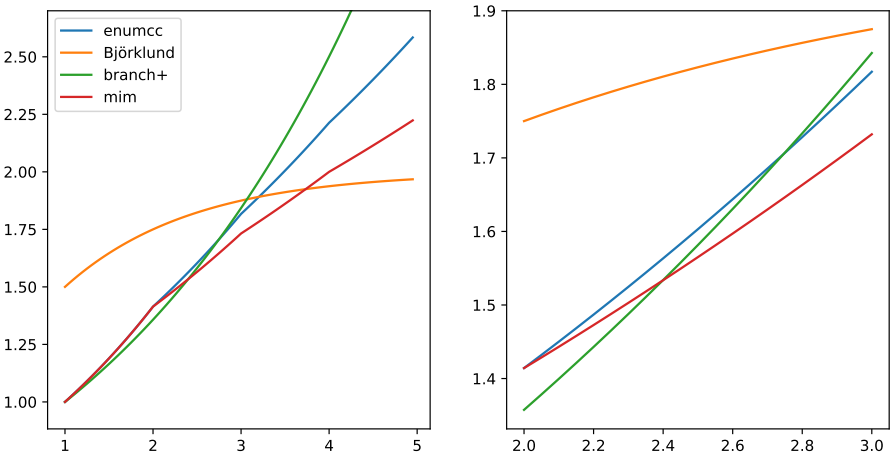


Figure 1 Comparison of the running times of algorithms for solving ATSP (enumcc, branch+, mim) and Directed Hamiltonicity (Björklund) in sparse digraphs. Horizontal axis: average degree d, vertical axis: base b from the running time bound of the form $\mathcal{O}^*(b^n)$.

Which algorithm is the best? Figure 1 compares four algorithms for solving ATSP and Directed Hamiltonicity in digraphs of average outdegree d described above:

- enumcc: enumerating cycle covers (Corollary 2.4),
- Björklund: adaptation of Björklund's bipartite graphs algorithm (Theorem 2.10),
- branch+: branching boosted by enumerating cycle covers (Theorem 1.1).
- mim: meet in the middle (Theorem 1.2),

The choice of the best (in terms of the asymptotic worst-case running time) algorithm depends on d, on whether we can afford exponential space, and on whether we solve ATSP or just Directed Hamiltonicity. We can conclude the following.

- **ATSP in polynomial space:** for $d < 2.746$ use branch+, for $d \in [2.746, 8.627]$ use enumcc, and for $d > 8.627$ use the general algorithm of Gurevich and Shelah [22].
- **ATSP in exponential space:** for $d < 2.398$ use branch+, for $d \in [2.398, 3.999]$ use mim, and for $d > 3.999$ use the algorithm of Cygan and Pilipczuk [15].
- **Directed Hamiltonicity in polynomial space:** for $d < 2.746$ use branch+, for $d \in [2.746, 3.203]$ use enumcc, for $d > 3.203$ use Björklund, and for sufficiently large d use the algorithm of Björklund [5].
- **Directed Hamiltonicity in exponential space:** for $d < 2.398$ use branch+, for $d \in [2.398, 3.734]$ use mim, for $d > 3.734$ use Björklund, and for sufficiently large d use the algorithm of Björklund and Williams [10].

2 Reductions from undirected graphs

The objective of this section is to recall two reductions from the ATSP to the (forced) TSP. Then, we will discuss existing methods of solving Undirected Hamiltonicity and TSP, and present their implications for corresponding problems in directed graphs. The summary of this section is presented in Tables 1 and 2.

2.1 General reductions

We recall that in the Forced Travelling Salesman Problem [16, 31, 33, 34], we are given an undirected graph G, a weight function $w : E(G) \to \mathbb{Z}$, and a subset $F \subseteq E(G)$. We say that a Hamiltonian cycle H is *admissible*, if $F \subseteq H$. The goal is to find an admissible Hamiltonian cycle of the minimum total weight of the edges (or, report that there is no such cycle). Moreover, we define the Bipartite Forced Matching TSP (BFM-TSP) as a special case of the Forced TSP, where graph G is bipartite, and the edges of F form a perfect matching in G.

The following lemma provides the relationship between the BFM-TSP and the ATSP.

▶ **Lemma 2.1.** *For every instance (G, w) of* ATSP*, where G is a digraph on n vertices, there is an equivalent instance $(\widehat{G}, \widehat{w}, M)$ of* BFM-TSP *such that $\widehat{G}$ is a graph on $2n$ vertices.*

Moreover, if both outdegrees and indegrees of G are bounded by D, then $\widehat{G}$ has maximum degree D. Similarly, if G has average outdegree d, then $\widehat{G}$ has average degree $d + 1$.

Proof. The proof is based on the folklore reduction from ATSP to TSP. Let (G, w) be an instance of ATSP. Let $V^{\mathsf{out}} = \{v^{\mathsf{out}} \mid v \in V(G)\}$ and $V^{\mathsf{in}} = \{v^{\mathsf{in}} \mid v \in V(G)\}$. We define $\widehat{G}$ as a bipartite graph on the vertex set $V(\widehat{G}) = V^{\mathsf{out}} \cup V^{\mathsf{in}}$ with edges $E(\widehat{G}) = \{u^{\mathsf{out}}v^{\mathsf{in}} \mid (u, v) \in E(G)\} \cup M$, where M is the perfect matching $M = \{v^{\mathsf{in}}v^{\mathsf{out}} \mid v \in V(G)\}$. The edges of $E(\widehat{G}) \setminus M$ inherit the weight from G, i.e. for $(u, v) \in E(G)$ we set $\widehat{w}(u^{\mathsf{out}}v^{\mathsf{in}}) = w(uv)$. Edges of M have weight 0. It is easy to see that the instance has the desired properties (deferred to the full version due to space constraints). ◀

Lemma 2.1 implies, in particular, that if there is an algorithm for BFM-TSP running in time $\mathcal{O}^*(f(n))$, then there is an algorithm for ATSP running in time $\mathcal{O}^*(f(2n))$.

2.2 Enumerating cycle covers

Let $(\widehat{G}, \widehat{w}, M)$ be an instance of BFM-TSP, and let $\mathcal{M}$ be a family of all perfect matchings in $\widehat{G} - M$. We observe that every cycle cover in $\widehat{G}$ which contains all edges of M is of the form $M \cup M'$, where $M' \in \mathcal{M}$. Hence, our goal is to find a matching $M' \in \mathcal{M}$ such that $M \cup M'$ is a Hamiltonian cycle in $\widehat{G}$, and the weight of M' is minimum possible. One way to do it is to list all the perfect matchings $M' \in \mathcal{M}$, and choose the best one among these which form with M a Hamiltonian cycle in $\widehat{G}$. We will investigate the complexity of such an approach in sparse graphs.

It is known that all perfect matchings in bipartite graph $\widehat{G}$ can be listed in time $|\mathcal{M}|n^{\mathcal{O}(1)}$ and polynomial space [18]. Hence, it is enough to provide a bound on the size of $\mathcal{M}$ in sparse graphs. We start with recalling a classic result of Bregman.

▶ **Theorem 2.2** (Bregman-Minc inequality [12, 32])**.** *Let A be an $n \times n$ binary matrix, and let r_i denote the number of ones in the i-th row. Then*

$$\operatorname{per} A \leq \prod_{i=1}^{n} (r_i!)^{1/r_i}.$$

▶ **Corollary 2.3.** ATSP *restricted to digraphs of outdegree bounded by D can be solved in time $(D!)^{n/D}n^{\mathcal{O}(1)}$ and polynomial space.*

Proof. Given an instance (G, w) of ATSP, we use Lemma 2.1 to obtain an equivalent instance $(\widehat{G}, \widehat{w}, M)$ of BFM-TSP. Then, $H := \widehat{G} - M$ is a bipartite graph on $V^{\mathsf{out}} \cup V^{\mathsf{in}}$, and all vertices of V^{out} in H have degree at most D. Since the number of perfect matchings coincides with the permanent of the adjacency matrix, and vertex degrees correspond to the number of ones in the corresponding rows, by Theorem 2.2, there are at most $(D!)^{n/D}$ perfect matchings in H. Hence, according to our initial observation, the instance $(\widehat{G}, \widehat{w}, M)$ can be solved in time $(D!)^{n/D}n^{\mathcal{O}(1)}$. ◀

To the best of our knowledge, Corollary 2.3 provides the fastest polynomial space algorithm for $D \in \{3, 4, \ldots, 8\}$. The Bregman-Minc inequality is also useful for digraphs with bounded average outdegree.

▶ **Corollary 2.4 (♠[1]).** ATSP *restricted to digraphs of average outdegree d can be solved in time $\mu(d)^n n^{\mathcal{O}(1)}$ and polynomial space, where*

$$\mu(d) = (\lfloor d \rfloor !)^{\frac{\lfloor d \rfloor + 1 - d}{\lfloor d \rfloor}} (\lceil d \rceil !)^{\frac{d - \lfloor d \rfloor}{\lceil d \rceil}}$$

In particular, for integral values of d, the running time is bounded by $(d!)^{n/d}n^{\mathcal{O}(1)}$.

2.3 Branching algorithms

One of the most common techniques which is used for solving NP-hard problems in sparse graphs is branching (bounded search trees). It is based on optimizing exhaustive search algorithms by bounding the size of the recursion tree. In case of TSP, the first result of this kind is due to Eppstein [16], and can be stated as follows.

▶ **Theorem 2.5** ([16])**.** Forced TSP *restricted to subcubic graphs can be solved in time $2^{(n-|F|)/3}n^{\mathcal{O}(1)}$ and polynomial space.*

▶ **Corollary 2.6.** ATSP *restricted to digraphs with all out- and indegrees at most 2 can be solved in time $\mathcal{O}^*(2^{n/3})$ and polynomial space.*

Proof. Let (G, w) be an instance of ATSP, where G is a digraph with all out- and indegrees at most 2. We apply Lemma 2.1 to obtain an equivalent instance $(\widehat{G}, \widehat{w}, M)$ of BFM-TSP. We know that $\widehat{G}$ has $2n$ vertices, and is subcubic. Moreover, $(\widehat{G}, \widehat{w}, M)$ is an instance of Forced TSP with $|M| = n$ forced edges. Hence, we can use Theorem 2.5 to solve it in time $\mathcal{O}^*(2^{(2n-n)/3}) = \mathcal{O}^*(2^{n/3})$. ◀

By combining a simple reduction implicit in a paper of Plesník [30] with Corollary 2.6 we obtain the following.

▶ **Corollary 2.7 (♠).** ATSP *restricted to digraphs of maximum total degree 3 can be solved in time $\mathcal{O}^*(2^{n/6})$ and polynomial space.*

[1] Proofs marked by ♠ are omitted due to space constraints and can be found in the full version of the paper [27].

2.4 Meet in the middle technique

Gebauer [20] shows an algorithm for undirected graphs of maximum degree 4 by using so-called meet in the middle technique, which can be easily applied in digraphs with outdegrees bounded by D to get the following result.

▶ **Theorem 2.8** ([20]). ATSP *restricted to digraphs with outdegrees bounded by D can be solved in time $\mathcal{O}^*(D^{n/2})$ and exponential space.*

2.5 Algebraic methods

Björklund [4] shows the following result.

▶ **Theorem 2.9** ([4]). *There is a Monte Carlo algorithm which solves* Undirected Hamiltonicity *restricted to bipartite graphs of average degree at most d in time $\mathcal{O}^*((2-2^{1-d})^{n/2})$ and polynomial space.*

It turns out that the proof of Theorem 2.9 can be modified to get the following Theorem. The idea is to use the reduction of Lemma 2.1 to get a sparse bipartite graph and modify the construction of Theorem 2.9 so that a relevant forced matching is a part of the resulting Hamiltonian cycle.

▶ **Theorem 2.10.** *There is a Monte Carlo algorithm which solves* Directed Hamiltonicity *restricted to digraphs of average outdegree at most d in time $\mathcal{O}^*((2-2^{-d})^n)$ and polynomial space.*

Proof. We assume that the reader is familiar with the proof of Theorem 2.9. We apply Lemma 2.1 and we get a bipartite undirected graph $\widehat{G} = (I \cup J, \widehat{E})$ and a perfect matching $F \subseteq \widehat{E}$. Recall that $\widehat{G}$ has $2n$ vertices and average degree at most $d+1$. The goal is to decide whether $\widehat{G}$ has a Hamiltonian cycle H that contains F.

Similarly as in [4] we define a polynomial matrix M with rows indexed by the vertices of I, and columns indexed by the vertices of J, as follows.

$$M(a,x,z)_{i,j} = \begin{cases} \sum_{k \in I \setminus \{i\}} z_{i,j} z_{j,k}(a_{j,k} + x_k) & \text{when } ij \in F, \\ z_{i,j} z_{j,k}(a_{j,k} + x_k) & \text{when } ij \notin F, \text{ but } jk \in F. \end{cases}$$

These polynomials have three types of variables: x_i for every $i \in I$, $a_{j,i}$ for every edge $ji \in \widehat{E}$, $j \in J$, $i \in I$. The third type of variable is somewhat special. Pick a fixed edge $e^* = i^*j^* \in F$. For every edge $ij \in \widehat{E} \setminus \{e^*\}$ there is one variable with two names $z_{i,j}$ and $z_{j,i}$; there are also two different variables z_{i^*,j^*} and z_{j^*,i^*}. Then we define a polynomial over a large enough field of characteristic two:

$$\phi = \sum_{x \in \{0,1\}^{n/2}} \det(M(a,x,z))$$

Now we should prove that thanks to cancellation in a field of characteristic two, $\phi = \sum_{H \in \mathcal{H}} \prod_{ij \in H} z_{i,j}$, where $\mathcal{H}$ is the set of all Hamiltonian cycles in $\widehat{G}$ which contain F. Björklund (Lemma 3 in [4]) shows this equality for the original polynomial using three observations: 1) after cancellation, the surviving terms do not contain a-variables, 2) each surviving term corresponds to a unique cycle cover in the graph, and 3) terms corresponding to non-Hamiltonian cycle covers pair-up and cancel-out, because if we reverse the lexicographically first cycle that does not contain e^*, then we get exactly the same term (and if we reverse a Hamiltonian cycle we get a different term, because of the asymmetry in defining z variables). The arguments used in [4] for proving 1)-3) still hold for the new polynomial, essentially for the same reasons.

The second ingredient of Björklund's construction is an upper bound on probability that none of the columns of $M(a,x,z)$ is identically zero, where $x \in \{0,1\}^{n/2}$ is a fixed assignment, z is the vector of all $z_{i,j}$ variables, and $a \in \{0,1\}^{n/2}$ is a *random* assignment. The calculation relies on the observation that if for a vertex $j \in J$ we have $a_{j,i} + x_i \equiv 0 \pmod 2$ for all $ij \in \widehat{E}$, then the column of j is identically zero. Note that this observation still holds for our new design. It follows that the probability bounds derived in [4] apply also in our case.

The third ingredient is efficient identification of assignments $x \in \{0,1\}^{n/2}$, for which $\det(M(a,x,z))$ is non-zero (for fixed, random, values of a). This is done by creating a Boolean variable w_v corresponding to every variable x_v and building a CNF formula such that its satisfying assignments correspond to a superset of all assignments of x_v variables that result in non-zero $\det(M(a,x,z))$. Again, the fact that the resulting formula is in CNF follows from the fact that the j-th column is non-zero if for some $i \in I$ we have $a_{j,i} + x_i \equiv 1 \pmod 2$, which is also true in our design. Finally, Björklund [4] shows how to enumarate all satisfying assignments of the CNF formula efficiently, what is not altered in any way by our changes in the design of polynomial ϕ. ◀

3 Polynomial space algorithm

This section is devoted to the proof of Theorem 1.1. We begin with introducing some additional notions, then we provide a branching algorithm which will be later used as a subroutine, and finally we describe and analyse an algorithm for digraphs of average outdegree at most d.

3.1 Preliminaries

Interfaces and switching walks. Let G be a directed graph (digraph). For a vertex v a set I_v^{in} of all incoming edges to v or a set I_v^{out} of all outgoing edges from v will be called an *interface* of v. We define the type of an interface of v so that $\mathsf{type}(I_v^{\mathsf{in}}) = \mathsf{in}$ and $\mathsf{type}(I_v^{\mathsf{out}}) = \mathsf{out}$.

Consider a sequence of distinct edges $\pi = e_1, \ldots, e_k$ in G such that if we forget about the orientation of edges, then we get a walk $v_1, \ldots, v_{k+1}$ in the underlying undirected graph, where for $i = 1, \ldots, k$ edge e_i is an orientation of $v_i v_{i+1}$. Assume additionally that for every $i = 2, \ldots, k$ either both edges e_{i-1} and e_i enter v_i or both leave v_i, in other words, the orientation of edges on the walk alternates. Now, let $I_1, \ldots, I_{k+1}$ be the consecutive interfaces visited by π, i.e., for every $j = 1, \ldots, k+1$ we have that I_j is an interface of v_j and for every $j = 1, \ldots, k$, we have $e_j \in I_j \cap I_{j+1}$. If $|I_1|, |I_k| > 2$ and $|I_j| = 2$, for $j = 2, \ldots, k-1$, the sequence π will be called a *switching walk*. Similarly, if $|I_j| = 2$ for $j = 1, \ldots, k$, and $v_1 = v_{k+1}$, i.e., the walk $v_1, \ldots, v_{k+1}$ is closed, then π will be called a *switching circuit*. In both cases, *length* of π is defined as k. The sequence $v_1, \ldots, v_{k+1}$ is called the *vertex sequence* of π. Abusing the notation slightly, we will refer to π as a set, when it is convenient. The motivation for introducing the notions of switching walks and circuits is given by the following lemma.

▶ **Lemma 3.1.** *Let $\pi = \{e_1, \ldots, e_k\}$ be a switching walk or a switching circuit in a digraph G. Let $H \subseteq E(G)$ be a Hamiltonian cycle in G. Then, $H \cap \pi = \{e_{2i-1} \mid i = 1, \ldots, \lfloor \frac{k+1}{2} \rfloor\}$, or $H \cap \pi = \{e_{2i} \mid i = 1, \ldots, \lfloor \frac{k}{2} \rfloor\}$.*

Proof. Let us assume that π is a switching walk. (For a switching circuit the proof is analogous.) Consider two consecutive edges $e_i, e_{i+1} \in \pi$. By the definition of a switching walk, there is a vertex v with an interface I of size 2 such that $I = \{e_i, e_{i+1}\}$. Since the cycle H passes through v, we obtain that H must contain exactly one of the edges e_i and e_{i+1}, and the lemma easily follows. ◀

In some cases it is convenient to study switching walks and circuits in the language of an auxiliary bipartite graph. Let $V^{\mathsf{out}} = \{v^{\mathsf{out}} \mid v \in V(G)\}$ and $V^{\mathsf{in}} = \{v^{\mathsf{in}} \mid v \in V(G)\}$. The *interface graph of* G is the bipartite graph I_G such that $V(I_G) = V^{\mathsf{out}} \cup V^{\mathsf{in}}$ and $E(I_G) = \{u^{\mathsf{out}}v^{\mathsf{in}} \mid (u,v) \in E(G)\}$. Clearly, there is a one-to-one correspondence between interfaces in G and vertices of I_G, and the degree of a vertex in I_G is the size of the corresponding interface. Moreover, if $\pi = e_1, \ldots, e_k$ is a switching walk in G with a vertex sequence $v_1, \ldots, v_{k+1}$ and interface sequence $I_1, \ldots, I_{k+1}$, then π corresponds to a simple path $I(\pi) = v_1^{\mathsf{type}(I_1)}, \ldots, v_{k+1}^{\mathsf{type}(I_{k+1})}$ in G with endpoints of degree larger than 2, and all inner vertices of degree 2. Similarly, a switching circuit π corresponds to a simple cycle $I(\pi)$ in I_G with all vertices of degree 2 in I_G, i.e., $I(\pi)$ forms a connected component in I_G. Observe that both in the case of path and cycle above, the edges $I(\pi)$ are exactly the edges of I_G corresponding to the edges of π. Using the equivalence described in this paragraph, the following lemma is immediate.

▶ **Lemma 3.2.** *Edges of every digraph can be uniquely partitioned into switching walks and circuits. Moreover, the partition can be computed in linear time.*

Proof. Let G be a digraph. Recall that by the definition of I_G, there is a one-to-one correspondence between edges of G and edges of I_G. It is clear that edges of I_G can be uniquely partitioned into (1) cycles with all vertices of degree 2 and (2) paths with both endpoints of degree at least 3 and all inner vertices of degree 2. The corresponding switching circuits and switching walks form the desired partition of $E(G)$. An algorithm which constructs the partition is straightforward. ◀

3.2 Branching subroutine

Let us consider a digraph G. By $t_i(G)$ we will denote the number of vertices of G with outdegree equal to i. Let $k = n - t_1(G)$ be the number of vertices of G with outdegree at least 2, and let $s_1, \ldots, s_k$ be the sequence of these outdegrees. Then, let us denote the sum $\sum_{i=1}^{k}(s_i - 2)$ by $S(G)$. An analogous sum for indegrees will be denoted by $S^-(G)$. Note that if G has no vertex of out- or indegree 1, then by the handshaking lemma $S(G) = S^-(G)$.

▶ **Theorem 3.3.** ATSP *can be solved in time* $\mathcal{O}^*(2^{(n-t_1(G))/3 \,+\, \beta S(G)})$ *and polynomial space, where* $\beta = \log_2 3 - 1 < 0.585$.

Proof. The idea behind this algorithm is to branch on interfaces of size greater than 2, reducing the initial problem to the case of $(2,2)$-graphs, and then to apply Corollary 2.6. A detailed description is presented in Pseudocode 1. Our algorithm consists of two functions: $\mathsf{AtspBranching}(G, \mathsf{weight})$ – the main one, which solves ATSP in G, and an auxiliary function $\mathsf{AtspForcedEdge}(G, \mathsf{weight}, e)$ that returns the minimum weight of a Hamiltonian cycle H in G such that $e \in H$ (or ∞ if there is no such cycle). Note that AtspForcedEdge modifies the input digraph G, and calls AtspBranching on the new digraph G'. We observe that every Hamiltonian cycle in G' of weight w corresponds to a Hamiltonian cycle in G of weight $w + \mathsf{weight}(e)$ and containing edge e, and vice versa.

Given a digraph G with a function $\mathsf{weight} : E(G) \to \mathbb{Z}$, AtspBranching starts by considering a number of trivial cases $(a)-(c)$, where either G has only 2 vertices, or there is a vertex with out- or indegree at most 1. Next, we apply Lemma 3.2 to decompose $E(G)$ into switching walks and circuits, and we deal with a situation when there is a switching walk $\pi = (e_1, \ldots, e_{2k})$ of even length in G (cases $(d)-(e)$ in Pseudocode 1). Denote by I, respectively I', the interface which π starts, respectively ends, at. Consider a Hamiltonian cycle H in G. By Lemma 3.1 we obtain that either $e_1 \in H \cap \pi$, or $e_{2k} \in H \cap \pi$. We consider the following two cases.

Algorithm 1 AtspBranching(G, weight).

Input: G – a digraph on $n \geq 2$ vertices,
weight – a function $E(G) \to \mathbb{Z}$
Output: the minimum weight of a Hamiltonian cycle in G,
or ∞ if there is no such cycle

Function AtspForcedEdge(G, weight, e)**:**
Let $e = (u, v)$
$G_1 \leftarrow G$ with removed edges of the form (v, u), (u, x) and (x, v) for $x \in V(G)$
$G' \leftarrow G_1$ with contracted vertices u and v
$\mathsf{weight}' \leftarrow$ weights of $E(G')$ inherited from G appropriately
return $\mathsf{weight}(e)$ + AtspBranching(G', weight')

Function AtspBranching(G, weight)**:**
if *G has exactly two vertices u and v* **then** (a)
return $\mathsf{weight}((u, v)) + \mathsf{weight}((v, u))$ *if* $(u, v), (v, u) \in E(G)$*, or* ∞ *otherwise*
if *there is an empty interface in G i.e. a vertex of out- or indegree* 0 **then** (b)
return ∞
if *there is an interface $I = \{e\}$ of size* 1 **then** (c)
return AtspForcedEdge(G, weight, e)
Use Lemma 3.2 to partition $E(G)$ into switching walks and circuits
if *there is a switching walk π which begins and ends at the same interface I* **then** (d)
$G' \leftarrow G$ with removed edges of $I \setminus \pi$
return AtspBranching(G', weight)
if *there is a switching walk π of even length* **then** (e)
Let $\pi = (e_1, \ldots, e_{2k})$
return $\min($AtspForcedEdge$(G, \mathsf{weight}, e_1)$, AtspForcedEdge$(G, \mathsf{weight}, e_{2k}))$
if *there is no interface of size at least* 3 **then** (f)
Apply Corollary 2.6 to G and return the weight of the solution, or ∞
else (g)
Let $I = \{e_1, \ldots, e_s\}$ be an out-interface of size $s \geq 3$
$\mathsf{result} \leftarrow \infty$
for $i = 1, \ldots, s$ **do**
$\mathsf{result} \leftarrow \min(\mathsf{result},$ AtspForcedEdge$(G, \mathsf{weight}, e_i))$
return result

- If $I = I'$, then we have $H \cap I \in \{e_1, e_{2k}\}$, and thus all edges of $I \setminus \pi$ can be safely removed as they cannot be extended to a Hamiltonian cycle in G. This is realized in step (d) of the pseudocode. Note that if a switching walk π starts and ends at the same interface, then it must be of even length, since orientation of edges on π alternates.

- If $I \neq I'$, we branch by guessing if $e_1 \in H \cap \pi$, or $e_{2k} \in H \cap \pi$ (step (e) of the pseudocode).

If none of the above cases holds, we check whether all interfaces consist of at most 2 edges (cases $(f) - (g)$ in Pseudocode 1). If so, then G is a $(2, 2)$-graph, and we can solve ATSP for G by applying Corollary 2.6. If not, we choose an out-interface I of size at least 3, and we branch on it, by guessing which of the edges of I to pick as a part of a Hamiltonian cycle. Note that since G has no interface of size 1, then it has an interface of size at least 3 if and only if it has an out-interface of size at least 3.

Time complexity analysis. We begin with providing a few simple facts concerning the properties of our algorithm.

▷ Claim 3.4 (♠). During execution of algorithm AtspBranching, the value of $S(G)$ cannot increase.

▷ Claim 3.5 (♠). During execution of algorithm AtspBranching, graph G is simple, i.e. does not contain two edges of the same head and tail.

▷ Claim 3.6. Let $\pi = (e_1, \ldots, e_k)$ be a switching walk in G. Assume that during the run of our algorithm we decided to take an edge e_1 by calling AtspForcedEdge$(G, \mathsf{weight}, e_1)$. Then, by exhaustively applying rule (c) of AtspBranching to the resulting digraph, we will remove from G all edges of the form e_{2i}, and contract all edges of the form e_{2i+1}. An analogous statement can be made if we start with discarding edge e_1 instead of contracting it.

Denote $f(n, S) = 2^{n/3+\beta S}$, where β is the constant from Theorem 3.3. We need to prove that the running time of our algorithm is bounded by $f(n - t_1(G), S(G))n^{\mathcal{O}(1)}$. We proceed by induction on $t_1(G) + S(G)$.

If $t_1(G) > 0$, then our algorithm starts by choosing edges which form interfaces of size 1, what leads to a digraph with at most $\max(2, n - t_1(G))$ vertices. Hence, by the induction hypothesis the running time is bounded by $f(n - t_1(G), S(G))n^{\mathcal{O}(1)}$.

In what follows we assume $t_1(G) = 0$. If G satisfies condition (a) or (b), then our algorithm runs in polynomial time. Similarly, we can assume that G does not satisfy conditions (c) and (d), as applying the corresponding reductions exhaustively takes only polynomial time and does not increase the value of $S(G)$, according to Claim 3.4.

From now on, we assume that conditions $(a) - (d)$ do not hold for G. If $S(G) = 0$, then our algorithm executes the algorithm from Corollary 2.6 and therefore its running time is bounded by $\mathcal{O}^*(2^{n/3})$, as desired. Now, assume $S(G) > 0$. It remains to analyse cases (e) and (g) of AtspBranching.

Case (e). Let us assume that there is a switching walk $\pi = (e_1, \ldots, e_{2k})$ of even length in G which starts at interface I of size $s \geq 3$, and ends at interface $I' \neq I$ of size $s' \geq 3$. Let G' be a digraph obtained from G by running AtspForcedEdge$(G, \mathsf{weight}, e_1)$ and exhaustively applying rules $(a) - (d)$ to the resulting digraph.

Since edge e_1 is contracted in AtspForcedEdge, we have $|V(G')| \leq |V(G)| - 1$. We claim that $S(G') \leq S(G) - 2$. Assume $\mathsf{type}(I) = \mathsf{type}(I') = \mathsf{out}$. By Claim 3.6, for all $i = 1, \ldots, k$, edge e_{2i-1} was contracted, and edge e_{2i} was removed. We observe that contracting edge e_1 results in removing interface I from the graph, and discarding edge e_{2k} decreases the size of I' by 1. By Claim 3.4 operations performed on edges $e_2, \ldots, e_{2k-1}$ do not increase the value of $S(G)$. Hence, $S(G) - S(G') \geq (s - 2) + 1 \geq 2$, as desired. If $\mathsf{type}(I) = \mathsf{type}(I') = \mathsf{in}$, then by the same reasoning, we obtain $S^-(G') \leq S(G) - 2$ but since there are no interfaces of size 1 in G', we have $S(G') = S^-(G')$, and the claim follows.

Hence, by the induction hypothesis, the running time of our algorithm applied to G' is bounded by $f(n-1, S(G)-2)$. To obtain the desired bound for digraph G we need to show that $2f(n-1, S(G)-2) \leq f(n, S(G))$, or, equivalently $\log_2(2f(n-1, S(G)-2)) \leq \log_2 f(n, S(G))$. We obtain

$$\begin{aligned}\log_2(2f(n-1, S(G)-2)) &= 1 + \tfrac{n-1}{3} + \beta(S(G)-2) = \tfrac{n}{3} + \beta S(G) + \tfrac{2}{3} - 2\beta \\ &\leq \tfrac{n}{3} + \beta S(G) = \log_2 f(n, S(G)).\end{aligned}$$

Case (*g*). Now, we assume that G does not satisfy conditions $(a)-(f)$. Let I be an out-interface of size $s \geq 3$, and consider an edge $e \in I$. Let G' be a digraph obtained from G after choosing edge e by running AtspForcedEdge(G, weight, e), and let G'' be a digraph obtained from G' by the subsequent exhaustive application of rules $(a)-(d)$ by AtspBranching. Define $\Delta n = |V(G)| - |V(G'')|$, and $\Delta S = S(G) - S(G'')$.

▷ Claim 3.7. It holds that $\Delta n \geq 1$, $\Delta S \geq s-2 \geq 1$, and $\Delta n + \Delta S \geq s+1$.

Proof. For a digraph G we denote $n(G) = |V(G)|$. First, we analyse a direct impact of calling AtspForcedEdge(G, weight, e). All edges of I are removed from G, hence by Claim 3.4 we have $\Delta S \geq S(G) - S(G') \geq s-2 \geq 1$. Moreover, edge e gets contracted, and thus $\Delta n \geq n(G) - n(G') = 1$. We are left with proving that $(n(G') - n(G'')) + (S(G') - S(G'')) \geq 2$, since then we will have $\Delta n + \Delta S \geq s+1$.

Let π be the switching walk which starts with edge e. Let e' be the last edge of π (it is possible that π has length 1 and $e' = e$). We recall that at step (g) every switching walk in G is of odd length. Take an in-interface I' such that $e' \in I'$. By the definition of switching walk, $|I'| \geq 3$, so let e', e'_1, e'_2 be three different edges of I'. For $j = 1, 2$ denote by π_j the switching walk which ends with edge e'_j. Let e_j be the first edge of π_j, and let I_j be an out-interface such that $e_j \in I_j$.

Let $F, R \subseteq E(G)$ be edges of G which correspond to the edges that were taken (and hence, contracted) and removed, respectively, during the run of our algorithm which leads from digraph G to digraph G''. We have $e \in F$. By Claim 3.6 applied to π, we obtain $e' \in F$. Therefore, $e'_1, e'_2 \in R$, and again by Claim 3.6 applied to π_1 and π_2, we obtain $e_1, e_2 \in R$. Now, we consider a few cases.

- If I, I_1, I_2 are pairwise different out-interfaces, then during processing of digraph G' we removed edges e_1, e_2 from different out-interfaces of size at least 3. Therefore, $S(G') - S(G'') \geq 2$.
- If $I = I_1 = I_2$, then among switching walks π, π_1, π_2 there are least two of length greater than 1 (hence, of length at least 3), because otherwise the graph is not simple, contradicting Claim 3.5. Let us assume that these are walks π and π_1 (the other cases are analogous). Then, by Claim 3.6, during processing of digraph G' we contracted edge e' and the second edge of π_1. Therefore, $n(G') - n(G'') \geq 2$.
- If $I_1 = I_2 \neq I$, or $I = I_1 \neq I_2$, or $I = I_2 \neq I_1$, then at least one switching walk among π, π_1, π_2 is of length at least 3, and there is another interface apart from I that gets smaller. Hence, we obtain in a similar way as before that $n(G'') - n(G') \geq 1$, and $S(G'') - S(G') \geq 1$. ◁

Since $\Delta S \geq 1$, we have $S(G'') < S(G)$, and thus by the induction hypothesis the running time of our algorithm applied to G'' is bounded by $f(n(G''), S(G'')) = f(n - \Delta n, S(G) - \Delta S)$. In step (g) of AtspBranching we branch into s such subcases, hence we need to prove that $s \cdot f(n - \Delta n, S(G) - \Delta S)) \leq f(n, S(G))$. We will show the equivalent $\log_2(s \cdot f(n - \Delta n, S(G) - \Delta S)) \leq \log_2 f(n, S(G))$. Indeed,

$$
\begin{aligned}
&\log_2(s \cdot f(n - \Delta n, S(G) - \Delta S)) \\
&\quad = \log_2 s + \tfrac{n - \Delta n}{3} + \beta(S(G) - \Delta S) \\
&\quad = \tfrac{n}{3} + \beta S(G) + \log_2 s - \tfrac{\Delta n}{3} - \beta \Delta S \\
&\quad \leq \tfrac{n}{3} + \beta S(G) + \log_2 s - \tfrac{s+1-\Delta S}{3} - \beta \Delta S && \text{(Claim 3.7)} \\
&\quad = \tfrac{n}{3} + \beta S(G) + \log_2 s - \tfrac{s+1}{3} - (\beta - \tfrac{1}{3}) \Delta S \\
&\quad \leq \tfrac{n}{3} + \beta S(G) + \log_2 s - \tfrac{s+1}{3} - (\beta - \tfrac{1}{3})(s-2) && \text{(Claim 3.7)}
\end{aligned}
$$

$$
\begin{aligned}
&= \tfrac{n}{3} + \beta S(G) + \log_2 s - 1 - \beta(s-2) \\
&\leq \tfrac{n}{3} + \beta S(G) && (\triangle) \\
&= \log_2 f(n, S(G)).
\end{aligned}
$$

where inequality $(\triangle)$ follows from the fact that the function $x \mapsto \frac{\log_2 x - 1}{x-2}$ is decreasing on $[3, \infty)$, and thus it can be bounded by the value at $x = 3$ which is equal to β. Consequently, the inequality $\log_2 s \leq 1 + \beta(s-2)$ holds for $s \geq 3$. ◀

3.3 General algorithm

The idea behind our general algorithm is to run in parallel two algorithms: our branching algorithm from Theorem 3.3 (which we will refer to as Algorithm Ⓐ), and enumerating cycle covers from Subsection 2.2 (Algorithm Ⓑ) . We finish when one of these algorithms terminates. Our goal is to prove that the time complexity of such an approach is bounded by $\mathcal{O}^*(2^{\alpha(d-1)n})$ if we apply it to digraphs of average outdegree at most d, where α is the constant from Theorem 1.1. (Note that when implementing this algorithm, one may also compare the values of $\frac{n}{3} + \beta S(G)$ and $\alpha(d-1)n$, and, depending on the result, run either Algorithm Ⓐ, or Algorithm Ⓑ.) For the time complexity analysis, see the full version.

4 Exponential space algorithm

In this section we establish Theorem 1.2.

Let G be a digraph with n vertices and $m = dn$ edges. For simplicity, we assume in this section that n is even, for otherwise we can pick an arbitrary vertex v, and split it into two vertices v^{in} and v^{out} with edges inherited from v appropriately and with one additional edge $(v^{\mathsf{in}}, v^{\mathsf{out}})$ – this operation adds one vertex to the graph but does not increase the average outdegree. We will say that a simple path P in G is (l, D)-*light* if the length of P is l, and the sum of outdegrees of inner vertices of P is bounded by D. For a vertex $v \in V(G)$, and positive integers l, D, by $\mathcal{P}_{v,l,D}$ we will denote the family of all (l, D)-light paths in G which start at vertex v.

Our algorithm relies on the following two lemmas.

▶ **Lemma 4.1.** *Let H be a Hamiltonian cycle in G. Then, the edges of H can be partitioned into two $(n/2,\ m/2)$-light paths.*

▶ **Lemma 4.2.** *For a digraph G, a vertex v, and integers l, D, the family $\mathcal{P}_{v,l,D}$ can be computed in time $\tau(D/(l-1))^{l-1} n^{\mathcal{O}(1)}$ where the function τ is defined as in Theorem 1.2.*

Before we proceed to the proofs of above lemmas, let us see how to derive Theorem 1.2 from them. Given a digraph G, the algorithm starts by iterating over all pairs of distinct vertices u_1 and u_2. For each such a pair we use Lemma 4.2 to obtain the families $\mathcal{P}_1 = \mathcal{P}_{u_1,n/2,m/2}$ and $\mathcal{P}_2 = \mathcal{P}_{u_2,n/2,m/2}$. By filtering them, we may assume that all paths from $\mathcal{P}_1$ end at u_2, and all paths from $\mathcal{P}_2$ end at u_1. Next, we create a dictionary $\mathcal{D}$ with an entry $\{\text{key} : V(P_1), \text{value} : \text{weight}(P_1)\}$ for every path $P_1 \in \mathcal{P}_1$. (In case there is more than one path on the same set of vertices we keep only one entry with the minimum weight.) Then, we iterate over all paths $P_2 \in \mathcal{P}_2$, and we look up in $\mathcal{D}$ a subset $V'(P_2) := (V(G) \setminus V(P_2)) \cup \{u_1, u_2\}$. For every hit we calculate the sum: $\text{weight}(P_2) + \mathcal{D}[V'(P_2)]$, and we return the minimum of these values.

Algorithm 2 GeneratePaths(G, path, l, D).

Input: G – a digraph,
path – a sequence of vertices forming a path in G,
l, D – positive integers
Output: A collection of all simple paths of the form: path$\#(v_1, \ldots, v_l)$ such that $\sum_{i=1}^{l-1} \text{outdeg}(v_i) \leq D$

```
u ← the last vertex on path;
for vertex v1 such that (u, v1) ∈ E(G) and v1 ∉ path do
    if l = 1 then
        print path#v1;
    else if outdeg(v1) + (l − 2) ≤ D then                    (✓)
        GeneratePaths(G, path#v1, l − 1, D − outdeg(v1));
```

The correctness of this procedure is a direct corollary from Lemma 4.1. Moreover, the running time of the algorithm is dominated, up to a polynomial factor, by the running time of the algorithm from Lemma 4.2, which in our case is bounded by

$$\tau \left(\frac{\frac{m}{2}}{\frac{n}{2} - 1} \right)^{n/2-1} n^{\mathcal{O}(1)} = \tau \left(\frac{d}{1 - \frac{2}{n}} \right)^{n/2-1} n^{\mathcal{O}(1)} = \tau(d)^{n/2} n^{\mathcal{O}(1)}$$

where the last equality follows from the fact that when d is fixed, then for sufficiently large n we have $\lfloor d/(1 - \frac{2}{n}) \rfloor = \lfloor d \rfloor$. Note that we implement the dictionary $\mathcal{D}$ as a balanced tree, so each lookup takes time $\mathcal{O}(\log |\mathcal{D}|) = \mathcal{O}(n)$.

Proof of Lemma 4.1. Let $k = n/2$, and let $d_0, d_1, \ldots, d_{2k-1}$ be the outdegrees of consecutive vertices on H. Denote $S_i = d_i + d_{i+1} + \ldots + d_{i+k-1}$. (In this proof indices are understood modulo $2k$.) We need to prove that for some index j both expressions $S_j - d_j$ and $S_{j+k} - d_{j+k}$ do not exceed $m/2$.

Let $R_i := S_i - S_{i+k}$. We observe that $R_k = S_k - S_0 = -R_0$. Hence, there exists an index $j \in \{0, \ldots, k-1\}$ such that $R_j \cdot R_{j+1} \leq 0$. Without loss of generality, we may assume that $R_j \leq 0$ (equivalently, $S_j \leq S_{j+k}$), for otherwise we can just shift all indices by k. Then, $R_{j+1} \geq 0$ (equivalently, $S_{j+1+k} \leq S_{j+1}$). Thus we obtain

$$\begin{aligned} S_j - d_j &\leq S_j \leq \tfrac{1}{2}(S_j + S_{j+k}) = \tfrac{m}{2} \\ S_{j+k} - d_{j+k} &\leq S_{j+k+1} \leq \tfrac{1}{2}(S_{j+k+1} + S_{j+1}) = \tfrac{m}{2}. \end{aligned}$$

This ends the proof. ◀

Before we proceed to the proof of Lemma 4.2, we state a technical lemma.

▶ **Lemma 4.3 (♠).** *Let $a_1, \ldots, a_k$ be integers with an average bounded by $\bar{a}$. Then, $a_1 \cdot \ldots \cdot a_k \leq \tau(\bar{a})^k$.*

Proof of Lemma 4.2. We apply a simple branching procedure which starts at vertex v, and at each step guesses the next vertex on a path by considering all *reasonable* possibilities. A detailed description of the algorithm can be found in Pseudocode 2. (To compute the family $\mathcal{P}_{v,l,D}$ we call the function GeneratePaths with the arguments $(G, \{v\}, l, D)$.) Note that before appending a vertex to the current path we check whether the sum of outdegrees on the new path is not too large (line marked with (✓) in the Pseudocode). More precisely, we check whether appending a sequence of vertices of outdegree 1 to the new path would give us a correct (l, D)-path.

The correctness of such a procedure is straightforward. It remains to estimate its time complexity. Let $\mathcal{T}(G)$ be a search tree representing execution of this algorithm. We claim that $\mathcal{T}(G)$ contains at most $\tau(D/(l-1))^{l-1}n$ leaves, where $\tau(d) = \lfloor d \rfloor^{\lfloor d \rfloor+1-d}(\lfloor d \rfloor + 1)^{d-\lfloor d \rfloor} \leq d$

We will say that a directed rooted tree $\mathcal{T}$ has the property $(\star)$ if for any path $u_0, \ldots, u_h$ from the root of $\mathcal{T}$ to some leaf u_h we have $h \leq l$, and the sum $\sum_{i=1}^{h-1} \mathrm{outdeg}(u_i)$ is bounded by $D - (l-h)$. From the description of the algorithm we see that $\mathcal{T}(G)$ has the property $(\star)$. Indeed, let $u_0, \ldots, u_h$ be a path from the root to a leaf in $\mathcal{T}(G)$. Before the algorithm entered the vertex u_{h-1} the following condition was checked:

$$\mathrm{outdeg}(u_{h-1}) + (l - (h-2) - 2) \leq D - \sum_{i=1}^{h-2} \mathrm{outdeg}(u_i)$$

which is equivalent to $\sum_{i=1}^{h-1} \mathrm{outdeg}(u_i) \leq D - (l-h)$. Hence it is enough to prove the following claim. (A similar statement appears in the work of Gebauer [19].)

▷ Claim 4.4. Any tree $\mathcal{T}$ with the property $(\star)$ has at most $\tau(D/(l-1))^{l-1}n$ leaves.

Given a tree $\mathcal{T}$ with the property $(\star)$ we modify it so that the property $(\star)$ is preserved and the number of leaves in it does not increase. First, we may assume that all leaves in $\mathcal{T}$ are at depth exactly l. Indeed, let $u_0, \ldots, u_h$ be a path from the root of $\mathcal{T}$ to some leaf u_h at depth $h < l$. Then, we may append to it a path $u_h, u_{h+1}, \ldots, u_l$ – this operation does not change the number of leaves, and the property $(\star)$ is preserved because

$$\sum_{i=1}^{l-1} \mathrm{outdeg}(u_i) = \sum_{i=1}^{h-1} \mathrm{outdeg}(u_i) + (l-h) \leq D$$

Next, we modify $\mathcal{T}$ iteratively. Let $\mathcal{T}_1 := \mathcal{T}$. At i-th step, for $i = 1, \ldots, l-1$, we consider the family $\mathcal{S}_i$ of all subtrees in $\mathcal{T}_i$ with a root at depth i. Let $S_i \in \mathcal{S}_i$ be a subtree with the maximum number of leaves. We create a tree $\mathcal{T}_{i+1}$ by substituting in $\mathcal{T}_i$ all subtrees from $\mathcal{S}_i$ with S_i. We observe that for every $i = 1, \ldots, l-1$ tree $\mathcal{T}_i$ has depth l, the number of leaves in $\mathcal{T}_i$ is bounded by the number of leaves in $\mathcal{T}_{i+1}$, and all vertices in $\mathcal{T}_i$ at the same depth $j \leq i-1$ have the same outdegree. Combining the latter property with the fact that the condition $(\star)$ holds for leaves in the subtree S_i, we obtain inductively that every tree $\mathcal{T}_i$ still has the property $(\star)$.

Now, we consider the tree $\mathcal{T}_l$. For $i = 0, \ldots, l-1$ let d_i be the outdegree of any vertex at depth i in $\mathcal{T}_l$. Then we may bound the number of leaves in $\mathcal{T}_l$ by

$$d_0 \cdot \prod_{i=1}^{l-1} d_i \leq n \left(\frac{\sum_{i=1}^{l-1} d_i}{l-1} \right)^{l-1} \leq n \left(\frac{D}{l-1} \right)^{l-1}$$

To obtain a tighter bound on the size of $\mathcal{T}_l$ we observe that in the above estimation we obtain an equality only if $d_i = D/(l-1)$ for $i = 1, \ldots, l-1$. However, this is impossible unless expression $D/(l-1)$ is integral. After applying Lemma 4.3 we get the tighter bound which proves the claim of Lemma 4.2. ◀

References

1 Eric T. Bax. Inclusion and exclusion algorithm for the Hamiltonian path problem. *Inf. Process. Lett.*, 47(4):203–207, September 1993. doi:10.1016/0020-0190(93)90033-6.

2 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, January 1962. doi:10.1145/321105.321111.

3 Andreas Björklund. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014. doi:10.1137/110839229.

4 Andreas Björklund. Exploiting sparsity for bipartite hamiltonicity. In Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao, editors, *29th International Symposium on Algorithms and Computation (ISAAC 2018)*, volume 123 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:11, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ISAAC.2018.3.

5 Andreas Björklund. An asymptotically fast polynomial space algorithm for hamiltonicity detection in sparse directed graphs, 2020. arXiv:2009.11780.

6 Andreas Björklund and Thore Husfeldt. The parity of directed Hamiltonian cycles. In *FOCS*, pages 727–735. IEEE Computer Society, 2013.

7 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. The traveling salesman problem in bounded degree graphs. *ACM Trans. Algorithms*, 8(2):18:1–18:13, 2012. doi:10.1145/2151171.2151181.

8 Andreas Björklund, Vikram Kamat, Lukasz Kowalik, and Meirav Zehavi. Spotting trees with few leaves. *SIAM J. Discret. Math.*, 31(2):687–713, 2017. doi:10.1137/15M1048975.

9 Andreas Björklund, Petteri Kaski, and Ioannis Koutis. Directed hamiltonicity and out-branchings via generalized laplacians. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 91:1–91:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.91.

10 Andreas Björklund and Ryan Williams. Computing permanents and counting Hamiltonian cycles by listing dissimilar vectors. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 25: 1–25: 14. Schloss Dagstuhl - Leibniz Center for Computer Science, 2019. doi:10.4230/LIPIcs.ICALP.2019.25.

11 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

12 Lev Meerovich Brègman. Some properties of nonnegative matrices and their permanents. *Doklady Akademii Nauk*, 211(1):27–30, 1973.

13 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3), March 2018. doi:10.1145/3148227.

14 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.23.

15 Marek Cygan and Marcin Pilipczuk. Faster exponential-time algorithms in graphs of bounded average degree. *Inf. Comput.*, 243:75–85, 2015. doi:10.1016/j.ic.2014.12.007.

16 David Eppstein. The traveling salesman problem for cubic graphs. *J. Graph Algorithms Appl.*, 11(1):61–81, 2007. doi:10.7155/jgaa.00137.

17 Fedor V. Fomin, Serge Gaspers, Saket Saurabh, and Alexey A. Stepanov. On two techniques of combining branching and treewidth. *Algorithmica*, 54(2):181–207, 2009. doi:10.1007/s00453-007-9133-3.

18 Komei Fukuda and Tomomi Matsui. Finding all the perfect matchings in bipartite graphs. *Applied Mathematics Letters*, 7(1):15–18, 1994.

19 Heidi Gebauer. How many Hamilton cycles and perfect matchings are there? Master's thesis, ETH Zürich, March 2007.

20 Heidi Gebauer. On the number of Hamilton cycles in bounded degree graphs. In Robert Sedgewick and Wojciech Szpankowski, editors, *Proceedings of the Fifth Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2008, San Francisco, California, USA, January 19, 2008*, pages 241–248. SIAM, 2008. doi:10.1137/1.9781611972986.8.

21 Heidi Gebauer. Finding and enumerating Hamilton cycles in 4-regular graphs. *Theor. Comput. Sci.*, 412(35):4579–4591, 2011. doi:10.1016/j.tcs.2011.04.038.

22 Yuri Gurevich and Saharon Shelah. Expected computation time for Hamiltonian path problem. *SIAM J. Comput.*, 16(3):486–502, June 1987. doi:10.1137/0216034.

23 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. In *Proceedings of the 1961 16th ACM National Meeting*, ACM '61, page 71.201–71.204, New York, NY, USA, 1961. Association for Computing Machinery. doi:10.1145/800029.808532.

24 Richard M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Oper. Res. Lett.*, 1(2):49–51, April 1982. doi:10.1016/0167-6377(82)90044-X.

25 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. A bound on the pathwidth of sparse graphs with applications to exact algorithms. *SIAM J. Discret. Math.*, 23(1):407–427, 2009. doi:10.1137/080715482.

26 Samuel Kohn, Allan Gottlieb, and Meryle Kohn. A generating function approach to the traveling salesman problem. In *Proceedings of the 1977 Annual Conference*, ACM '77, page 294–300, New York, NY, USA, 1977. Association for Computing Machinery. doi:10.1145/800179.810218.

27 Łukasz Kowalik and Konrad Majewski. The asymmetric travelling salesman problem in sparse digraphs, 2020. arXiv:2007.12120.

28 Jesper Nederlof. Bipartite TSP in $o(1.9999^n)$ time, assuming quadratic time matrix multiplication. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 40–53. ACM, 2020. doi:10.1145/3357713.3384264.

29 Mohd Shahrizan Othman, Aleksandar Shurbevski, and Hiroshi Nagamochi. Polynomial-space exact algorithms for the bipartite traveling salesman problem. *IEICE Trans. Inf. Syst.*, 101-D(3):611–612, 2018. doi:10.1587/transinf.2017FCL0003.

30 Ján Plesník. The NP-completeness of the Hamiltonian cycle problem in planar digraphs with degree bound two. *Inf. Process. Lett.*, 8(4):199–201, 1979. doi:10.1016/0020-0190(79)90023-1.

31 Frank Rubin. A search procedure for Hamilton paths and circuits. *J. ACM*, 21(4):576–580, October 1974. doi:10.1145/321850.321854.

32 Alexander Schrijver. A short proof of Minc's conjecture. *Journal of combinatorial theory, Series A*, 25(1):80–83, 1978.

33 Mingyu Xiao and Hiroshi Nagamochi. An exact algorithm for TSP in degree-3 graphs via circuit procedure and amortization on connectivity structure. *Algorithmica*, 74(2):713–741, 2016. doi:10.1007/s00453-015-9970-4.

34 Mingyu Xiao and Hiroshi Nagamochi. An improved exact algorithm for TSP in graphs of maximum degree 4. *Theory Comput. Syst.*, 58(2):241–272, 2016. doi:10.1007/s00224-015-9612-x.

35 Norhazwani Md Yunos, Aleksandar Shurbevski, and Hiroshi Nagamochi. An improved-time polynomial-space exact algorithm for TSP in degree-5 graphs. *J. Inf. Process.*, 25:639–654, 2017. doi:10.2197/ipsjjip.25.639.

On the Parameterized Complexity of Reconfiguration of Connected Dominating Sets

Daniel Lokshtanov
University of California Santa Barbara, CA, USA
daniello@ucsb.edu

Amer E. Mouawad
Department of Computer Science, American University of Beirut, Lebanon
aa368@aub.edu.lb

Fahad Panolan
Department of Computer Science and Engineering, IIT Hyderabad, India
fahad@cse.iith.ac.in

Sebastian Siebertz
University of Bremen, Germany
siebertz@uni-bremen.de

Abstract

In a reconfiguration version of a decision problem $\mathcal{Q}$ the input is an instance of $\mathcal{Q}$ and two feasible solutions S and T. The objective is to determine whether there exists a step-by-step transformation between S and T such that all intermediate steps also constitute feasible solutions. In this work, we study the parameterized complexity of the Connected Dominating Set Reconfiguration problem (CDS-R). It was shown in previous work that the Dominating Set Reconfiguration problem (DS-R) parameterized by k, the maximum allowed size of a dominating set in a reconfiguration sequence, is fixed-parameter tractable on all graphs that exclude a biclique $K_{d,d}$ as a subgraph, for some constant $d \geq 1$. We show that the additional connectivity constraint makes the problem much harder, namely, that CDS-R is W[1]-hard parameterized by $k + \ell$, the maximum allowed size of a dominating set plus the length of the reconfiguration sequence, already on 5-degenerate graphs. On the positive side, we show that CDS-R parameterized by k is fixed-parameter tractable, and in fact admits a polynomial kernel on planar graphs.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases reconfiguration, parameterized complexity, connected dominating set, graph structure theory

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.24

Related Version A full version of the paper is available at `https://arxiv.org/pdf/1910.00581.pdf`.

Funding *Amer E. Mouawad*: The second author is supported by URB project "A theory of change through the lens of reconfiguration".

1 Introduction

In a decision problem $\mathcal{Q}$, we are usually asked to determine the existence of a feasible solution for an instance $\mathcal{I}$ of $\mathcal{Q}$. In a *reconfiguration version* of $\mathcal{Q}$, we are instead given a source feasible solution S and a target feasible solution T and we are asked to determine whether it is possible to transform S into T by a sequence of step-by-step transformations such that after each intermediate step we also maintain feasible solutions. Formally, we consider a graph, called the *reconfiguration graph*, that has one vertex for each feasible solution and where two vertices are connected by an edge if we allow the transformation between the two

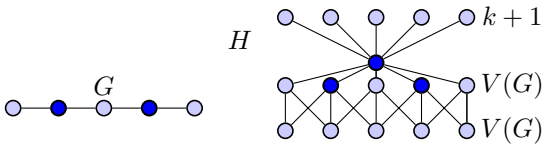


Figure 1 A graph G with a minimum dominating set of size $k = 2$ marked in dark blue and the graph H obtained in the standard reduction from Dominating Set to Connected Dominating Set. G has a dominating set of size k if and only if H has a connected dominating set of size $k + 1$. If p is equal to the pathwidth of G then the pathwidth of H is bounded by $2p + 1$.

corresponding solutions. We are then asked to determine whether S and T are connected in the reconfiguration graph, or even to compute a shortest path between them. Historically, the study of reconfiguration questions predates the field of computer science, as many classic one-player games can be formulated as such reachability questions [18, 20], e.g., the 15-puzzle and Rubik's cube. More recently, reconfiguration problems have emerged from computational problems in different areas such as graph theory [1, 16, 17], constraint satisfaction [12, 25] and computational geometry [5,19,23], and even quantum complexity theory [11]. Reconfiguration problems have been receiving considerable attention in recent literature, we refer the reader to [24, 28, 32] for an extensive overview.

In this work, we consider the Connected Dominating Set Reconfiguration problem (CDS-R) in undirected graphs. A *dominating set* in a graph G is a set $D \subseteq V(G)$ such that every vertex of G lies either in D or is adjacent to a vertex in D. A dominating set D is a *connected dominating set* if the graph induced by D is connected. The Dominating Set problem and its connected variant have many applications, including the modeling of facility location problems, routing problems, and many more.

We study CDS-R under the *Token Addition/Removal* model (TAR model). Suppose we are given a connected dominating set D of a graph G, and imagine that a token is placed on each vertex in D. The TAR rule allows either the addition or removal of a single token at a time from D, if this results in a connected dominating set of size at most a given bound $k \geq 1$. A sequence $D_1, \ldots, D_\ell$ of connected dominating sets of a graph G is called a *reconfiguration sequence* between D_1 and D_ℓ under TAR if the change from D_i to D_{i+1} respects the TAR rule, for $1 \leq i < \ell$. The *length* of the reconfiguration sequence is $\ell - 1$.

The (Connected) Dominating Set Reconfiguration problem for TAR gets as input a graph G, two (connected) dominating sets S and T and an integer $k \geq 1$, and the task is to decide whether there exists a reconfiguration sequence between S and T under TAR using at most k tokens.

Structural properties of the reconfiguration graph for k-dominating sets were studied in [14, 31]. The Dominating Set Reconfiguration problem was shown to be PSPACE-complete in [15], even on split graphs, bipartite graphs, planar graphs and graphs of bounded bandwidth. Both pathwidth and treewidth of a graph are bounded by its bandwidth, hence the Dominating Set Reconfiguration problem is PSPACE-complete on graphs of bounded pathwidth and treewidth. These hardness results motivated the study of the parameterized complexity of the problem. It was shown in [26] that the Dominating Set Reconfiguration problem is W[2]-hard when parameterized by $k + \ell$, where k is the bound on the number of tokens and ℓ is the length of the reconfiguration sequence. However, the problem becomes fixed-parameter tractable on graphs that exclude a fixed complete bipartite graph $K_{d,d}$ as a subgraph, as shown in [22]. Such so-called *biclique-free* classes are very general sparse graph classes, including in particular the planar graphs, which are $K_{3,3}$-free.

In this work we study the complexity of CDS-R. The standard reduction from Dominating Set to Connected Dominating Set shows that CDS-R is also PSPACE-complete, even on graphs of bounded pathwidth (Figure 1). We hence turn our attention to the parameterized complexity of the problem. We first show that the additional connectivity constraint makes the problem much harder, namely, that CDS-R parameterized by $k + \ell$ is W[1]-hard already on 5-degenerate graphs. As 5-degenerate graphs exclude the biclique $K_{6,6}$ as a subgraph, Dominating Set Reconfiguration is fixed-parameter tractable on much more general graph classes than its connected variant. To prove hardness we first introduce an auxiliary problem that we believe is of independent interest. In the Colored Connected Subgraph problem we are given a graph G, an integer k, and a (not necessarily proper) coloring $c\colon V(G) \to C$, for some color set C with $|C| \leq k$. The question is whether G contains a vertex subset H on at most k vertices such that $G[H]$ is connected and H contains at least one vertex of every color in C (i.e., $c(V(H)) = C$). The reconfiguration variant Colored Connected Subgraph Reconfiguration (CCS-R) is defined as expected. We first prove that CCS-R reduces to CDS-R by a parameter preserving reduction (where $k + \ell$ is the parameter) and the degeneracy of the reduced graph is at most the degeneracy of the input graph plus one. We then prove that the known W[1]-hard problem Multicolored Clique (see [3] for definitions) reduces to CCS-R on 4-degenerate graphs. The last reduction has the additional property that for an input (G, c, k) of Multicolored Clique the resulting instance of CCS-R admits either a reconfiguration sequence of length $\mathcal{O}(k^3)$, or no reconfiguration sequence at all. Hence, we derive that both CDS-R and CCS-R are W[1]-hard parameterized by $k + \ell$ on 5-degenerate and 4-degenerate graphs, respectively.

The existence of a reconfiguration sequence of length at most ℓ with connected dominating sets of size at most k can be expressed by a first-order formula of length depending only on k and ℓ. It follows from [13] that the problem is fixed-parameter tractable parameterized by $k + \ell$ on every nowhere dense graph class and the same is implied by [2] for every class of bounded cliquewidth. Nowhere dense graph classes are very general classes of uniformly sparse graphs, in particular the class of planar graphs is nowhere dense. Nowhere dense classes are themselves biclique-free, but are not necessarily degenerate. Hence, our hardness result on degenerate graphs essentially settles the question of fixed-parameter tractability for the parameter $k + \ell$ on sparse graph classes. It remains an interesting open problem to find dense graph classes beyond classes of bounded cliquewidth on which the problem is fixed-parameter tractable.

We then turn our attention to the smaller parameter k alone. We show that CDS-R parameterized by k is fixed-parameter tractable on the class of planar graphs. Our approach is as follows. We first compute a small *domination core* for G, a set of vertices that captures exactly the domination properties of G for dominating sets of sizes not larger than k. The notion of a domination core was introduced in the study of the Distance-r Dominating Set problem on nowhere dense graph classes [4]. While the classification of interactions with the domination core would suffice to solve Dominating Set Reconfiguration on nowhere dense classes, additional difficulties arise for the connected variant. In a second step we use planarity to identify large subgraphs that have very simple interactions with the domination core and prove that they can be replaced by constant size gadgets such that the reconfiguration properties of G are preserved.

Observe that CDS-R parameterized by k is trivially fixed-parameter tractable on every class of bounded degree. The existence of a connected dominating set of size k implies that the diameter of G is bounded by $k + 2$, which in every bounded degree class implies a bound on the size of the graph depending only on the degree and k. We conjecture that CDS-R is fixed-parameter tractable parameterized by k on every nowhere dense graph class. However, resolving this conjecture remains open for future work (see Figure 2).

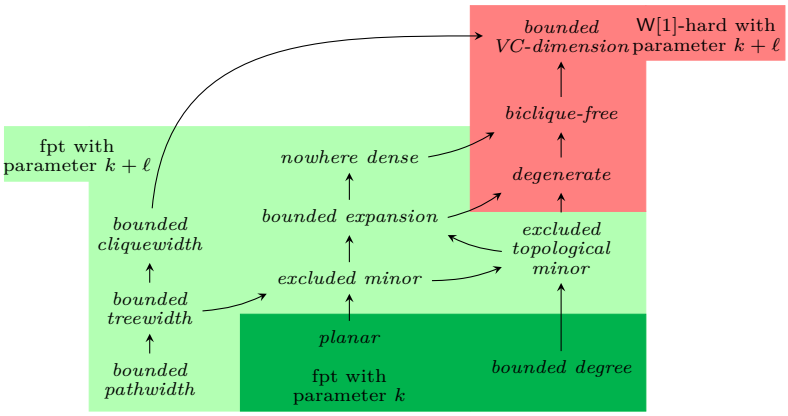


Figure 2 The map of tractability for Connected Dominating Set Reconfiguration. The classes colored in dark green admit an fpt algorithm with parameter k, the classes colored in light green admit an FPT algorithm with parameter $k+\ell$. On the classes colored in red the problem is W[1]-hard with respect to the parameter $k+\ell$.

The rest of the paper is organized as follows. We give background on graph theory and fix our notation in Section 2. We show hardness of CDS-R on degenerate graphs in Section 3 and show how to handle the planar case in Section 4. Due to space constraints proofs of results marked with a $\star$ are deferred to the full version of the paper.

2 Preliminaries

We denote the set of natural numbers by $\mathbb{N}$. For $n \in \mathbb{N}$, we let $[n] = \{1, 2, \ldots, n\}$. We assume that each graph G is finite, simple, and undirected. We let $V(G)$ and $E(G)$ denote the vertex set and edge set of G, respectively. An edge between two vertices u and v in a graph is denoted by $\{u, v\}$ or uv. The *open neighborhood* of a vertex v is denoted by $N_G(v) = \{u \mid \{u, v\} \in E(G)\}$ and the *closed neighborhood* by $N_G[v] = N_G(v) \cup \{v\}$. The degree of a vertex v, denoted $d_G(v)$, is $|N_G(v)|$. For a set of vertices $S \subseteq V(G)$, we define $N_G(S) = \{v \notin S \mid \{u, v\} \in E(G), u \in S\}$ and $N_G[S] = N_G(S) \cup S$. The subgraph of G induced by S is denoted by $G[S]$, where $G[S]$ has vertex set S and edge set $\{\{u, v\} \in E(G) \mid u, v \in S\}$. We let $G - S = G[V(G) \setminus S]$. A graph G is d-degenerate if every subgraph $H \subseteq G$ has a vertex of degree at most d. For a set C, we use $K[C]$ to denote the complete graph on vertex set C. For an integer $r \in \mathbb{N}$, an r-independent set in a graph G is a subset $U \subseteq V(G)$ such that for any two distinct vertices $u, v \in U$, the distance between u and v in G is more than r. An independent set in a graph is a 1-independent set. A subset of vertices U in G is called a separator in G if $G - U$ is has more than one connected component. For $s, t \in V(G)$, we say U is an (s, t)-separator in G if there is no path from s to t in $G - U$.

3 Hardness on degenerate graphs

In this section we prove that CDS-R and CCS-R are W[1]-hard when parameterized by $k + \ell$ even on 5-degenerate and 4-degenerate graphs, respectively. Towards that, we first give a polynomial-time reduction from the W[1]-hard Multicolored Clique problem to CCS-R on 4-degenerate graphs with the property that for an input (G, c, k) of Multicolored Clique the resulting instance of CCS-R admits either a reconfiguration sequence of length $\mathcal{O}(k^3)$ or no reconfiguration sequence at all. As a result, we conclude

that CCS-R is W[1]-hard when parameterized by $k + \ell$ on 4-degenerate graphs. Then, we give a parameter-preserving polynomial-time reduction from CCS-R to CDS-R. Let us first formally define the CCS problem.

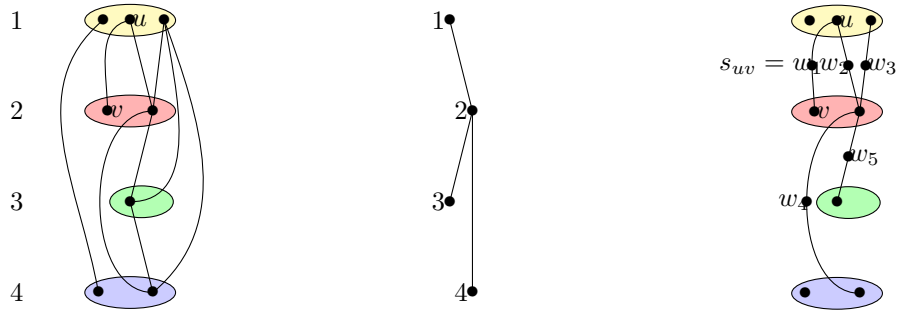
Colored Connected Subgraph (CCS)	**Parameter:** k
Input: A graph G, a vertex-coloring $c\colon V(G) \to C$, and $k \in \mathbb{N}$ such that $\lvert C\rvert \leq k$	
Question: Is there a vertex subset $S \subseteq V(G)$ of at most k vertices with at least one vertex from every color class such that $G[S]$ is connected?	

Reduction from Multicolored Clique to CCS-R

We now present the reduction from Multicolored Clique to CCS-R, which we believe to be of independent interest. We can assume, without loss of generality, that for an input (G, c, k) of Multicolored Clique, G is connected and c is a proper vertex-coloring, i.e., for any two distinct vertices $u, v \in V(G)$ with $c(u) = c(v)$ we have $\{u, v\} \notin E(G)$. Before we proceed let us define a graph operation.

▶ **Definition 3.1.** *Let G be a graph and let $c\colon V(G) \to \{1, \ldots, k\}$ be a proper vertex coloring of $V(G)$. Let H be a graph on the vertex set $\{1, \ldots, k\}$. We define the graph $G \restriction_c H$ as follows. We remove all edges $\{u, v\} \in E(G)$ such that $c(u) = i$ and $c(v) = j$ and $\{i, j\} \notin E(H)$. We subdivide every remaining edge, i.e. for every remaining edge $\{u, v\}$ we introduce a new vertex s_{uv}, remove the edge $\{u, v\}$ and introduce instead the two edges $\{u, s_{uv}\}$ and $\{v, s_{uv}\}$. We write $W(G \restriction_c H)$ for the set of all subdivision vertices s_{uv} (see Figure 3).*

That is, to construct $G \restriction_c H$, we first make a subgraph of G by deleting the edges between different color classes if there are no edges between the "corresponding" vertices in H, and then subdivide the remaining edges. Let (G, c, k) be the input instance of Multicolored Clique, where G is a connected graph and c is a proper k-vertex-coloring of G. We construct an instance $(H, \widehat{c}\colon V(H) \mapsto [k+1], Q_s, Q_t, 2k)$ of CCS-R (Q_s and Q_t are the source and target sets that we describe later). Note that the bound on the sizes of the solutions in the reconfiguration sequence is at most $2k$.



(a) A graph G and a proper coloring $c\colon V(G) \to \{1, \ldots, 4\}$.

(b) A graph H on the vertex set $\{1, \ldots, 4\}$.

(c) The graph $G \restriction_c H$. Here, $W(G \restriction_c H) = \{w_1, \ldots, w_5\}$.

Figure 3 Construction of $G \restriction_c H$.

We first construct a routing gadget. For $1 \leq i \leq k$, let T^i be the star with vertex set $\{1, \ldots, k\}$ having vertex i as the center. For any $1 \leq i \leq k$ and $1 \leq r \leq 20k$, we let $H^{(i,r)}$ be a copy of the graph $G \restriction_c T^i$. We let $c_{(i,r)}$ be the the partial vertex-coloring of $H^{(i,r)}$ that is naturally inherited from G. For an illustration, consider the input instance (G, c, k) of Multicolored Clique depicted in Figure 3a. Then, T^2 is identical to the graph H in Figure 3b and Figure 3c represents $H^{(2,r)} = G \restriction_c T^2$, for any $1 \leq r \leq 20k$. Now, for $1 \leq i \leq k$ we define a graph H^i as follows. We use $W(H^{(i,r)})$ to denote the set of subdivision vertices in $H^{(i,r)}$. For $1 \leq r < 20k$ and all vertices u, v in $V(H^{(i,r)}) \setminus W(H^{(i,r)})$, we connect the copy of the subdivision vertex s_{uv} in $H^{(i,r)}$ (if it exists) with the copies of the vertices u and v in $H^{(i,r+1)}$ (see Figure 4 for an illustration of a portion of H^1). We use $W(H^i)$ to denote the set of subdivision vertices $\bigcup_{r \in [20k]} W(H^{(i,r)})$.

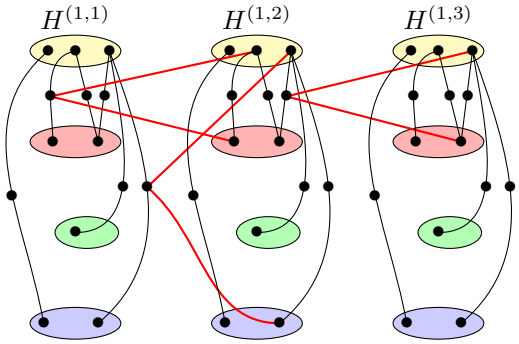


Figure 4 Construction of H^1 from the instance (G, c) depicted in Figure 3a. The red edges are some of the "crossing" edges but not all of them.

For each $1 \leq i \leq k$, we use c_i to denote a coloring on $V(H^i)$ that is a union of $c_{(i,1)}, c_{(i,2)}, \ldots, c_{(i,20k)}$ and we color all the copies of the subdivision vertices using a new color $k+1$. In other words, we know that for each $u \in V(H^i)$ we have $u \in V(H^{(i,r)})$, for some $r \in \{1, \ldots, 20k\}$. Hence, if $u \in V(H^{(i,r)}) \setminus W(H^{(i,r)})$ then we set $c_i(u) = c_{(i,r)}(u)$. For all $s_{uv} \in W(H^i)$, we set $c_i(s_{uv}) = k+1$.

Now, define a graph R, which is super graph of $H^1 \cup \ldots \cup H^k$, as follows. For $1 \leq i < k$ and all vertices u and v, we connect the copy of the subdivision vertex s_{uv} in $H^{(i,20k)}$ (if it exists) with the copies of the vertices u and v in $H^{(i+1,1)}$ (see Figure 5 for an illustration).

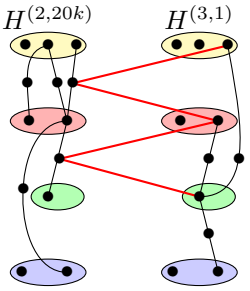


Figure 5 Illustration of the subgraph of R induced on $V(H^{(2,20k)}) \cup V(H^{3,1})$ constructed from the instance (G, c, k) depicted in Figure 3a. The red edge are some of the "crossing edges".

We additionally introduce two subgraphs H^0 and H^{k+1}. The graph H^0 is obtained by subdividing each edge of a star on vertex set $\{v_1, \ldots, v_k\}$ centered at v_1. Here we use $w_2, \ldots, w_k$ to denote the subdivision vertices. Similarly, the graph H^{k+1} is obtained by subdividing each

edge of star on $\{x_1, \ldots, x_k\}$ centered at x_k. Here $y_1, \ldots, y_{k-1}$ denote the subdivision vertices. Let c_0 and c_{k+1} be the colorings on $\{v_1, \ldots, v_k, w_2, \ldots, w_k\}$ and $\{x_1, \ldots, x_k, y_1, \ldots, y_{k-1}\}$, respectively, defined as follows. For all $1 \leq i \leq k$, $c_0(v_i) = i$ and $c_{k+1}(x_i) = i$. For all $2 \leq i \leq k$, $c_0(w_i) = k+1$ and for all $1 \leq i \leq k-1$, $c_{k+1}(y_i) = k+1$. Observe that we may interpret H^0 as $K[\{v_1, \ldots, v_k\}] \restriction_{c_0} T^0$ and H^{k+1} as $K[\{x_1, \ldots, x_k\}] \restriction_{c_{k+1}} T^{k+1}$, where T^0 and T^{k+1} are two trees on vertex set $\{1, \ldots, k\}$, with $E(T^0) = \{\{1, i\} \colon 2 \leq i \leq k\}$ and $E(T^{k+1}) = \{\{k, i\} \colon 1 \leq i \leq k-1\}$.

Finally, for each $2 \leq i \leq k$, we connect the "subdivision vertex" w_i (adjacent to v_1 and v_i) to all vertices $v \in V(H^{(1,1)})$ colored 1 or i, i.e., with $c_{(1,1)}(v) \in \{1, i\}$. For each subdivision vertex $s_{ab} \in W(H^{(k,20k)})$, we connect s_{ab} to x_k and x_i, where $k = c_k(a) = c_{(k,20k)}(a)$ and $i = c_k(b) = c_{(k,20k)}(b)$. Recall that s_{ab} is adjacent to a vertex of color k and a vertex of color i, for some $i < k$. This completes the construction of H (see Figure 6). We define $\widehat{c} \colon V(H) \mapsto [k+1]$ to be the union of $c_0, \ldots, c_{k+1}$. We define the starting configuration Q_s as the set $\{v_1, \ldots, v_k, w_2, \ldots, w_k\}$ and the target configuration Q_t as the set $\{x_1, \ldots, x_k, y_1, \ldots, y_{k-1}\}$.

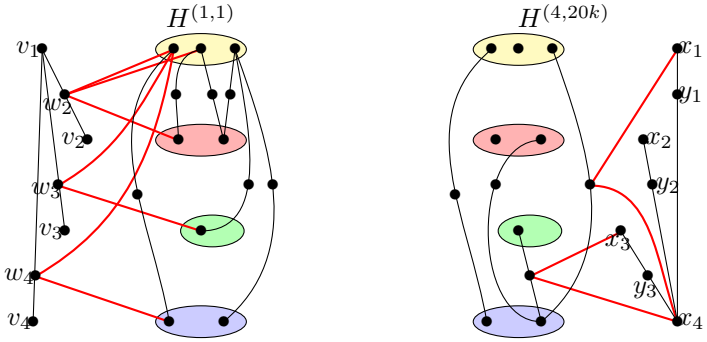


Figure 6 Illustration of connection between H^0 and R, and H^{k+1} and R from the instance (G, c, k) depicted in Figure 3a. The red edge are some of the "crossing edges" between H^0 and H^1, and H^k and H^{k+1}.

▶ **Proposition 3.2.** *The sets Q_s and Q_t are solutions of size $2k-1$ of the CCS instance $(H, \widehat{c}, 2k)$.*

We now consider the instance $(H, \widehat{c}, Q_s, Q_t, 2k)$ of the CCS-R problem. Before we analyze the reconfiguration properties of H, let us verify that H is 4-degenerate.

▶ **Lemma 3.3.** *The graph H is 4-degenerate.*

Proof. We iteratively remove minimum degree vertices and show that we can always remove a vertex of degree at most 4 in each step.

- Every subdivision vertex $w \in W(H^i)$ for $1 \leq i \leq k$ has degree at most 4; it has 4 neighbors in $V(H^i) \cup V(H^{i+1})$.
- After removal of all subdivision vertices the degree of the remaining vertices of each H^i is at most one. That is, a vertex in $H^{(1,1)}$ may have a neighbor in $\{w_2, \ldots, w_k\}$.
- After the removal of $V(H^1) \cup \ldots V(H^k)$, the degree of all vertices except v_1 and x_k is at most 2.
- Finally we remove v_1 and x_k.

This completes the proof. ◀

▶ **Lemma 3.4** (⋆). *If there exists a k-colored clique in G then there is reconfiguration sequence of length $\mathcal{O}(k^3)$ from Q_s to Q_t in $(H, \widehat{c}, 2k)$.*

Proof sketch. Informally, in every step our solution consists of vertices corresponding to a clique and at most k vertices from the subdivision vertices present in the clique. The reconfiguration sequence checks whether all the edges between the clique vertices are present.

We aim to shift the connected vertices of Q_s through the subgraphs $H^1, \ldots, H^k$ (in that order) to maintain connectivity and eventually shift all the tokens to Q_t. For each $u_i \in V(G)$, $1 \leq j \leq k$ and $1 \leq r \leq 20k$, we use $u_i^{(j,r)}$ to denote the copy of u_i in $H^{(j,r)}$. Let $C = \{u_1, \ldots, u_k\}$ be a k-colored clique in G such that $c(u_i) = i$, for all $1 \leq i \leq k$. To prove the lemma, we need to define a reconfiguration sequence starting from Q_s and ending at Q_t such that the cardinality of any solution in the sequence is at most $2k$. First we define k "colored" trees $\widehat{T}_1, \ldots, \widehat{T}_k$ each on $2k-1$ vertices, and then prove that there are reconfiguration sequences from Q_s to $V(\widehat{T}_1)$, $V(\widehat{T}_i)$ to $V(\widehat{T}_{i+1})$ for all $1 \leq i < k$, and $V(\widehat{T}_k)$ to Q_t.

We start by defining $\widehat{T}_1, \ldots, \widehat{T}_k$. For each $1 \leq i \leq k$, $C_i = \{u_1^{(i,1)}, \ldots, u_k^{(i,1)}\}$ and $S_i = \{z \in V(H^{(i,1)})\colon N_{H^{(i,1)}}(z) \cap C_i = 2\}$. That is, for each $1 \leq j \leq k$ and $j \neq i$, $s_{u_i^{(i,1)} u_j^{(i,1)}} \in S_i$ (the subdivision vertex on the edge $u_i^{(i,1)} u_j^{(i,1)}$ is in S_i), and $|S_i| = k-1$. In other words, C_i contains the copies of the vertices of the clique C in $H^{(i,1)}$ and S_i contains subdivision vertices corresponding to $k-1$ edges in the clique incident on the ith colored vertex of the clique, such that $H[C_i \cup S_i]$ is a tree. Now, define $\widehat{T}_i = H[C_i \cup S_i]$. It is easy to verify that $\widehat{c}(C_i \cup S_i) = \{1, \ldots, k+1\}$ and hence $C_i \cup S_i = V(\widehat{T}_i)$ is a solution to the CCS instance $(H, \widehat{c}, 2k)$. Let $T_s = H[Q_s]$ and $T_t = H[Q_t]$. Note that T_s and T_t are trees on $2k-1$ vertices.

Case 1: Reconfiguration from Q_s to $V(\widehat{T}_1)$. Informally, we move to $\widehat{T}_1$ by adding a token on $u_i^{(1,1)}$ and then removing tokens from v_i for i in the order $2, \ldots, k, 1$ (for a total of $2k$ token additions/removals). Finally, we move the tokens from $\{w_2, \ldots, w_{k-1}\}$ to S_1 in $2(k-1)$ steps. The length of the reconfiguration sequence is $2k + 2(k-1) = 4k-2$.

Case 2: Reconfiguration from $V(\widehat{T}_i)$ to $V(\widehat{T}_{i+1})$. First we define $20k$ trees $P_1, \ldots P_{20k}$, each on $2k-1$ vertices such that for all $1 \leq r \leq 20k$, (i) $V(P_r) \subseteq V(H^{(i,r)})$, and (ii) $\widehat{T}_i = P_1$. Then we give a reconfiguration sequence from $V(P_r)$ to $V(P_{r+1})$ for all $r \in [20k-1]$ and a reconfiguration sequence from $V(P_{20k})$ to $V(\widehat{T}_{i+1})$.

Recall that $C = \{u_1, \ldots, u_k\}$ is a k-colored clique in G such that $c(u_i) = i$ for all $1 \leq i \leq k$. For $1 \leq r \leq 20k$, let $C_i^r = \{u_1^{(i,r)}, \ldots, u_k^{(i,r)}\}$ and $S_i^r = \{z \in V(H^{(i,r)})\colon N_{H^{(i,r)}}(z) \cap C_i^r = 2\}$. That is, for each $1 \leq j \leq k$ and $j \neq i$, $s_{u_i^{(i,r)} u_j^{(i,r)}} \in S_i^r$ (i.e, the subdivision vertex on the edge $u_i^{(i,r)} u_j^{(i,r)}$ is in S_i^r) and $|S_i^r| = k-1$. Let $P_r = H[C_i^r \cup S_i^r]$. Notice that for all $r \in [20k]$, P_r is a tree on $2k-1$ vertices. Moreover, for each $1 \leq r \leq 20k$, $V(P_r)$ is a solution to the CCS instance $(H, \widehat{c}, 2k)$. By arguments similar to those given for Case 1, one can prove that there is a reconfiguration sequence of length $4k-2$ from $V(P_r)$ to $V(P_{r+1})$, for all $1 \leq r < 20k$.

For the reconfiguration sequence from $V(P_{20k})$ to $V(\widehat{T}_{i+1})$ we refer the reader to the complete proof in the full version of the paper.

Case 3: Reconfiguration from $V(\widehat{T}_k)$ to $V(T_t)$. The arguments for this case are similar to those given in Case 1, we therefore omit the details. ◀

▶ **Lemma 3.5** (⋆). *If there is a reconfiguration sequence from Q_s to Q_t then there is a k-colored clique in G.*

▶ **Theorem 3.6.** *CCS-R parameterized by $k + \ell$ is* W[1]*-hard on 4-degenerate graphs.*

Reduction from CCS-R to CDS-R

We give a polynomial-time parameter-preserving reduction from CCS-R to CDS-R that is fairly straightforward. Let (G, c, Q_s, Q_t, k) be an instance of CCS-R. Let $c\colon V(G) \mapsto \{1, \ldots, k'\}$, where $k' \leq k$. We construct a graph H as follows. For each $1 \leq i \leq k'$, we add a vertex d_i and connect d_i to all the vertices in $c^{-1}(i)$. Next, for each $1 \leq i \leq k'$, we add a pendant vertex x_i (i.e., $\{d_i, x_i\}$ is an edge). Let $D = \{d_1, \ldots, d_{k'}\}$. We output $(H, Q_s \cup D, Q_t \cup D, k + k')$ as the new CDS-R instance.

▶ **Lemma 3.7.** *If G is a d-degenerate graph then H is a $(d+1)$-degenerate graph.*

Proof. For each vertex $v \in V(G)$, $d_H(v) = d_G(v) + 1$. Thus, after removing $V(G)$ and $\{x_i\colon 1 \leq i \leq k'\}$, the remaining graph is edgeless. ◀

It is easy to verify that for any reconfiguration sequence $Q_s = R_1, \ldots, R_\ell = Q_t$ of the instance (G, c, Q_s, Q_t, k) of CCS-R, $Q_s \cup D = R_1 \cup D, \ldots, R_\ell \cup D = Q_t \cup D$ is a reconfiguration sequence of the instance $(H, Q_s \cup D, Q_t \cup D, k + k')$ of CDS-R. Now we prove the reverse direction.

▶ **Lemma 3.8.** *If $(H, Q_s \cup D, Q_t \cup D, k + k')$ is a* yes*-instance then (G, c, Q_s, Q_t, k) is a* yes*-instance.*

Proof. Notice that the set D is contained in any connected dominating set of H. Moreover for any minimal connected dominating set Z in H, $Z \cap \{x_i\colon 1 \leq i \leq k'\} = \emptyset$, $H[Z \setminus D]$ is connected, and $Z \setminus D$ contains a vertex from $c^{-1}(i)$ for all $1 \leq i \leq k'$ (recall that G is a subgraph of H). Therefore, by deleting D from each set in a reconfiguration sequence of $(H, Q_s \cup D, Q_t \cup D, k + k')$, we get a valid reconfiguration sequence of (G, c, Q_s, Q_t, k). This completes the proof. ◀

Thus, by Theorem 3.6, we have the following theorem.

▶ **Theorem 3.9.** *CDS-R parameterized by $k + \ell$ is* W[1]*-hard on 5-degenerate graphs.*

4 Fixed-parameter tractability on planar graphs

This section is devoted to proving that CDS-R under TAR parameterized by k is fixed-parameter tractable on planar graphs. In fact, we show that the problem admits a polynomial kernel. Recall that a kernel for a parameterized problem $\mathcal{Q}$ is a polynomial-time algorithm that computes for each instance (I, k) of $\mathcal{Q}$ an equivalent instance (I', k') with $|I'| + k' \leq f(k)$ for some computable function f. The kernel is polynomial if the function f is polynomial. We prove that for every instance (G, S, T, k) of CDS-R, with G planar, we can compute in polynomial time an instance (G', S, T, k) where $|V(G')| \leq h(k)$ for some polynomial h, G' planar, and where there exists a reconfiguration sequence under TAR from S to T in G (using at most k tokens) if and only if such a sequence exists in G'.

Our approach is as follows. We first compute a small *domination core* for G, that is, a set of vertices that captures exactly the domination properties of G for dominating sets of sizes not larger than k. While the classification of interactions with the domination core would suffice to solve Dominating Set Reconfiguration, additional difficulties arise for the connected variant. In a second step we use planarity to identify large subgraphs that have very simple interactions with the domination core and prove that they can be replaced by constant size gadgets such that the reconfiguration properties of G are preserved.

4.1 Domination cores

▶ **Definition 4.1.** *Let G be a graph and let $k \geq 1$ be an integer. A* k-domination core *is a subset $C \subseteq V(G)$ of vertices such that every set $X \subseteq V(G)$ of size at most k that dominates C also dominates G.*

It is not difficult to see that Dominating Set is fixed-parameter tractable on all graphs that admit a k-domination core of size at most $f(k)$ that is computable in time $g(k) \cdot n^c$, for any computable functions f, g and constant c. This approach was first used (implicitly) in [4] to solve Distance-r Dominating Set on nowhere dense graph classes. In case k is the size of a minimum (distance-r) dominating set, one can establish the existence of a linear size k-domination core on classes of bounded expansion [6] (including the class of planar graphs) and a polynomial size (in fact an almost linear size) k-domination core on nowhere dense graph classes [8, 21]. If k is not minimum, there exist classes of bounded expansion such that a k-domination core must have at least quadratic size [7]. The most general graph classes that admit k-domination cores are given in [9]. Moreover, Dominating Set Reconfiguration and Distance-r Dominating Set Reconfiguration are fixed-parameter tractable on all graphs that admit small (distance-r) k-domination cores [22, 30].

▶ **Lemma 4.2.** *There exists a polynomial h such that for all $k \geq 1$, every planar graph G admits a polynomial-time computable k-domination core of size at most $h(k)$.*

The lemma is implied by Theorem 1.6 of [21] by the fact that planar graphs are nowhere dense. We want to stress again that the polynomial size of the k-domination core results from the fact that k may not be the size of a minimum dominating set, if k is minimum we can find a linear size core. Explicit bounds on the degree of the polynomial can be derived from [27, 29], but we refrain from doing so to not disturb the flow of ideas.

The following lemma is immediate from the definition of a k-domination core.

▶ **Lemma 4.3.** *If C is a k-domination core and D is a dominating set of size at most k that contains a vertex set $W \subset D$ such that $N[D] \cap C = N[D \setminus W] \cap C = C$, then $D \setminus W$ is also a dominating set.*

▶ **Definition 4.4.** *Let G be a graph and let $A \subseteq V(G)$. The* projection *of a vertex $v \in V(G) \setminus A$ into A is the set $N(v) \cap A$. If two vertices u, v have the same projection into A we write $u \sim_A v$.*

Obviously, the relation $\sim_A$ is an equivalence relation. The following lemma is folklore, one possible reference is [10].

▶ **Lemma 4.5.** *Let G be a planar graph and let $A \subseteq V(G)$. Then there exists a constant c such that there are at most $c \cdot |A|$ different projections to A, that is, the equivalence relation $\sim_A$ has at most $c \cdot |A|$ equivalence classes.*

4.2 Reduction rules

Let G be an embedded planar graph. We say that a vertex v *touches* a face f if v is drawn inside f or belongs to the boundary of f or is adjacent to a vertex on the boundary of f. We fix two connected dominating sets S and T of size at most k. We will present a sequence of lemmas, each of which implies a polynomial-time computable reduction rule that allows us to transform G to a planar graph G' that inherits its embedding from G, with $S, T \subseteq V(G')$ and that has the same reconfiguration properties with respect to S and T as G. To not overload

notation, after stating a lemma with a reduction rule, we assume that the reduction rule is applied until this is no longer possible and call the resulting graph again G. We also assume that whenever one or more of our reduction rules are applicable, then they are applied in the order presented. We will guarantee that S and T will always be connected dominating sets of size at most k, hence, after each application of a reduction rule, we can recompute a k-domination core in polynomial time. This yields only polynomial overhead and allows us to assume that we always have marked a k-domination core C of size at most $h(k)$ as described in Lemma 4.2. This allows us to state the lemmas as if G and C are fixed. Without loss of generality we assume that C contains S and T.

▶ **Definition 4.6.** *A set $W \subseteq V(G) \setminus C$ of vertices is* irrelevant *if there is a reconfiguration sequence from S to T in G if and only if there is a reconfiguration sequence from S to T in $G - W$.*

▶ **Definition 4.7.** *Let $u, v \in V(G)$ be distinct vertices. We call the set $D(u, v) := (N(u) \cap N(v)) \cup \{u, v\}$ the* diamond *induced by u and v. We call $|N(u) \cap N(v)|$ the* thickness *of $D(u, v)$.*

▶ **Lemma 4.8.** *If G contains a diamond $D(u, v)$ of thickness greater than $3k$, then at least one of u or v must be occupied by a token in every reconfiguration sequence from S to T.*

Proof. Assume $S = S_1, \ldots, S_t = T$ is a reconfiguration sequence from S to T and $u, v \notin S_i$ for some $1 \leq i \leq t$. Then every $s \in S_i$ can dominate at most 3 vertices of $N(u) \cap N(v)$:

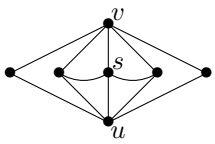


Figure 7 A vertex $s \in S_i$ can dominate at most 3 vertices of $N(u) \cap N(v)$.

otherwise u, v, s together with 3 vertices of $N(u) \cap N(v)$ different from u, v and s would form a complete bipartite graph $K_{3,3}$. ◀

▶ **Lemma 4.9.** *If G contains a diamond $D(u, v)$ of thickness greater than $3k$, then we can remove all internal edges in $D(u, v)$, i.e., edges with both endpoints in $N(u) \cap N(v)$.*

Proof. Assume $S = S_1, \ldots, S_t = T$ is a reconfiguration sequence from S to T. According to Lemma 4.8, for each $1 \leq i \leq t$, $S_i \cap \{u, v\} \neq \emptyset$. Hence all vertices of $N(u) \cap N(v)$ are always dominated by at least one of u or v, say by u. Moreover, having tokens on more than one vertex of $N(u) \cap N(v)$ will never create connectivity via internal edges that is not already there via edges incident on u. In other words, for any connected dominating set S of G, if an edge yz is used for connectivity, where $y, z \in N(u) \cap N(v)$, then the edge can be replaced by the path yuz or the path yvz (depending on which of u or v is in S). ◀

As described earlier, we now apply the reduction rule of Lemma 4.9 until this is no longer possible, and name the resulting graph again G. As we did not make use of the properties of a k-domination core in the lemma, it is sufficient to recompute a k-domination core C after applying the reduction rule exhaustively. In the following it may be necessary to recompute it after each application of a reduction rule. We will not mention these steps explicitly in the following.

▶ **Lemma 4.10 (⋆).** *If G contains a diamond $D(u, v)$ of thickness greater than $4|C| + 3k + 1$ then G contains an irrelevant vertex.*

We may in the following assume that G does not contain diamonds of thickness greater than $4|C| + 3k + 1$.

▶ **Corollary 4.11.** *If a vertex $v \in V(G)$ has degree greater than $(4|C| + 3k + 1) \cdot k$, then the token on v is never lifted throughout a reconfiguration sequence.*

Proof. Assume $S = S_1, \ldots, S_t = T$ is a reconfiguration sequence from S to T in G and assume there is S_i with $v \notin S_i$. The dominating set S_i has at most k vertices and must dominate $N(v)$. Hence, there must be one vertex $u \in S_i$ that dominates at least a $1/k$ fraction of $N(v)$, which is larger than $4|C| + 3k + 1$. Then there is a diamond $D(u, v)$ of thickness greater than $4|C| + 3k + 1$, which does not exist after application of the reduction rule of Lemma 4.10. ◀

According to Corollary 4.11, the only vertices that can have high degree after applying the reduction rules are vertices that are never lifted throughout a reconfiguration sequence. This gives rise to another reduction rule that is similar to the rule of Lemma 4.9.

▶ **Lemma 4.12.** *Assume v is a vertex of degree greater than $(4|C| + 3k + 1) \cdot k$. Then we may remove all edges with both endpoints in $N(v)$.*

Proof. Let G' be the graph obtained from G by removing all edges with both endpoints in $N(v)$. We claim that reconfiguration between S and T is possible in G if and only if it is possible in G'. The fact that S and T are in fact connected dominating sets in G' is implied by the argument below.

Assume $S = S_1, \ldots, S_t = T$ is a reconfiguration sequence from S to T in G. We claim that the same sequence is a reconfiguration sequence in G'. According to Corollary 4.11, $v \in S_i$ for all $1 \leq i \leq t$. This implies that S_i is connected in G' for all $1 \leq i \leq t$, as all $x, y \in S_i$ that are no longer connected by an edge in G' but were connected in G are connected via a path of length 2 using the vertex v. It is also easy to see that S_i is a dominating set in G', as all vertices that are no longer dominated by $s \in S_i$ in G are still dominated by v. Observe that this in particular implies that S and T are connected dominating sets in G'. Vice versa, if $S = S_1, \ldots, S_t = T$ is a reconfiguration sequence from S to T in G', this is trivially also a reconfiguration sequence in G. ◀

The following reduction rule is obvious.

▶ **Lemma 4.13.** *If a vertex v has more than $k + 1$ pendant neighbours, i.e., neighbors of degree exactly one, then it suffices to retain exactly $k + 1$ of them in the graph.*

▶ **Lemma 4.14.** *There are at most $c|C| \cdot (4|C| + 3k + 1)$ vertices of $V(G) \setminus C$ that have 2 neighbours in C, where c is the constant of Lemma 4.5.*

Proof. According to Lemma 4.5 there are at most $c|C|$ different projections to C. Each projection class that has at least 3 representatives has size at most 2, as otherwise we would find a $K_{3,3}$ as a subgraph, contradicting the planarity of G. Consider a class with a projection of size 2 into C. Denote these two vertices of C by u and v. If this class has more than $4|C| + 3k + 1$ representatives, then $D(u, v)$ is a diamond of thickness greater than $4|C| + 3k + 1$, which cannot exist after exhaustive application of the reduction rule of Lemma 4.10. ◀

We now come to the description of our final reduction rule. Let D denote the set of vertices containing both C and all vertices of $V(G) \setminus C$ having at least two neighbors in C. In other words, $V(G) \setminus D$ contains all those vertices in $V(G) \setminus C$ that have exactly one neighbor in C. According to Lemma 4.14 at most $c|C| \cdot (4|C| + 3k + 1)$ vertices have two neighbors in C, hence $|D| \leq c|C| \cdot (4|C| + 3k + 1) + |C| =: p$.

▶ **Lemma 4.15** (⋆). *Assume there are two vertices u and v with degree greater than $4p + (4|C| + 3k + 1) \cdot k + 1$. Let $\mathcal{P}$ be a maximum set of vertex-disjoint paths of length at least 2 that run between u and v using only vertices in $V(G) \setminus D$. If $|\mathcal{P}| > 4p + (4|C| + 3k + 1) \cdot k + 1$, then there is G' such that the instances (G, S, T, k) and (G', S, T, k) are equivalent, G' is planar, and $|V(G')| < |V(G)|$.*

▶ **Theorem 4.16.** *CDS-R under TAR parameterized by k admits a polynomial kernel on planar graphs.*

Proof. Our kernelization algorithm starts by computing (in polynomial time) a k-domination core C of size at most $h(k)$ as described in Lemma 4.2. Without loss of generality we assume that C contains S and T. After each application of a reduction rule, we recompute the core, giving a polynomial blow-up of the running time. We are left to prove that each reduction rule can be implemented in polynomial time and that we end up with a polynomial number of vertices. It is clear that the reduction rules of Lemma 4.10, Lemma 4.12 and Lemma 4.13 can easily be implemented in polynomial time. The reduction rule of Lemma 4.15 is slightly more involved, however, we can use a standard maximum-flow algorithm to compute in polynomial time a maximum set of vertex-disjoint paths in a subgraph of G. It remains to bound the size of G. Recall that we call D the set of all vertices C and of all vertices of $V(G) \setminus C$ that have at least 2 neighbors in C. It follows from Lemma 4.14 that D has size at most $c|C| \cdot (4|C| + 3k + 1) + |C| =: p$, where c is the constant of Lemma 4.5. We are left to bound the number of vertices in $V(G) \setminus C$ having exactly one neighbour in C (recall that each vertex in $V(G) \setminus C$ has at least one neighbour in $S \cup T \subseteq C$).

Let $p' = (4p + (4|C| + 3k + 1) \cdot k + 1) \cdot (4|C| + 3k + 1) \cdot k + k + 1$, which is still a polynomial in k. Towards a contradiction, assume that there exists an equivalence class Q in $\sim_C$ with a projection of size one containing more than p' vertices. Let $u \in C$ denote the projection of the aforementioned class. Due to Lemma 4.13, we know that at most $k + 1$ of the vertices in Q are pendant, i.e., adjacent to only u in G. Since we cannot apply the reduction rule of Lemma 4.12 any more, we know that there are no edges with both endpoints in Q. Hence, all but $k + 1$ vertices of Q must be adjacent to at least one other vertex in $V(G) \setminus C$. Let $R = N_G(Q) \setminus \{u\}$ denote this set of neighbours. No vertex in R can be adjacent to more than $4|C| + 3k + 1$ vertices of Q, as we cannot apply the reduction rule of Lemma 4.10. The vertices of R must be dominated by S, and cannot be dominated by u, as otherwise two neighbours of u would be connected. Hence, there is $v \in S$ different from u that dominates at least a $1/k$ fraction of R. This implies the existence of at least $4p + (4|C| + 3k + 1) \cdot k + 1$ vertex-disjoint paths of length at least 2 that run between u and v. But in this case, the reduction rule of Lemma 4.15 is applicable. Therefore, we conclude that Q cannot exist, obtaining a bound on the size of all equivalence classes of $\sim_C$, as needed. ◀

References

1 Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Connectedness of the graph of vertex-colourings. *Discrete Mathematics*, 308(56):913–919, 2008.

2 Bruno Courcelle, Johann A Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

3 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

4 Anuj Dawar and Stephan Kreutzer. Domination problems in nowhere-dense classes. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009*, pages 157–168, 2009.

5 Erik D. Demaine and Joseph O'Rourke. *Geometric folding algorithms - linkages, origami, polyhedra*. Cambridge University Press, 2007.

6 Pål Grønås Drange, Markus Sortland Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Felix Reidl, Fernando Sánchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar. Kernelization and sparseness: the case of dominating set. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016*, pages 31:1–31:14, 2016.

7 Eduard Eiben, Mithilesh Kumar, Amer E. Mouawad, Fahad Panolan, and Sebastian Siebertz. Lossy kernels for connected dominating set on sparse graphs. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*, pages 29:1–29:15, 2018.

8 Kord Eickmeyer, Archontia C. Giannopoulou, Stephan Kreutzer, O-joung Kwon, Michal Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Neighborhood complexity and kernelization for nowhere dense classes of graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 63:1–63:14, 2017.

9 Grzegorz Fabianski, Michal Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. Progressive algorithms for domination and independence. In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019*, pages 27:1–27:16, 2019.

10 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. *J. Comput. Syst. Sci.*, 84:219–242, 2017.

11 Sevag Gharibian and Jamie Sikora. Ground state connectivity of local hamiltonians. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming, ICALP 2015*, pages 617–628, 2015.

12 Parikshit Gopalan, Phokion G. Kolaitis, Elitza N. Maneva, and Christos H. Papadimitriou. The connectivity of Boolean satisfiability: computational and structural dichotomies. *SIAM Journal on Computing*, 38(6):2330–2355, 2009.

13 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *Journal of the ACM (JACM)*, 64(3):17, 2017.

14 Ruth Haas and Karen Seyffarth. The k-dominating graph. *Graphs and Combinatorics*, 30(3):609–617, 2014.

15 Arash Haddadan, Takehiro Ito, Amer E. Mouawad, Naomi Nishimura, Hirotaka Ono, Akira Suzuki, and Youcef Tebbal. The complexity of dominating set reconfiguration. *Theor. Comput. Sci.*, 651:37–49, 2016. doi:10.1016/j.tcs.2016.08.016.

16 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12-14):1054–1065, 2011.

17 Takehiro Ito, Marcin Kamiński, and Erik D. Demaine. Reconfiguration of list edge-colorings in a graph. *Discrete Applied Mathematics*, 160(15):2199–2207, 2012.

18 Wm. Woolsey Johnson and William E. Story. Notes on the "15" puzzle. *American Journal of Mathematics*, 2(4):397–404, 1879.

19 Iyad A. Kanj and Ge Xia. Flip distance is in FPT time o(n+ k * cˆk). In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015*, pages 500–512, 2015.

20 Graham Kendall, Andrew J. Parkes, and Kristian Spoerer. A survey of NP-complete puzzles. *ICGA Journal*, pages 13–34, 2008.

21 Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Polynomial kernels and wideness properties of nowhere dense graph classes. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 1533–1545, 2017.

22 Daniel Lokshtanov, Amer E. Mouawad, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Reconfiguration on sparse graphs. *J. Comput. Syst. Sci.*, 95:122–131, 2018.

23 Anna Lubiw and Vinayak Pathak. Flip distance between two triangulations of a point set is NP-complete. *Comput. Geom.*, 49:17–23, 2015.

24 Amer E. Mouawad. *On Reconfiguration Problems: Structure and Tractability*. PhD thesis, University of Waterloo, 2015.

25 Amer E. Mouawad, Naomi Nishimura, Vinayak Pathak, and Venkatesh Raman. Shortest reconfiguration paths in the solution space of boolean formulas. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 985–996, 2015.

26 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. *Algorithmica*, 78(1):274–297, 2017.

27 Wojciech Nadara, Marcin Pilipczuk, Roman Rabinovich, Felix Reidl, and Sebastian Siebertz. Empirical evaluation of approximation algorithms for generalized graph coloring and uniform quasi-wideness. In *17th International Symposium on Experimental Algorithms, SEA 2018*, pages 14:1–14:16, 2018.

28 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.

29 Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. On the number of types in sparse graphs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 799–808. ACM, 2018.

30 Sebastian Siebertz. Reconfiguration on nowhere dense graph classes. *Electr. J. Comb.*, 25(3):P3.24, 2018.

31 Akira Suzuki, Amer E. Mouawad, and Naomi Nishimura. Reconfiguration of dominating sets. *Journal of Combinatorial Optimization*, 32(4):1182–1195, 2016.

32 Jan van den Heuvel. The complexity of change. *Surveys in combinatorics*, 409(2013):127–160, 2013.

On the Fine-Grained Parameterized Complexity of Partial Scheduling to Minimize the Makespan

Jesper Nederlof
Utrecht University, Algorithms and Complexity Group, The Netherlands
https://webspace.science.uu.nl/~neder003/
j.nederlof@uu.nl

Céline M. F. Swennenhuis
Eindhoven University of Technology, Combinatorial Optimization Group, The Netherlands
https://research.tue.nl/nl/persons/c%C3%A9line-swennenhuis
c.m.f.swennenhuis@tue.nl

Abstract

We study a natural variant of scheduling that we call *partial scheduling*: In this variant an instance of a scheduling problem along with an integer k is given and one seeks an optimal schedule where not all, but only k jobs, have to be processed.

Specifically, we aim to determine the fine-grained parameterized complexity of partial scheduling problems parameterized by k for all variants of scheduling problems that minimize the makespan and involve unit/arbitrary processing times, identical/unrelated parallel machines, release/due dates, and precedence constraints. That is, we investigate whether algorithms with runtimes of the type $f(k)n^{\mathcal{O}(1)}$ or $n^{\mathcal{O}(f(k))}$ exist for a function f that is as small as possible.

Our contribution is two-fold: First, we categorize each variant to be either in P, NP-complete and fixed-parameter tractable by k, or W[1]-hard parameterized by k. Second, for many interesting cases we further investigate the run time on a finer scale and obtain run times that are (almost) optimal assuming the Exponential Time Hypothesis. As one of our main technical contributions, we give an $\mathcal{O}(8^k k(|V| + |E|))$ time algorithm to solve instances of partial scheduling problems minimizing the makespan with unit length jobs, precedence constraints and release dates, where $G = (V, E)$ is the graph with precedence constraints.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Fixed-Parameter Tractability, Scheduling, Precedence Constraints

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.25

Related Version A full version of the paper is available at [22], https://arxiv.org/pdf/1912.03185.pdf.

Funding *Jesper Nederlof*: ERC project no. 617951. and no. 853234. and NWO project no. 024.002.003.
Céline M. F. Swennenhuis: NWO project no. 613.009.031b, ERC project no. 617951.

1 Introduction

Scheduling is one of the most central application domains of combinatorial optimization. In the last decades, huge combined effort of many researchers led to major progress on understanding the worst-case computational complexity of almost all natural variants of scheduling: By now, for most of these variants it is known whether they are NP-complete or not. Scheduling problems provide the context of some of the most classic approximation algorithms. For example, in the standard textbook by Shmoys and Williamson on approximation algorithms [28] a wide variety of techniques are illustrated by applications to scheduling problems. See also the standard textbook on scheduling by Pinedo [23] for more background.

Instead of studying approximation algorithms, another natural way to deal with NP-completeness is *Parameterized Complexity* (PC). While the application of general PC theory to the area of scheduling has still received considerably less attention than the approximation point of view, recently its study has seen explosive growth, as witnessed by a plethora of publications (e.g. [2, 13, 16, 20, 26, 27]). Additionally, many recent results and open problems can be found in a survey by Mnich and van Bevern [19], and even an entire workshop on the subject was recently held [18].

In this paper we advance this vibrant research direction with a complete mapping of how several standard scheduling parameters influence the parameterized complexity of minimizing the makespan in a natural variant of scheduling problems that we call *partial scheduling*. Next to studying the classical question of whether parameterized problems are in P, FPT or W-hard, we also follow the well-established modern perspective of "fine-grained" PC and aim at run times of the type $f(k)n^{\mathcal{O}(1)}$ or $n^{f(k)}$ for the smallest function f of parameter k.

Partial Scheduling. In many scheduling problems arising in practice, the set of jobs to be scheduled is not predetermined. We refer to this as *partial scheduling*. Partial scheduling is well-motivated from practice, as it arises naturally for example in the following scenarios:

1. Due to uncertainties a *close-horizon approach* may be employed and only few jobs out of a big set of jobs will be scheduled in a short but fixed time-window,
2. In freelance markets typically a large database of jobs is available and a freelancer is interested in selecting only a few of the jobs to work on,
3. The selection of the jobs to process may resemble other choices the scheduler should make, such as to outsource non-processed jobs to various external parties.

Partial scheduling has been previously studied in the equivalent forms of *maximum throughput scheduling* [24] (motivated by the first example setting above), *job rejection* [25], *scheduling with outliers* [12], *job selection* [8, 15, 29] and its special case *interval selection* [5].

In this paper, we conduct a rigorous study of the parameterized complexity of partial scheduling, parameterized by *the number of jobs to be scheduled*. We denote this number by k. While several isolated results concerning the parameterized complexity of partial scheduling do exist, this parameterization has (somewhat surprisingly) not been rigorously studied yet.[1] We address this and study the parameterized complexity of the (arguably) most natural variants of the problem. We fix as objective to minimize the makespan while scheduling at least k jobs, for a given integer k and study all variants with the following characteristics:

- 1 machine, identical parallel machines or unrelated parallel machines,
- release/due dates, unit/arbitrary processing times, and precedence constraints.

Note that a priori this amounts to $3 \times 2 \times 2 \times 2 \times 2 = 48$ variants.

1.1 Our Results

We give a classification of the parameterized complexity of these 48 variants. Additionally, for each variant that is not in P, we give algorithms solving them and lower bounds under ETH. To easily refer to a variant of the scheduling problem, we use the standard three-field notation by Graham et al. [11]. See Section 2 for an explanation of this notation. To accommodate our study of partial scheduling, we extend the $\alpha|\beta|\gamma$ notation as follows:

▶ **Definition 1.1.** *We let k-sched in the γ-field indicate that we only schedule k out of n jobs.*

[1] We compare the previous works and other relevant studied parameterization in the end of this section.

We study the fine-grained parameterized complexity of all problems $\alpha|\beta|\gamma$, where $\alpha \in \{1, P, R\}$, the options for β are all combinations for $r_j, prec, d_j, p_j = 1$, and γ is fixed to $\gamma = k\text{-sched}, C_{\max}$. Our results are explicitly enumerated in Table 1.

Table 1 The fine-grained parameterized complexity of partial scheduling, where γ denotes k-sched, $C_{\max}$ and S.I. abbreviates Subgraph Isomorphism. Since $p_j = 1$ implies that the machines are identical, the mentioned number of 48 combinations reduces to 40 different scheduling problems. The $\mathcal{O}^*$ notation omits factors polynomial in the input size.

		Problem Description	Parameterized Complexity in k	Result Type	Lower Bound under ETH		Run Time
					Excluded Run Time	Reduction from	
Precedence Relations	1	$1\|\text{prec}, p_j = 1\|\gamma$	P	**[A]**			$n^{\mathcal{O}(1)}$
	2	$1\|r_j, \text{prec}, p_j = 1\|\gamma$	P	**[A]**			$n^{\mathcal{O}(1)}$
	3	$1\|d_j, \text{prec}, p_j = 1\|\gamma$	W[1]-hard	**[B]**	$n^{o(k/\log k)}$	3-Coloring	$n^{\mathcal{O}(k)}$
	4	$1\|r_j, d_j, \text{prec}, p_j = 1\|\gamma$	W[1]-hard	**[B]**	$n^{o(k/\log k)}$	3-Coloring	$n^{\mathcal{O}(k)}$
	5	$P\|\text{prec}, p_j = 1\|\gamma$	FPT	**[C]**	$\mathcal{O}^*(2^{o(\sqrt{k \log k})})$	$P\|\text{prec}, p_j = 1\|C_{\max}$	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	6	$P\|r_j, \text{prec}, p_j = 1\|\gamma$	FPT	**[C]**	$\mathcal{O}^*(2^{o(\sqrt{k \log k})})$	$P\|\text{prec}, p_j = 1\|C_{\max}$	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	7	$P\|d_j, \text{prec}, p_j = 1\|\gamma$	W[1]-hard	**[B]**	$n^{o(k/\log k)}$	3-Coloring	$n^{\mathcal{O}(k)}$
	8	$P\|r_j, d_j, \text{prec}, p_j = 1\|\gamma$	W[1]-hard	**[B]**	$n^{o(k/\log k)}$	3-Coloring	$n^{\mathcal{O}(k)}$
	9	$1\|\text{prec}\|\gamma$	W[1]-hard	**[D]**	$n^{o(\sqrt{k})}$	k-Clique	$n^{\mathcal{O}(k)}$
	10	$1\|r_j, \text{prec}\|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	Partitioned S.I.	$n^{\mathcal{O}(k)}$
	11	$1\|d_j, \text{prec}\|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	Partitioned S.I.	$n^{\mathcal{O}(k)}$
	12	$1\|r_j, d_j, \text{prec}\|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	Partitioned S.I.	$n^{\mathcal{O}(k)}$
	13	$P\|\text{prec}\|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	Partitioned S.I.	$n^{\mathcal{O}(k)}$
	14	$P\|r_j, \text{prec}\|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	Partitioned S.I.	$n^{\mathcal{O}(k)}$
	15	$P\|d_j, \text{prec}\|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	Partitioned S.I.	$n^{\mathcal{O}(k)}$
	16	$P\|r_j, d_j, \text{prec}\|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	Partitioned S.I.	$n^{\mathcal{O}(k)}$
	17	$R\|\text{prec}\|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	Partitioned S.I.	$n^{\mathcal{O}(k)}$
	18	$R\|r_j, \text{prec}\|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	Partitioned S.I.	$n^{\mathcal{O}(k)}$
	19	$R\|d_j, \text{prec}\|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	Partitioned S.I.	$n^{\mathcal{O}(k)}$
	20	$R\|r_j, d_j, \text{prec}\|\gamma$	W[1]-hard	**[D]**	$n^{o(k/\log k)}$	Partitioned S.I.	$n^{\mathcal{O}(k)}$
No Precedence Relations	21	$1\|p_j = 1\|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	22	$1\|r_j, p_j = 1\|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	23	$1\|d_j, p_j = 1\|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	24	$1\|r_j, d_j, p_j = 1\|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	25	$P\|p_j = 1\|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	26	$P\|r_j, p_j = 1\|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	27	$P\|d_j, p_j = 1\|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	28	$P\|r_j, d_j, p_j = 1\|\gamma$	P	**[E]**			$n^{\mathcal{O}(1)}$
	29	$1\|\|\gamma$	P	**[F]**			$n^{\mathcal{O}(1)}$
	30	$1\|r_j\|\gamma$	P	**[F]**			$n^{\mathcal{O}(1)}$
	31	$1\|d_j\|\gamma$	P	**[F]**			$n^{\mathcal{O}(1)}$
	32	$1\|r_j, d_j\|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	Subset Sum	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	33	$P\|\|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	Subset Sum	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	34	$P\|r_j\|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	Subset Sum	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	35	$P\|d_j\|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	Subset Sum	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	36	$P\|r_j, d_j\|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	Subset Sum	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	37	$R\|\|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	Subset Sum	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	38	$R\|r_j\|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	Subset Sum	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	39	$R\|d_j\|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	Subset Sum	$\mathcal{O}^*(2^{\mathcal{O}(k)})$
	40	$R\|r_j, d_j\|\gamma$	FPT	**[G]**	$\mathcal{O}^*(2^{o(k)})$	Subset Sum	$\mathcal{O}^*(2^{\mathcal{O}(k)})$

The rows of Table 1 are lexicographically sorted on (i) precedence relations / no precedence relations, (ii) a single machine, identical machines or unrelated machines (iii) release dates and/or deadlines. Because their presence has a major influence on the character of the problem we stress the distinction between variants with and without *precedence constraints*.[2]

[2] A precedence constraint $a \prec b$ enforces that job a needs to be finished before job b can start.

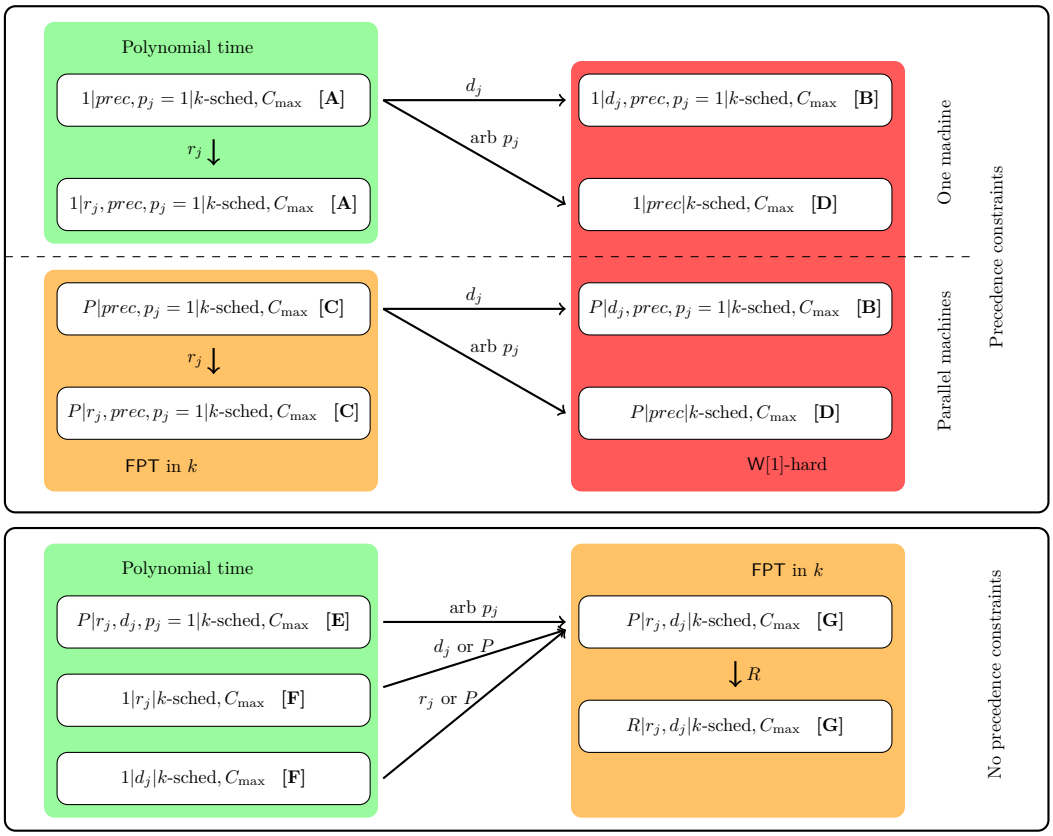


Figure 1 An illustration of the various result types as indicated in Table 1. Arrows indicate how a problem is generalized by another problem.

On a high abstraction level, our contribution is two-fold:

1. We present a *classification* of the complexity of all aforementioned variants of partial scheduling with the objective of minimizing the makespan. Specifically, we classify all variants to be either solvable in polynomial time, to be fixed-parameter tractable in k and NP-hard, or to be W[1]-hard.
2. For most of the studied variants we present both an algorithm and a lower bound that shows that our algorithm cannot be significantly improved unless the Exponential Time Hypothesis (ETH) fails.

Thus, while we completely answer a classical type of question in the field of Parameterized Complexity, we pursue in our second contribution a more modern and fine-grained understanding of the best possible run time with respect to the parameter k. For several of the studied variants, the lower bounds and algorithms listed in Table 1 follow relatively quickly. However, for many other cases we need substantial new insights to obtain (almost) matching upper and lower bounds on the runtime of the algorithms solving them. We have grouped the rows in *result types* [**A**]-[**G**] depending on our methods for determining their complexity.

1.2 Our new Methods

We now describe some of our most significant technical contributions for obtaining the various types (listed as [**A**]-[**G**] in Table 1) of results. Note that we skip some less interesting cases in this introduction; for a complete argumentation of all results from Table 1 we refer to

the full version of the paper. The main building blocks and logical implications to obtain the results from Table 1 are depicted in Figure 1. We now discuss these building blocks of Figure 1 in detail.

Precedence Constraints. Our main technical contribution concerns result type **[C]**. The simplest of the two cases, $P|\text{prec}, p_j = 1|k\text{-sched}, C_{\max}$, cannot be solved in $\mathcal{O}^*(2^{o(\sqrt{k \log k})})$ time assuming the Exponential Time Hypothesis and not in $2^{o(k)}$ unless sub-exponential time algorithms for the Biclique problem exist, due to reductions by Jansen et al. [14]. Our contribution lies in the following theorem that gives an upper bound for the more general of the two problems that matches the latter lower bound:

▶ **Theorem 1.2.** *$P|r_j, prec, p_j = 1|k\text{-sched}, C_{\max}$ can be solved in $\mathcal{O}(8^k k(|V| + |E|))$ time,[3] where $G = (V, E)$ is the precedence graph given as input.*

Theorem 1.2 will be proved in Section 3. The first idea behind the proof is based on a natural[4] dynamic programming algorithm indexed by anti-chains of the partial order naturally associated with the precedence constraints. However, evaluating this dynamic program naïvely would lead to an $n^{\mathcal{O}(k)}$ time algorithm, where n is the number of jobs.

Our key idea is to only compute a subset of the table entries of this dynamic programming algorithm, guided by a new parameter of an antichain called the *depth*. Intuitively, the depth of an antichain A indicates the number of jobs that can be scheduled after A in a feasible schedule without violating the precedence constraints.

We prove Theorem 1.2 by showing we may restrict attention in the dynamic programming algorithm to antichains of depth at most k, and by bounding the number of antichains of depth at most k indirectly by bounding the number of *maximal* antichains of depth at most k. We believe this methodology should have more applications for scheduling problems with precedence constraints.

Surprisingly, the positive result of Theorem 1.2 is in *stark contrast* with the seemingly symmetric case where only deadlines are present: Our next result, indicated as **[B]** in Figure 1 shows it is much harder:

▶ **Theorem 1.3.** *$P|d_j, prec, p_j = 1|k\text{-sched}, C_{\max}$ is $\mathsf{W}[1]$-hard, and cannot be solved in $n^{o(k/\log k)}$ time assuming the ETH.*

Theorem 1.3 is a consequence of a reduction outlined in Section 4. Note the $\mathsf{W}[1]$-hardness follows from a natural reduction from the k-Clique problem (presented originally by Fellows and McCartin [9]), but this reduction increases the parameter k to $\Omega(k^2)$ and would only exclude $n^{o(\sqrt{k})}$ time algorithms assuming the ETH. To obtain the tighter bound from Theorem 1.3, we instead provide a non-trivial reduction from the 3-Coloring problem based on a new selection gadget.

For result type **[D]**, we give a lower bound by a (relatively simple) reduction from Partitioned Subgraph Isomorphism in Theorem 4.6 and Corollary 4.7. Since it is conjectured that Partitioned Subgraph Isomorphism cannot be solved in $n^{o(k)}$ time assuming the ETH, our reduction is a strong indication that the simple $n^{\mathcal{O}(k)}$ time algorithm (see [22]) cannot be improved significantly in this case.

[3] We assume basic arithmetic operations with the release dates take constant time.
[4] A similar dynamic programming approach was also present in for example [7].

No Precedence Constraints. The second half of our classification concerns scheduling problems without precedence constraints, and is easier to obtain than the first half. Results **[E]**, **[F]** are consequences of a greedy algorithm and Moore's algorithm [21] that solves the problem $1||\sum_j U_j$ in $\mathcal{O}(n \log n)$ time. Notice that this also solves the problem $1|r_j|k\text{-sched}, C_{\max}$, by reversing the schedule and viewing the release dates as the deadlines. For result type **[G]** we show that a standard technique in parameterized complexity, the color coding method, can be used to get a $2^{\mathcal{O}(k)}$ time algorithm for the most general problem of the class, being $R|r_j, d_j|k\text{-sched}, C_{\max}$. All lower bounds on the run time of algorithms for problems of type **[G]** are by a reduction from Subset Sum, but for $1|r_j, d_j|k\text{-sched}, C_{\max}$ this reduction is slightly different.

1.3 Related Work

The interest in parameterized complexity of scheduling problems recently witnessed an explosive growth, resulting in e.g. a workshop [18] and a survey by Mnich and van Bevern [19] with a wide variety of open problems.

The parameterized complexity of partial scheduling parameterized by the number of processed jobs, or equivalently, the number of jobs "on time" was studied before: Fellows et al. [9] studied a problem called k-Tasks On Time that is equivalent to $1|d_j, prec, p_j = 1|k\text{-sched}, C_{\max}$ and showed that it is W[1]-hard when parameterized by k,[5] and FPT parameterized by k and the width of the partially ordered set induced by the precedence constraints. Van Bevern et al. [27] showed that the Job Interval Selection problem, where each job is given a set of possible intervals to be processed on, is FPT in k. Bessy et al. [2] consider partial scheduling with a restriction on the jobs called "Coupled-Task", and also remarked the current parameterization is relatively understudied.

Another related parameter is the number of jobs that are *not scheduled*, that also has been studied in several previous works [4, 9, 20]. For example, Mnich and Wiese [20] studied the parameterized complexity of scheduling problems with respect to the number of rejected jobs in combination with other variables as parameter. If n denotes the number of given jobs, this parameter equals $n - k$. The two parameters are somewhat incomparable in terms of applications: In some settings only few jobs out of many alternatives need to be scheduled, but in other settings rejecting a job is very costly and thus will happen rarely. However, a strong advantage of using k as parameter is in terms of its computational complexity: If the version of the problem with all jobs mandatory is NP-complete it is trivially NP-complete for $n - k = 0$, but it may still be FPT in k.

1.4 Organization of this paper

This paper is organized as follows: We start with some preliminaries in Section 2. In Section 3 we present the proof of Theorem 1.2, and in Section 4 we describe the reductions for result types **[B]** and **[D]**. In Section 5 we give the algorithm for result type **[G]** and in Section 6 we present a conclusion. The proofs of some Theorems and Lemma's of Section 3 are omitted. These are indicated with a † and the full proofs can be found in the full version of the paper ([22]). In that version, we also motivate all cases from Table 1.

[5] Our results **[C]** and **[D]** build on and improve this result.

2 Preliminaries: The three-field notation by Graham et al.

Throughout this paper we denote scheduling problems using the three-field classification by Graham et al. [11]. Problems are classified by parameters $\alpha|\beta|\gamma$. The α describes the machine environment. This paper uses $\alpha \in \{1, P, R\}$, indicating whether there are one (1), identical (P) or unrelated (R) parallel machines available. Here identical refers to the fact that every job takes a fixed amount of time process independent of the machine, and unrelated means a job could take different time to process per machine. The β field describes the job characteristics, which in this paper can be a combination of the following values: *prec* (precedence constraints), r_j (release dates), d_j (deadlines) and $p_j = 1$ (all processing times are 1). We assume without loss of generality that all release dates and deadlines are integers.

The γ field concerns the optimization criteria. A given schedule determines C_j, the completion time of job j, and U_j, the unit penalty which is 1 if $C_j > d_j$, and 0 if $C_j \leq d_j$. In this paper we use the following optimization criteria

- $C_{\max}$: minimize the makespan (i.e. the maximum completion time C_j of any job),
- $\sum_j U_j$: minimize the number of jobs that finish after their deadline,
- k-sched: maximize the number of processed jobs; in particular, process at least k jobs.

A schedule is said to be *feasible* if no constraints (deadlines, release dates, precedence constraints) are violated.

3 Result Type C: Precedence Constraints, Release Dates and Unit Processing Times

In this section we provide a fast algorithm for partial scheduling with release dates and unit processing times parameterized by the number k of scheduled jobs (Theorem 1.2). There exists a simple, but slow, algorithm with runtime $\mathcal{O}^*(2^{k^2})$ that already proves that this problem is FPT in k: This algorithm branches k times on jobs that can be processed next. If more than k jobs are available at a step, then processing these jobs greedily is optimal. Otherwise, we can recursively try to schedule all non-empty subsets of jobs to schedule next, and a $\mathcal{O}^*(2^{k^2})$ time algorithm is obtained via a standard (bounded search-tree) analysis. To improve on this algorithm, we present a dynamic programming algorithm based on table entries indexed by antichains in the precedence graph G describing the precedence relations. Such an antichain describes the maximal jobs already scheduled in a partial schedule. Our key idea is that, to find an optimal solution, it is sufficient to restrict our attention to a subset of all antichains. This subset will be defined in terms of the *depth* of an antichain. With this algorithm we improve the runtime to $\mathcal{O}(8^k k(|V| + |E|))$.

By binary search, we can restrict attention to a variant of the problem that asks whether there is a feasible schedule with makespan at most $C_{\max}$, for a fixed universal deadline $C_{\max}$.

Notation for Posets. Any precedence graph G is a directed acyclic graph and therefore induces a partial order $\prec$ on $V(G)$. Indeed, if there is a path from x to y, we let $x \preceq y$. An *antichain* is a set $A \subseteq V(G)$ of mutually incomparable elements. We say A is *maximal* if there is no antichain A' with $A \subset A'$. The set of *predecessors* of A is $\text{pred}(A) = \{x \in V(G) : \exists a \in A : x \preceq a\}$, and the the set of *comparables* of A is $\text{comp}(A) = \{x \in V(G) : \exists a \in A : x \preceq a \text{ or } x \succeq a\}$. Note $\text{comp}(A) = V(G)$ if and only if A is maximal.

An element $x \in V(G)$ is a *minimal* element if $x \preceq y$ for all $y \in \text{comp}(\{x\})$. An element $x \in V(G)$ is a *maximal* element if $x \succeq y$ for all $y \in \text{comp}(\{x\})$. Furthermore $\min(G) = \{x \mid x \text{ is a minimal element in } G\}$ and $\max(G) = \{x \mid x \text{ is a maximal element in } G\}$.

Notice that $\max(G)$ is exactly the antichain A such that $\text{pred}(A) = V(G)$. We denote the subgraph of G induced by S with $G[S]$. We may assume that $r_j < r_{j'}$ if $j \prec j'$ since job j' will be processed later than r_j in any schedule. To handle release dates we use the following:

▶ **Definition 3.1.** *Let G be a precedence graph. Then G^t is the precedence graph restricted to all jobs that can be scheduled on or before time t, i.e. all jobs with release date at most t.*

We assume $G = G^{C_{\max}}$, since all jobs with release date greater than $C_{\max}$ can be ignored.

The Algorithm. We now introduce our dynamic programming algorithm for $P|r_j, prec, p_j = 1|k\text{-sched}, C_{\max}$. Let m be the number of machines available. We start with defining the table entries. For a given antichain $A \subseteq V(G)$ and integer t we define

$$S(A,t) = \begin{cases} 1, & \text{if there exists a feasible schedule of makespan } t \text{ that processes } \text{pred}(A), \\ 0, & \text{otherwise.} \end{cases}$$

Computing the values of $S(A,t)$ can be done by trying all combinations of scheduling at most m jobs of A at time t and then checking whether all remaining jobs of $\text{pred}(A)$ can be scheduled in makespan $t-1$. To do so, we also verify that all the jobs in A actually have a release date at or before t. Formally, we have the following recurrence for $S(A,t)$:

▶ **Lemma 3.2.**

$$S(A,t) = (A \subseteq V(G^t)) \wedge \bigvee_{X \subseteq A : |X| \leq m} S(A', t-1) : A' = \max(\text{pred}(A) \setminus X).$$

Proof. If $A \not\subseteq V(G^t)$, then there is a job $j \in A$ with $r_j > t$. And thus $S(A,t) = 0$.

For any $X \subseteq A$, X is a set of maximal elements with respect to $G[\text{pred}(A)]$, and consists of pair-wise incomparable jobs, since A is an antichain. So, we can schedule all jobs from X at time t without violating any precedence constraints. Define $A' = \max(\text{pred}(A) \setminus X)$ as the unique antichain such that $\text{pred}(A) \setminus X = \text{pred}(A')$. If $S(A', t-1) = 1$ and $|X| \leq m$, we can extend the schedule of $S(A', t-1)$ by scheduling all X at time t. In this way we get a feasible schedule processing all jobs of $\text{pred}(A)$ before or at time t. So if we find such an X with $|X| \leq m$ and $S(A', t-1) = 1$, we must have $S(A,t) = 1$.

For the other direction, if for all $X \subseteq A$ with $|X| \leq m$, $S(A', t-1) = 0$, then no matter which set $X \subseteq A$ we try to schedule at time t, the remaining jobs cannot be scheduled before t. Note that only jobs from A can be scheduled at time t, since those are the maximal jobs. Hence, there is no feasible schedule and $S(A,t) = 0$. ◀

The above recurrence cannot be directly evaluated, since the number of different antichains of a graph can be big: there can be as many as $\binom{n}{k}$ different antichains with $|\text{pred}(A)| \leq k$, for example in the extreme case of an independent set. Even when we restrict our precedence graph to have out degree k, there could be k^k different antichains, for example in k-ary trees. To circumvent this issue, we restrict our dynamic programming algorithm only to a specific subset of antichains. To do this, we use the following new notion of the *depth* of an antichain.

▶ **Definition 3.3.** *Let A be an antichain. Define the depth (with respect to t) of A as*

$$d^t(A) = |\text{pred}(A)| + |\min(G^t - \text{comp}(A))|.$$

We also denote $d(A) = d^{C_{\max}}(A)$.

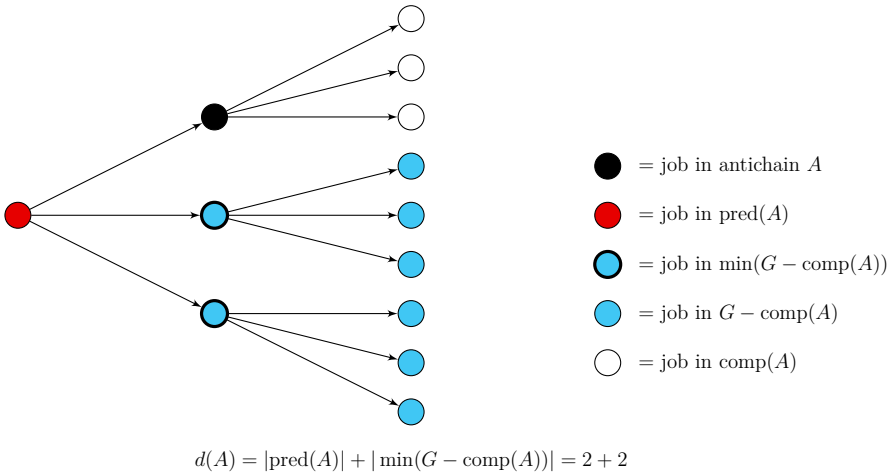


Figure 2 Example of an antichain and its depth in a perfect 3-ary tree. We see that $|\text{pred}(A)| = 2$, but $d(A) = 4$. If $k = 2$, the dynamic programming algorithm will not compute $S(A, t)$ since $d(A) > k$. The only antichains with depth ≤ 2 are the empty set and the root node r on its own as a set. Indeed $d(\emptyset) = d(\{r\}) = 1$. Note that for instances with $k = 2$, a feasible schedule may exist. If so, we will find that $R(\{r\}, 1) = 1$, which will be defined later. In this way, we can still find the antichain A as a solution.

The intuition behind this definition is that it quantifies the number of jobs that can be scheduled before (and including) A without violating precedence constraints. See Figure 2 for an example of an antichain and its depth. We restrict the dynamic programming algorithm to only compute $S(A, t)$ for A satisfying $d^t(A) \leq k$. This ensures that we do not go "too deep" into the precedence graph unnecessarily at the cost of a slow runtime.

Because of this restriction in the depth, it could happen that we check no antichains with k or more predecessors, while there are corresponding feasible schedules. It is therefore possible that for some antichains A with $d^t(A) > k$, there is a feasible schedule for all $\geq k$ jobs in pred(A) before time $C_{\max}$, but the value $S(A, C_{\max})$ will not be computed. To make sure we still find an optimal schedule, we also compute the following condition $R(A, t)$ for all $t \leq C_{\max}$ and antichains A with $d^t(A) \leq k$:

$$R(A,t) = \begin{cases} 1, & \text{if there exists a feasible schedule with makespan at most } C_{\max} \text{ that} \\ & \text{processes pred}(A) \text{ on or before } t \text{ and processes jobs from} \\ & \min(G - \text{pred}(A)) \text{ after } t, \text{ with a total of } k \text{ jobs processed,} \\ 0, & \text{otherwise.} \end{cases}$$

By definition of $R(A, t)$, if $R(A, t) = 1$ for any A and $t \leq C_{\max}$, then we find a feasible schedule that processes k jobs on time.[6] We show in [22] that $R(A, t)$ can be quickly computed:

▶ **Lemma 3.4** (†). *There is an $\mathcal{O}(|V|k + |E|)$ time algorithm* `fill(A, t)` *that, given an antichain A, integer t, and value $S(A, t)$, computes $R(A, t)$.*

The algorithm `fill(A, t)` checks if $S(A, t) = 1$ and if so, greedily schedules jobs from $\min(G - \text{pred}(A))$ after t in order of smallest release date. If $k - |\text{pred}(A)|$ jobs can be scheduled before $C_{\max}$, it returns "true" ($R(A, t) = 1$). Otherwise, it returns "false" ($R(A, t) = 0$).

[6] The reverse direction is more difficult and postponed to Lemma 3.6.

Combining all steps gives us the algorithm as described in Algorithm 1. It remains to bound its runtime and argue its correctness.

Algorithm 1 Algorithm for $P|\text{pred}, p_j = 1|k\text{-sched}, C_{\max}$.

```
foreach t = 1, ..., C_max do
    Enumerate all antichains A in G^t with d^t(A) ≤ k using Lemma 3.5
    foreach antichain A in G^t with d^t(A) ≤ k do
        Compute S(A, t) using Lemma 3.2
        if fill(S(A, t), A, t) then
            return TRUE
return FALSE
```

Runtime. To analyze the runtime of the dynamic programming algorithm, we need to bound the number of checked antichains. Recall that we only check antichains A with $d^t(A) \leq k$ for each time $t \leq C_{\max}$. We first analyze the number of antichains A with $d(A) \leq k$ in any graph and use this to upper bound the number of antichains checked at time t.

▶ **Lemma 3.5** (†). *For any t, there are at most 4^k antichains A with $d^t(A) \leq k$ in any precedence graph $G = (V, E)$, and they can be enumerated within $\mathcal{O}(4^k(|V| + |E|))$ time.*

Notice that to compute each $S(A, t)$, we look at a maximum of $\binom{k}{m} \leq 2^k$ different sets X. Computing the antichain A' such that $A' = \max(\text{pred}(A) \setminus X)$ takes $\mathcal{O}(|V| + |E|)$ time. After this computation, $R(A, t)$ is directly computed in $\mathcal{O}(|V|k + |E|)$ time. For each time $t \in \{1, ..., C_{\max}\}$, there are at most 4^k different antichains A for which we compute $S(A, t)$ and $R(A, t)$. Since $C_{\max} \leq k$, we therefore have total runtime of $\mathcal{O}(4^k k(2^k(|V| + |E|) + (|V|k + |E|)))$. Hence, Algorithm 1 runs in time $\mathcal{O}(8^k k(|V| + |E|))$.

Correctness of algorithm. To show that the algorithm described in Algorithm 1 indeed returns the correct answer, the following lemma is clearly sufficient:

▶ **Lemma 3.6** (†). *A feasible schedule for k jobs with makespan at most $C_{\max}$ exists if and only if $R(A, t) = 1$ for some $t \leq C_{\max}$ and antichain A with $d^t(A) \leq k$.*

To prove Lemma 3.6, we consider the schedule which corresponds to an antichain which has minimal depth. We then conclude that it either should be witnessed by some $R(A, t)$ or that there is another antichain with even smaller depth, which contradicts the assumption. The proof heavily relies on the intricacies of the definition of depth.

4 Result Types B and D: One Machine and Precedence Constraints

In this section we show that Algorithm 1 cannot be even slightly generalized further: if we allow job-dependent deadlines or non-unit processing times, the problem becomes W[1]-hard parameterized by k and cannot be solved in $n^{o(k/\log k)}$ time unless the ETH fails.

Job-dependent deadlines. The fact that combining precedence constraints with job-dependent deadlines makes the problem W[1]-hard, is a direct consequence from the fact that $1|prec, p_j = 1|\sum_j U_j$ is W[1]-hard, parameterized by $n - \sum_j U_j = k$ where n is the number of jobs [9]. It is important to notice that the notation of these problems implies that each job

can have its own deadline. Hence, we conclude from this that $1|d_j, prec, p_j = 1|k\text{-sched}, C_{\max}$ is W[1]-hard parameterized by k. This is a reduction from k-Clique and therefore we get a lower bound on algorithms for the problem of $n^{\Omega(\sqrt{k})}$. Based on the Exponential Time Hypothesis, we now sharpen this lower bound with a reduction from 3-Coloring:

▶ **Theorem 4.1.** *$1|d_j, prec, p_j = 1|k\text{-sched}, C_{\max}$ is W[1]-hard parameterized by k. Furthermore, there is no algorithm solving $1|d_j, prec, p_j = 1|k\text{-sched}, C_{\max}$ in $2^{o(n)}$ time where n is the number of jobs, assuming ETH.*

Proof. The proof will be a reduction from 3-Coloring, for which no $2^{o(|V|+|E|)}$ algorithm exists under the Exponential Time Hypothesis [6, pages 471-473]. Let the graph $G = (V, E)$ be the instance of 3-Coloring with $|V| = n'$ and $|E| = m'$. We then create the following instance for $1|d_j, prec, p_j = 1|k\text{-sched}, C_{\max}$.

- For each vertex $v_i \in V$, create 6 jobs:
  - v_i^1, v_i^2 and v_i^3 with deadline $d_{v_i} = i$,
  - w_i^1, w_i^2 and w_i^3 with deadline $d_{w_i} = n' + 2m' + 1 - i$,

  add precedence constraints $v_i^1 \prec w_i^1$, $v_i^2 \prec w_i^2$ and $v_i^3 \prec w_i^3$. These jobs represent which color for each vertex will be chosen (if v_i^1 and w_i^1 are processed, vertex i gets color 1).
- For each edge $e_j \in E$, create 12 jobs:
  - e_j^{12}, e_j^{13}, e_j^{21}, e_j^{23}, e_j^{31} and e_j^{32} with deadline $d_{e_j} = n' + j$,
  - f_j^{12}, f_j^{13}, f_j^{21}, f_j^{23}, f_j^{31} and f_j^{32} with deadline $d_{f_j} = n' + m' + 1 - j$,

  add precedence constraints $e_j^{ab} \prec f_j^{ab}$. These jobs represent what the colors of the endpoints of an edge will be. So if the jobs e_j^{ab} and f_j^{ab} are processed for $e = \{u, v\}$, then vertex u has color a and vertex v has color b. Since the endpoints should have different colors, the jobs e_j^{aa} and f_j^{aa} do not exist.
- For each e_j^{ab} with $e = \{u, v\}$ add the precedence constraints $u^a \prec e_j^{ab}$ and $v^b \prec e_j^{ab}$.
- Set $C_{\max} = k = 2n' + 2m'$.

We now prove that the created instance is a yes instance if and only if the original 3-Coloring instance is a yes instance. Assume that there is a 3-coloring of the graph $G = (V, E)$. Then there is also a feasible schedule: For each vertex v_i with color a, process the jobs v_i^a and w_i^a at their respective deadlines. For each edge $e_j = \{u, v\}$ with u colored a and v colored b, process the jobs e_j^{ab} and f_j^{ab} exactly at their respective deadlines. Notice that because it is a 3-coloring, each edge has endpoints of different colors, so these jobs exist. Also note that no two jobs were processed at the same time. Exactly $2n' + 2m'$ jobs were processed before time $2n' + 2m'$. Furthermore, no precedence constraints were violated.

For the other direction, assume that we have a feasible schedule in our created instance of $1|d_j, prec, p_j = 1|k\text{-sched}, C_{\max}$. Let $\mathcal{V}_i = \{v_i^1, v_i^2, v_i^3\}$, $\mathcal{W}_i = \{w_i^1, w_i^2, w_i^3\}$, and let $\mathcal{E}_j = \{e_j^{12}, e_j^{13}, e_j^{21}, e_j^{23}, e_j^{31}, e_j^{32}\}$ and $\mathcal{F}_j = \{f_j^{12}, f_j^{13}, f_j^{21}, f_j^{23}, f_j^{31}, f_j^{32}\}$. We show by induction on i that out of each of the sets $\mathcal{V}_i$, $\mathcal{W}_i$, $\mathcal{E}_j$ and $\mathcal{F}_j$, exactly one job was scheduled at its deadline.

Since we have a feasible schedule, at time $2m' + 2n'$ one of the jobs of $\mathcal{W}_1$ must be scheduled, since they are the only jobs with a deadline greater than $2n + 2m - 1$. However, if w_1^a was scheduled at time $2m' + 2n'$, then the job v_1^a must be processed at time 1 because of precedence constraints and since its deadline is 1. Note, that no other jobs from $\mathcal{V}_1$ and $\mathcal{W}_1$ can be processed, due to their deadlines and precedence constraints.

Now assume that all sets $\mathcal{V}_1, ..., \mathcal{V}_{i-1}, \mathcal{W}_1, ..., \mathcal{W}_{i-1}$ have exactly one job scheduled at their respective deadline, and no more can be processed. Since we have a feasible schedule, one job should be scheduled at time $2n' + 2m' - (i - 1)$. However, since no more jobs from $\mathcal{W}_1, ..., \mathcal{W}_{i-1}$ can be scheduled, the only possible jobs are from $\mathcal{W}_i$ since they are the only other jobs with a deadline greater than $2n' + 2m' - i$. However, if w_i^a was scheduled at

time $2n' + 2m' - (i-1)$, then the job v_i^a must be processed at time i because of precedence constraints, its deadline at i and because at times $1, ..., i-1$ other jobs had to be processed. Also, no other job from $\mathcal{V}_i$ can be processed in the schedule, since they all have deadline i. As a consequence, no other jobs from $\mathcal{W}_1$ can be processed, as they are restricted to precedence constraints. So the statement holds for all set $\mathcal{V}_i$ and $\mathcal{W}_i$. In the exact same way, one can conclude the same about all sets $\mathcal{E}_j$ and $\mathcal{F}_j$.

Because of this, we see that each job and each vertex have received a color from the schedule. They must form a 3-coloring, because a job from $\mathcal{E}_j$ could only be processed if the two endpoints got two different colors. Hence the 3-Coloring instance is a yes instance.

As $k = 2n' + 2m'$ we therefore conclude there is no $2^{o(n)}$ algorithm under the ETH. ◀

Note that this bound significantly improves the old lower bound of $2^{\Omega(\sqrt{n})}$ implied by the the reduction from k-Clique reduction: Since $k \leq n$, Theorem 4.1 implies that

▶ **Corollary 4.2.** *Assuming ETH, there is no algorithm solving* $1|d_j, prec, p_j = 1|k\text{-}sched, C_{\max}$ *in* $n^{o(k/\log(k))}$ *where* n *is the number of jobs.*

Non-unit processing times. We show that having non-unit processing times combined with precedence constraints make the problem W[1]-hard even on one machine. The proof of Theorem 4.3 heavily builds on the reduction from k-Clique to k-Tasks On Time by Fellows and McCartin [9].

▶ **Theorem 4.3.** $1|prec|k\text{-}sched, C_{\max}$ *is* W[1]*-hard, parameterized by* k*.*

Proof. The proof is a reduction from k-Clique. We start with $G = (V, E)$, an instance of k-Clique. For each vertex $v \in V$, create a job j_v with $p_{j_v} = 2$. For each edge $e \in E$, create a job j_e with $p_{j_e} = 1$. Now for each edge (u, v), add the following two precedence relations: $j_u \prec j_e$ and $j_v \prec j_e$, so before one can process a job associated with an edge, both jobs associated with the endpoints of that edge need to be finished. Now let $k' = k + \frac{1}{2}k(k-1)$ and $C_{\max} = 2k + \frac{1}{2}k(k-1)$. We will now prove that $1|prec|k'\text{-sched}, C_{\max}$ is a yes instance if and only of k-Clique is a yes instance.

Assume that the k-Clique instance is a yes instance, then process first the k jobs associated with the vertices of the k-clique. Next process the $\frac{1}{2}k(k-1)$ jobs associated with the edges of the k-clique. In total, $k + \frac{1}{2}k(k-1) = k'$ jobs are now processed with a makespan of $2k + \frac{1}{2}k(k-1)$. Hence, the instance of $1|prec|k'\text{-sched}, C_{\max}$ is a yes instance.

For the other direction, assume $1|prec|k'\text{-sched}, C_{\max}$ to be a yes instance, so we have found a feasible schedule. For any feasible schedule, if one schedules l jobs associated with vertices, then at most $\frac{1}{2}l(l-1)$ jobs associated with edges can be processed, because of the precedence constraints. However, because $k' = k + \frac{1}{2}k(k-1)$ jobs were done in the feasible schedule before $C_{\max} = 2k + \frac{1}{2}k(k-1)$, at most k jobs associated with vertices can be processed, because they have processing time of size 2. Hence, we can conclude that exactly k vertex-jobs and $\frac{1}{2}k(k-1)$ edge-jobs were processed. Hence, there were k vertices connected through $\frac{1}{2}k(k-1)$ edges, which is a k-clique. ◀

The proofs of Theorem 4.6 and Corollary 4.7 are reductions from Partitioned Subgraph Isomorphism. Let $P = (V', E')$ be a "pattern" graph, $G = (V, E)$ be a "target" graph, and $\chi : V \to V'$ a "coloring" of the vertices of G with elements from P. A *χ-colorful P-subgraph of G* is a mapping $\varphi : V' \to V$ such that (1) for each $\{u, v\} \in E'$ it holds that $\{\varphi(u), \varphi(v)\} \in E$ and (2) for each $u \in V'$ it holds that $\chi(\varphi(u)) = u$. If χ and G are clear from the context they may be omitted in this definition.

▶ **Definition 4.4** (Partitioned Subgraph Isomorphism). *Given graphs $G = (V, E)$ and $P = (V', E')$, $\chi : V \to V'$. Determine whether there is a χ-colorful P-subgraph of G.*

▶ **Theorem 4.5** (Marx [17]). *Partitioned Subgraph Isomorphism cannot be solved in $n^{o(|E'|/\log|E'|)}$ time assuming the Exponential Time Hypothesis (ETH).*

We will now reduce Partitioned Subgraph Isomorphism to $1|\text{prec}, r_j|k\text{-sched}, C_{\max}$.

▶ **Theorem 4.6.** *$1|prec, r_j|k\text{-}sched, C_{\max}$ cannot be solved in $n^{o(k/\log k)}$ time assuming the Exponential Time Hypothesis (ETH).*

Proof. Let $G = (V, E)$, $P = (V', E')$ and $\chi : V \to V'$. We will write $V' = \{1, \ldots, s\}$. Define for $i = 0, \ldots, s$ the following important time stamps:

$$t_i := \sum_{j=1}^{i} 3^{s+1-j}.$$

Construct the following jobs for the instance of the $1|\text{prec}, r_j|k\text{-sched}, C_{\max}$ problem:

- For $i = 1, \ldots, s$:
  - For each vertex $v \in V$ such that $\chi(v) = i$, create a job j_v with processing time $p(j_v) = 3^{s+1-i}$ and release date t_{i-1}.
- For each $(v, w) \in E$ such that $(\chi(v), \chi(w)) \in E'$, create a job $j_{v,w}$ with $p(j_{v,w}) = 1$ and release date t_s. Add precedence constraints $j_v \prec j_{v,w}$ and $j_w \prec j_{v,w}$.

Then ask whether there exists a solution to the scheduling problem for $k = s + |E'|$ with makespan $C_{\max} \leq t_s + |E'|$.

Let the Partitioned Subgraph Isomorphism instance be a yes-instance and let $\varphi : V(P) \to V(G)$ be a colorful P-subgraph. We claim the following schedule is feasible:

- For $i = 1, \ldots, s$:
  - Process $j_{\varphi(i)}$ at its release date t_{i-1}.
- Process for each $(i, i') \in E'$ the job $j_{\varphi(i),\varphi(i')}$ somewhere in the interval $[t_s, t_s + |E'|]$.

Notice that all jobs are indeed processed after their release date and that in total there are $k = s + |E'|$ processed before $C_{\max} \leq t_s + |E'|$. Furthermore, all precedence constraints are respected as any edge job is processed after both its predecessors. Also, the edge jobs $e^{\varphi(i),\varphi(i')}$ must exist, as $\varphi(P)$ is a properly colored P-subgraph. Therefore, we can conclude that indeed this schedule is feasible.

For the other direction, assume that there is a solution to the created instance of $1|\text{prec}, r_j|k\text{-sched}, C_{\max}$. Define $J_i = \{j_v : \chi(v) = i\}$. We will first prove that at most 1 job from each set J_i can be processed in a feasible schedule. To do this, we first prove that at most 1 job from each set J_i can be processed before t_s. Any job in J_i has release date $t_{i-1} = \sum_{j=1}^{i-1} 3^{s+1-j}$. Therefore, there is only $t_s - t_{i-1} = \sum_{j=i}^{s} 3^{s+1-j}$ time left to process the jobs from J_i before time t_s. However, the processing time of any job in J_i is 3^{s+1-i}, and since $2 \cdot 3^{s+1-i} > \sum_{j=i}^{s} 3^{s+1-j}$, at most 1 job from J_i can be processed before t_s. Since all jobs not in some J_i have their release date at t_s, at most s jobs are processed at time t_s. Thus at time t_s, there are $|E'|$ time unit left to process $|E'|$ jobs, because of the choice of k and makespan. Hence the only way to get a feasible schedule is to process exactly one job from each set J_i at its respective release date and process exactly $|E'|$ edge jobs after t_s.

Let v^i be the vertex, such that j_v was processed in the feasible schedule with color i. We will show that $\varphi : V(P) \to V(G)$, defined as $\varphi(i) = v^i$, is a function such that $\varphi(P)$ is a properly colored P-subgraph of G. Hence, we are left to prove that for each $(i, i') \in E'$,

the edge $(\varphi(i), \varphi(i')) \in E$, i.e. that for each $(i, i') \in E'$, the job $j_{\varphi(i),\varphi(i')}$ was processed. Because only the vertex jobs $j_{\varphi(1)}, j_{\varphi(2)}, \ldots, j_{\varphi(s)}$ were processed, the precedence constraints only allow for edge jobs $j_{\varphi(i),\varphi(i')}$ to be processed. We created edge job $j_{v,w}$ if and only if $(v, w) \in E$ and $(\chi(v), \chi(w)) \in E'$, hence the $|E'|$ edge jobs have to be exactly the edge jobs $j_{\varphi(i),\varphi(i')}$ for $(i, i') \in E'$. Therefore, we proved indeed that $\varphi(P)$ is a colorful P-subgraph of G.

Notice that $k = s + |E'| \leq 3|E'|$ as we may assume the number of vertices in P is at most $2|E'|$. Hence the given bound follows. ◀

▶ **Corollary 4.7.** *$2|prec|k\text{-}sched, C_{\max}$ cannot be solved in $n^{o(k/\log k)}$ time assuming the Exponential Time Hypothesis (ETH).*

Proof. We can use the same idea for the reduction from Partitioned Subgraph Isomorphism as in the proof of Theorem 4.6, except for the release dates, as they are not allowed in this type of scheduling problem. To simulate the release dates, we use the second machine as a release date machine, meaning that we will create a job for each upcoming release date and will require these new jobs to be processed. More formally: For $i = 1, \ldots, s$, create a job j_{r_i} with processing time 3^{s+1-i} and precedence constraints $j_{r_i} \prec j$ for any job j that had release date t_i in the original reduction. Furthermore let $j_{r_i} \prec j_{r_{i+1}}$. Then we add $|E'|$ jobs j' with processing time 1 and with precedence relations $j_{r_s} \prec j'$. We then ask whether there exists a feasible schedule with $k = 2s + 2|E'|$ and with makespan $t_s + |E'|$. All newly added jobs are required in any feasible schedule and therefore, all other arguments from the previous reduction also hold. Finally, note that k is again linear in $|E'|$. ◀

5 Result Type G: k-scheduling without Precedence Constraints

The problem $P|k\text{-sched}|C_{\max}$, cannot be solved in $2^{o(k)}$ time assuming the ETH by a reduction to Subset Sum. We show that the problem is fixed-parameter tractable with a matching run time in k, even in the case of unrelated machines, release dates and deadlines, denoted by $R|r_j, d_j, k\text{-sched}|C_{\max}$.

▶ **Theorem 5.1.** *$R|r_j, d_j, k\text{-}sched|C_{\max}$ is fixed-parameter tractable in k and can be solved in $\mathcal{O}^*((2e)^k k^{\mathcal{O}(\log k)})$ time.*

Proof. We give an algorithm that solves any instance of $R|r_j, d_j, k\text{-sched}|C_{\max}$ within $\mathcal{O}^*((2e)^k k^{\mathcal{O}(\log k)})$ time. The algorithm is a randomized algorithm that can be de-randomized using the color coding method, as described by Alon et al. [1]. The algorithm first (randomly) picks a coloring $c : \{1, ..., n\} \to \{1, ..., k\}$, so each job is given one of the k available colors. We then compute whether there is a feasible colorful schedule, i.e. a feasible schedule that processes exactly one job of each color. If this colorful schedule can be found, then it is possible to schedule at least k jobs before $C_{\max}$.

Given a coloring c, we compute whether there exists a colorful schedule in the following way. Define for $1 \leq i \leq m$ and $X \subseteq \{1, ..., k\}$:

$$B_i(X) = \text{minimum makespan of all schedules on machine } i \text{ processing } |X| \text{ jobs, each from a different color in } X.$$

Clearly $B_i(\emptyset) = 0$, and all values $B_i(X)$ can be computed in $\mathcal{O}(2^k n)$ time using the following:

▶ **Lemma 5.2.** *Let $\min\{\emptyset\} = \infty$. Then*

$$B_i(X) = \min_{l \in X} \min_{j : c(j) = l} \{C_j = \max\{r_j, B_i(X \setminus \{l\})\} + p_{ij} : C_j \leq d_j\}.$$

Proof. In a schedule on one machine with $|X|$ jobs using all colors from X, one job should be scheduled as last, defining the makespan. So for all possible jobs j, we compute what the minimal end time would be if j was scheduled at the end of the schedule. This j cannot start before its release date or before all other colors are scheduled. ◀

Next, define for $1 \leq i \leq m$ and $X \subseteq [k]$, $A_i(X)$ to be 1 if $B_i(X) \leq C_{\max}$, and to be 0 otherwise. So $A_i(X) = 1$ if and only if $|X|$ jobs, each from a different color of X, can be scheduled on machine i before $C_{\max}$. A colorful feasible schedule exists if and only if there is some partition $X_1, ..., X_m$ of $\{1, .., k\}$ such that $\Pi_{i=1}^{m} A_i(X_i) = 1$. The *subset convolution* of two functions is defined as $(A_i * A_{i'})(X) = \sum_{Y \subseteq X} A_i(Y)A_{i'}(X \setminus Y)$. Then $\Pi_{i=1}^{m} A_i(X_i) = 1$ if and only if $(A_1 * \cdots * A_m)(\{1, ..., k\}) > 0$. The value of $(A_1 * \cdots * A_m)(\{1, ..., k\}) > 0$ can be computed in $2^k k^{\mathcal{O}(1)}$ time using fast subset convolution [3].

An overview of the randomized algorithm is given in Algorithm 2. If the k jobs that are processed in an optimal solution are all in different colors, the algorithm outputs true. By standard analysis, k jobs are all assigned different colors with probability at least $1/e^k$, and thus e^k independent trials to boost the error probability of the algorithm to at most $1/2$.

Algorithm 2 Algorithm for solving $R|r_j, d_j, k\text{-sched}|C_{\max}$.

```
For a given coloring c:
foreach i = 1, ..., m do
    foreach X ⊆ {1, .., k} in order of increasing size do
        Compute B_i(X) using Lemma 5.2.
        Set A_i(X) = 1 if B_i(X) ≤ C_max, set A_i(X) = 0 otherwise.
Compute (A_1 * ··· * A_m)({1, ..., k}) using fast subset convolution [3].
if (A_1 * ··· * A_m)({1, ..., k}) > 0 then
    return TRUE
```

By using the standard methods by Alon et al. [1], Algorithm 2 can be derandomized. ◀

6 Concluding Remarks

We classify all studied variants of partial scheduling parameterized by the number of jobs to be scheduled to be either in P, NP-complete and fixed-parameter tractable by k, or W[1]-hard parameterized by k. Our main technical contribution is an $\mathcal{O}(8^k k(|V| + |E|))$ time algorithm for $P|r_j, \text{prec}, p_j = 1|k\text{-sched}, C_{\max}$.

In a fine-grained sense, the cases we left open are cases 3-20 from Table 1. We believe in fact algorithms in rows 5-6 and 10-20 are optimal: An $n^{o(k)}$ time algorithm for any case from result type **[C]** or **[D]** would imply either a $2^{o(n)}$ time algorithm for Biclique or an $n^{o(k)}$ time algorithm for Partitioned Subgraph Isomorphism, which both would be surprising. It would be interesting to see whether for any of the remaining cases with precedence constraints and unit processing times a "sub-exponential" time algorithm exists.

A related case is $P3|\text{prec}, p_j = 1|C_{max}$ (where $P3$ denotes three machines). It is a famously hard open question (see e.g. [10]) whether this can be solved in polynomial time, but maybe it is doable to try to solve this question in sub-exponential time, e.g. $2^{o(n)}$?

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

2 Stéphane Bessy and Rodolphe Giroudeau. Parameterized complexity of a coupled-task scheduling problem. *Journal of Scheduling*, 22(3):305–313, 2019.

3 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 67–74. ACM, 2007.

4 Hans L. Bodlaender and Michael R. Fellows. W[2]-hardness of precedence constrained k-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995.

5 Julia Chuzhoy, Rafail Ostrovsky, and Yuval Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. *Mathematics of Operations Research*, 31(4):730–738, 2006.

6 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

7 Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. Scheduling partially ordered jobs faster than 2^n. *Algorithmica*, 68(3):692–714, 2014. doi:10.1007/s00453-012-9694-7.

8 Joonyup Eun, Chang Sup Sung, and Eun-Seok Kim. Maximizing total job value on a single machine with job selection. *Journal of the Operational Research Society*, 68(9):998–1005, 2017.

9 Michael R. Fellows and Catherine McCartin. On the parametric complexity of schedules to minimize tardy tasks. *Theoretical Computer Science*, 298(2):317–324, 2003.

10 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

11 Ron L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5(2):287–326, 1979.

12 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Danny Segev. Scheduling with outliers. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, volume 5687 of *Lecture Notes in Computer Science*, pages 149–162. Springer, 2009.

13 Danny Hermelin, Matthias Mnich, and Simon Omlor. Single machine batch scheduling to minimize the weighted number of tardy jobs. *CoRR*, 2019. arXiv:1911.12350.

14 Klaus Jansen, Felix Land, and Maren Kaluza. Precedence scheduling with unit execution time is equivalent to parametrized biclique. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 329–343. Springer, 2016.

15 Christos Koulamas and Shrikant S. Panwalkar. A note on combined job selection and sequencing problems. *Naval Research Logistics*, 60(6):449–453, 2013.

16 Christophe Lenté, Matthieu Liedloff, Ameur Soukhal, and Vincent T'kindt. Exponential algorithms for scheduling problems, 2014.

17 Dániel Marx. Can you beat treewidth? *Theory of Computing*, 6:85–112, 2010.

18 Nicole Megow, Matthias Mnich, and Gerhard Woeginger. Lorentz Workshop 'Scheduling Meets Fixed-Parameter Tractability', 2019.

19 Matthias Mnich and René van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 2018.

20 Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1-2):533–562, 2015.

21 J. Michael Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1):102–109, 1968.

22 Jesper Nederlof and Céline Swennenhuis. Parameterized complexity of partial scheduling. *arXiv preprint*, 2019. arXiv:1912.03185.

23 Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.

24 Jirí Sgall. Open problems in throughput scheduling. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 2–11, 2012.

25 Dvir Shabtay, Nufar Gaspar, and Moshe Kaspi. A survey on offline scheduling with rejection. *Journal of Scheduling*, 16(1):3–28, 2013.

26 René van Bevern, Robert Bredereck, Laurent Bulteau, Christian Komusiewicz, Nimrod Talmon, and Gerhard J. Woeginger. Precedence-constrained scheduling problems parameterized by partial order width. In *Discrete Optimization and Operations Research - 9th International Conference, DOOR 2016, Vladivostok, Russia, September 19-23, 2016, Proceedings*, pages 105–120, 2016.

27 René van Bevern, Matthias Mnich, Rolf Niedermeier, and Mathias Weller. Interval scheduling and colorful independent sets. *Journal of Scheduling*, 18(5):449–469, 2015.

28 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

29 Bibo Yang and Joseph Geunes. A single resource scheduling problem with job-selection flexibility, tardiness costs and controllable processing times. *Computers & Industrial Engineering*, 53(3):420–432, 2007.

On the Parameterized Complexity of Maximum Degree Contraction Problem

Saket Saurabh
The Institute Of Mathematical Sciences, HBNI, Chennai, India
University of Bergen, Norway
saket@imsc.res.in

Prafullkumar Tale
CISPA - Helmholtz Center for Information Security, Saarbrücken, Germany
prafullkumar.tale@cispa.saarland

Abstract

In the Maximum Degree Contraction problem, input is a graph G on n vertices, and integers k, d, and the objective is to check whether G can be transformed into a graph of maximum degree at most d, using at most k edge contractions. A simple brute-force algorithm that checks all possible sets of edges for a solution runs in time $n^{\mathcal{O}(k)}$. As our first result, we prove that this algorithm is asymptotically optimal, upto constants in the exponents, under Exponential Time Hypothesis (ETH).

Belmonte, Golovach, van't Hof, and Paulusma studied the problem in the realm of Parameterized Complexity and proved, among other things, that it admits an FPT algorithm running in time $(d+k)^{2k} \cdot n^{\mathcal{O}(1)} = 2^{\mathcal{O}(k \log(k+d))} \cdot n^{\mathcal{O}(1)}$, and remains NP-hard for every constant $d \geq 2$ (Acta Informatica (2014)). We present a different FPT algorithm that runs in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$. In particular, our algorithm runs in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, for every fixed d. In the same article, the authors asked whether the problem admits a polynomial kernel, when parameterized by $k+d$. We answer this question in the negative and prove that it does not admit a polynomial compression unless NP $\subseteq$ coNP/*poly*.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Graph Contraction Problems, FPT Algorithm, Lower Bound, ETH, No Polynomial Kernel

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.26

Related Version Full version: `http://arxiv.org/abs/2009.11793`.

Funding *Saket Saurabh*: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 819416), and Swarnajayanti Fellowship (No DST/SJF/MSA01/2017-18).
Prafullkumar Tale: This research is a part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement SYSTEMATICGRAPH (No. 725978).

Acknowledgements We want to thank the anonymous reviewers for their valuable feedback.

1 Introduction

For any graph class $\mathcal{H}$, the $\mathcal{H}$-Modification problem takes as input a graph G and an integer k, and asks whether one can make at most k modifications in G such that the resulting graph is in $\mathcal{H}$. These types of modification problems are one of the central problems in graph theory and have received a considerable attention in algorithm design. With appropriate choice of $\mathcal{H}$ and allowed modification operations, $\mathcal{H}$-Modification can encapsulate well studied problems like Vertex Cover, Chordal Completion, Cluster Editing, Hadwinger Number, etc. Some natural and well-studied graph modification operations are vertex

deletion, edge deletion, edge addition, and edge contraction. The focus of the vast majority of papers on graph modification problems has been to the first three operations. Consider an example of $\mathcal{H}_{\leq d}$-Modification problem where $\mathcal{H}_{\leq d}$ is the collection of all graphs that has maximum degree at most d. If allowed modification operation is vertex deletion then we know the problem as Bounded Degree Deletion (BDD) and if it is edge contraction then as Maximum Degree Contraction (MDC). The complexity of BDD and several of its variants has been extensively studied [7, 9, 10, 13, 15, 16, 18, 25, 31] whereas, to the best of our knowledge, only [8] addressed MDC. In this article, we enhance our understanding of the second problem and answer an open question stated in [8].

The *contraction* of edge uv in simple graph G deletes vertices u and v from G, and replaces them by a new vertex, which is made adjacent to vertices that were adjacent to either u or v. For a set of edges F in $E(G)$, we denote the graph obtained from G by contracting all edges in F by G/F. In the $\mathcal{H}$-Contraction problem, an input is a graph G and an integer k, and the aim is to decide whether there is a set F of at most k edges in G such that G/F is in $\mathcal{H}$. Early papers by Watanabe et al. [33, 34] and Asano and Hirata [6] showed that $\mathcal{H}$-Contraction is NP-Hard for simple graph classes like trees, paths, stars, etc. Brouwer proved that it is NP-Hard even to decide whether a graph can be contracted to a path of length four [11]. Note that this problem admits a simple polynomial time algorithm if we consider any other modification operation. This has been a recurring theme in graph modification problems. For the same target graph class, edge contraction problem tends to more difficult than their counterparts where modification operation is vertex/edge addition/deletion. This difficulty is evident even in the realm of the Parameterized Complexity and Exact Exponential Algorithms.

In Parameterized Complexity, $\mathcal{H}$-Contraction problems are studied with the number of edges allowed to contract, k, as parameter. Heggernes et al. [24] proved that if $\mathcal{H}$ is the set of acyclic graphs then $\mathcal{H}$-Contraction is FPT but does not admit a polynomial kernel unless NP $\subseteq$ coNP/*poly*. The vertex deletion version of the problem, known as Feedback Vertex Set, admits a polynomial kernel. Series of papers studied the parameterized complexity for various graph classes like generalization and restrictions of trees [1, 3], cactus [26], bipartite graphs [21, 23], planar graphs [20], grids [32], cliques [12], split graphs [4], chordal graphs [29], bi-cliques [30], degree constrained graph classes [8, 19], etc. Krithika et al. [27] and Gunda et al. [22] studied $\mathcal{H}$-Contraction problems from the lenses of FPT approximation and lossy kernelization. Agarwal et al. [2] broke the 2^n-barrier for Path Contraction whereas Fomin et al. [17] showed that brute-force algorithms for Hadwinger Number problem and various other $\mathcal{H}$-Contraction problem are optimal under ETH.

Belmonte et al. [8] studied the parameterized complexity of $\mathcal{H}$-Contraction for three different classes $\mathcal{H}$: the class of graphs with maximum degree at most d, the class of d-regular graphs, and the class of d-degenerate graphs. They classified the parameterized complexity of all three problems with respect to the parameters k, d, and $d + k$. The first problem, also known as MDC, is defined as follows.

Maximum Degree Contraction	**Parameter:** $k + d$
Input: Graph G, integers k, d	
Question: Does there exist a subset F of $E(G)$ of size at most k such that every vertex in G/F has degree at most d?	

The authors proved that MDC is FPT when parameterized by $k + d$, W[2]-Hard when parameterized by k (even when restricted to split graphs), and para-NP-Hard when parameterized by d. Note that the problem is trivially solvable in polynomial time when $d \leq 1$ and NP-Hard for every constant $d \geq 2$.

Consider brute-force algorithm for MDC that given an instance (G, k, d), where graph G has n vertices, enumerates all subsets of edges of size at most k in G and for each subset contracts all edges in it to check whether the resulting graph has degree at most d. This algorithm runs in time $n^{\mathcal{O}(k)}$. Our first results states that this algorithm is optimal, up to constants in the exponents, under ETH.

▶ **Theorem 1.** *Unless ETH fails, there is no algorithm that given any instance (G, k, d) of* Maximum Degree Contraction *runs in time $n^{o(k)}$ and correctly determines whether it is a* Yes *instance.*

Belmonte et al. [8] presented an FPT algorithm for MDC that runs in time $(d+k)^{2k} \cdot n^{\mathcal{O}(1)}$. As for any non-trivial instance $d + k$ is smaller than n, we can conclude that there is no algorithm that given any instance (G, k, d) of MDC runs in time $(d + k)^{o(k)} \cdot n^{\mathcal{O}(1)}$ and correctly determines whether it is a Yes instance, unless ETH fails.

We remark that that the lower bound in Theorem 1 does not hold when d is a fixed constant and not a part of input. Hence, it is possible that MDC admits an algorithm that runs in time $k^{o(k)} \cdot n^{\mathcal{O}(1)}$ for a constant value of d. Belmonte et al. [8] proved that MDC problem admits linear vertex kernels on connected graphs when $d = 2$. This linear kernel leads to an FPT algorithm[1] running in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$. This hints that it is possible to design a better FPT algorithm for small values of d. Our second result shows that this is indeed the case.

▶ **Theorem 2.** *There is an algorithm that given an instance (G, k, d) of* Maximum Degree Contraction *runs in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ and correctly determines whether it is a* Yes *instance.*

We note that the reduction used in [8] to prove that MDC is NP-Hard for any constant $d \geq 2$ implies that there is no $2^{o(dk)}$ algorithm for this problem.

Next, we look at the kernelization of MDC. Belmonte et al. [8] left it as an open question to determine whether MDC admits a polynomial kernel when parameterized by $k + d$. Our last result answers this question in negative.

▶ **Theorem 3.** *Unless* NP $\subseteq$ coNP/*poly, Maximum Degree Contraction, parameterized by $k + d$, does not admit a polynomial compression.*

It is known that the Bounded Degree Deletion problem admits a kernel with $\mathcal{O}(d^3k)$ vertices [16]. Hence, $\mathcal{H}_{\leq d}$-Modification is another example for which changing the modification operations from vertex deletion to edge contraction changes the compressibility drastically.

Due to space constraints, we omit formal proofs of Theorem 1 and 3. **These proofs can be found in the full version of the paper.** In Section 2, we present some preliminaries. In Section 3, we give a reduction from $(k \times k)$-Permutation Independent Set to MDC. We present an FPT algorithm using universal sets and branching techniques in Section 4. In Section 5, we present a sketch of reduction from Red Blue Dominating Set to MDC. We conclude this article with an open question in Section 6.

2 Preliminaries

For a positive integer q, we denote set $\{1, 2, \ldots, q\}$ by $[q]$.

[1] The algorithm colors vertices in the reduced instance with two colors and contracts each connected component in the colored subgraphs.

Graph Theory. In this article, we consider simple graphs with a finite number of vertices. For an undirected graph G, sets $V(G)$ and $E(G)$ denote its set of vertices and edges, respectively. Unless otherwise specified, we use n to denote the number of vertices in the input graph G. We denote an edge with two endpoints u, v as (u, v). Two vertices u, v in $V(G)$ are *adjacent* to each other if there is an edge (u, v) in $E(G)$. The open neighborhood of a vertex v, denoted by $N_G(v)$, is the set of vertices adjacent to v and its degree $\deg_G(v)$ is $|N_G(v)|$. The closed neighborhood of a vertex v, denoted by $N_G[v]$, is the set $N(v) \cup \{v\}$. We omit the subscript in the notation for neighborhood and degree if the graph under consideration is clear. For a subset S of $V(G)$, we define $N[S] = \bigcup_{v \in S} N[v]$ and $N(S) = N[S] \setminus S$. For a subset F of edges, a subset of vertices $V(F)$ denotes the collection of endpoints of edges in F. We say a set of edges F *spans* a set of vertices S if $S \subseteq V(F)$. For a subset S of $V(G)$, we denote the graph obtained by deleting S from G by $G - S$ and the subgraph of G induced on the set S by $G[S]$. For two subsets S_1, S_2 of $V(G)$, edge set $E(S_1, S_2)$ denotes the edges with one endpoint in S_1 and another one in S_2. We say S_1, S_2 are adjacent if $E(S_1, S_2)$ is non empty. For an integer q, a q-coloring of graph G is a function $\phi : V(G) \to [q]$. A *proper coloring* of G is a q-coloring ϕ of $V(G)$ for some integer q such that for any edge (u, v), $\phi(u) \neq \phi(v)$. There is a proper coloring of the graph with $\Delta(G) + 1$ many colors which can found in polynomial time. A set of vertices S is said to be *independent set* if no two vertices in S are adjacent to each other. A set of edges F is called *matching* if no two edges in F share an endpoint. A graph is called *connected* if there is a path between every pair of distinct vertices. A subset S of $V(G)$ is said to be a *connected set* if $G[S]$ is connected. A *spanning tree* of a connected graph is its connected acyclic subgraph, which includes all the vertices of the graph.

Graph Contraction. The *contraction* of an edge uv in G deletes vertices u and v from G, and adds a new vertex which is adjacent to vertices that were adjacent to either u or v. This process does not introduce self-loops or parallel edges. The resulting graph is denoted by G/e. For a graph G and edge $e = uv$, we formally define G/e in the following way: $V(G/e) = (V(G) \cup \{w\}) \setminus \{u, v\}$ and $E(G/e) = \{xy \mid x, y \in V(G) \setminus \{u, v\}, xy \in E(G)\} \cup \{wx \mid x \in N_G(u) \cup N_G(v)\}$. Here, w is a new vertex. An edge contraction reduces the number of vertices in a graph by exactly one. Several edges might disappear because of one edge contraction. For a subset of edges F in G, graph G/F denotes the graph obtained from G by contracting each connected component in the sub-graph $G' = (V(F), F)$ to a vertex.

We now formally define a contraction of graph G to another graph H.

▶ **Definition 4** (Graph Contraction). *A graph G is said to be* contractible *to graph H if there is a function $\psi : V(G) \to V(H)$ such that following properties hold.*

1. *For any vertex h in $V(H)$, set $W(h) := \{v \in V(G) \mid \psi(v) = h\}$ is not empty and graph $G[W(h)]$ is connected.*
2. *For any two vertices h, h' in $V(H)$, edge hh' is present in H if and only if $E(W(h), W(h'))$ is not empty.*

We say graph G is contractible to H via mapping ψ. For a vertex h in H, set $W(h)$ is called a *witness set* associated with or corresponding to h. We define the *H-witness structure* of G, denoted by $\mathcal{W}$, as a collection of all witness sets. Formally, $\mathcal{W} = \{W(h) \mid h \in V(H)\}$. A witness structure $\mathcal{W}$ is a partition of vertices in G. If a *witness set* contains more than one vertex, then we call it *big* witness set, otherwise it is *small* witness set.

If graph G has a H-witness structure, then graph H can be obtained from G by a series of edge contractions. For a fixed H-witness structure, let F be the union of spanning trees of all witness sets. By convention, the spanning tree of a singleton set is the empty set. To obtain

graph H from G, it is sufficient to contract edges in F. Hence, $H = G/F$. For a G/F-witness structure $\mathcal{W}$ of G, there is a unique function $\psi : V(G) \to V(G/F)$ corresponding to it. We say graph G is *k-contractible* to H if the cardinality of F is at most k. In other words, H can be obtained from G by at most k edge contractions.

Maximum Degree Contraction. In this subsection, we state an observation related to MDC. We say a set of edges F is a *solution* to instance (G, k, d) if the number of edges in F is at most k and the maximum degree of graph G/F is at most d. The number of edges that we are allowed to contract, k, is also called *solution size*. The following observation specifies how a solution behaves locally.

▶ **Observation 5.** *Consider a* Yes *instance (G, k, d) of MDC and let v be a vertex of degree at least $d+1$ in G. Then, for any solution F to (G, k, d), there are at least two vertices in $N[v]$ that are in the same witness set in the G/F-witness structure of G.*

Proof. Let G is contractible to a graph G/F, via mapping ψ. Assume, for the sake of contradiction, that no two vertices in $N[v]$ are in the same witness set. This implies $|N[v]| = |\psi(N[v])|$, where $\psi(N_G[v]) = \bigcup_{u \in N_G[v]} \psi(u)$. As $\psi(N_G[v]) \subseteq N_{G/F}(\psi(v))$ and $|N[v]| > d+1$, vertex $\psi(v)$ is adjacent with $d+1$ or more vertices in G/F. This contradicts the fact that the maximum degree of vertices in G/F is at most d. Hence, our assumption was wrong and there are at least two vertices in $N[v]$ that are in some big-witness set in G/F-witness structure of G. ◀

3 A Lower Bound for the Algorithm

We present a reduction from $(k \times k)$-Permutation Independent Set (PIS) problem to Maximum Degree Contraction problem. In the $(k \times k)$-PIS problem we are given a graph H on a vertex set $[k] \times [k]$. In other words, the vertex set is formed by a $k \times k$ table. We denote vertices in the table by $v[i, j]$ for $1 \leq i, j \leq k$. The question is whether there exists an independent set X in H that contains exactly one vertex from each row and each column of the table. In other words, for every $i, j \in [k]$ there is exactly one element of X that has i on the first coordinate and j on the second coordinate. Note that without loss of generality we may assume that each row and each column of the table forms an independent set.

Reduction. The reduction accepts an instance, say (H, k), of $(k \times k)$-Permutation Independent Set as an input. Here, H is a graph with vertex set formed by a $k \times k$ table. The reduction modifies a copy of the graph H in the following way.

- It adds a vertex corresponding to each row in the table and makes it adjacent with all vertices in that row. Let $R = \{r_1, r_2, \ldots, r_k\}$ be the set of vertices corresponding to rows.
- It adds a vertex corresponding to each column in the table and makes it adjacent with all vertices in that column. Let $C = \{c_1, c_2, \ldots, c_k\}$ be the set of vertices corresponding to columns.
- It adds set $S = \{s_1, s_2, \ldots, s_k\}$ of k vertices. For every i in $[k]$, it makes s_i adjacent with every vertex in $V(H) \cup C$ and with r_i.
- For every vertex r_i in R, it adds k^2 pendant vertices and makes them adjacent with r_i.
- For every vertex c_j in C, it adds $(k^2 - k + 1)$ pendant vertices and makes them adjacent with c_j.

See Figure 1 for an illustration. Let G be the graph obtained from a copy of graph H with the above modifications. The algorithm returns $(G, k, k^2 + k)$ as instance of MDC.

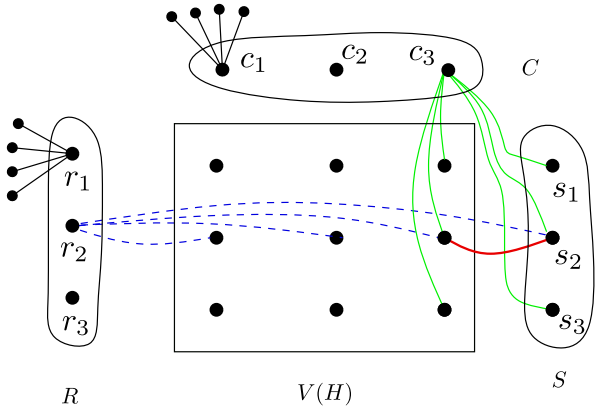


Figure 1 Dotted (blue) lines and thin (green) lines show the adjacency of vertices in R and C, respectively. Contracting the thick (red) edge $(v[2,3], s_2)$ represents selecting vertex $v[2,3]$ into the independent set. For the sake of clarity, we do not depict all edges present in the graph.

We present intuition of the proof of correctness. We describe how a solution, if it exists, to (G, k, d) leads to a solution to (H, k). We hope that this will also provide some intuition as to how a solution to (H, k) leads to a solution to (G, k, d). Note that S, C, R are independent sets in G. Every vertex in $R \cup C \cup S$ has degree $d+1$ and every vertex in $V(G) \setminus (R \cup C \cup S)$ has degree strictly less than d. We first argue that any solution for (G, k, d) can only contain edges in $E(G)$ that have one endpoint in $V(H)$ and another endpoint in S. Then, we prove that for every $i \in [k]$ a solution must pick an edge incident to some vertex in the i^{th} row and on s_i to reduce the degree of vertex r_i. We prove a similar statement for every column. Hence, for every $i \in [k]$, a solution contains an edge of the form $(v[i, j], s_i)$ for some $j \in [k]$. As there are at most k edges in a solution, every edge is of this form. For $i_1, i_2, j_1, j_2 \in [k]$, let $(v[i_1, j_1], s_{i_1})$ and $(v[i_2, j_2], s_{i_2})$ be two edges in a solution. We argue that if $(v[i_1, j_1], v[i_2, j_2])$ is an edge in G (and hence in H) then degrees of vertices obtained by contracting $(v[i_1, j_1], s_{i_1})$ and $(v[i_2, j_2], s_{i_2})$ are more than d. As this is true for any two arbitrary edges in the solution, their endpoints in $V(H)$ form an independent set in H.

We present a formal proof of correctness of this reduction in the full version of the paper. This reduction combined with the fact that unless ETH fails, $(k \times k)$-Permutation Independent Set can not be solved in time $k^{o(k)}$ [28] proves Theorem 1.

4 A Different FPT Algorithm

In this section, we present a different FPT algorithm for Maximum Degree Contraction. We introduce a variation of the problem called Labeled-Maximum Degree Contraction (Labeled-MDC). We present an FPT algorithm for Labeled-MDC and use it as a subroutine to present an FPT algorithm for MDC.

Informally, an instance of Labeled-MDC is an instance of MDC along with a labeling of vertices in the graph. Every vertex has a red or blue label. We are only interest in a solution that satisfies the following properties: (1) every edge has red labelled endpoints, and (2) for any red-labelled maximal connected component, a solution either spans none or all the vertices in that component. We remark that because of the second condition, this problem is *not* a restricted version of MDC. We formally define Labeled-MDC as follows.

Labeled-MDC	**Parameter:** $k+d$
Input: Graph G, a partition V_r, V_b of $V(G)$, and integers k, d	
Question: Does there exist a subset F of $E(G)$ of size at most k such that (a) every vertex in G/F has degree at most d; (b) $V(F) \subseteq V_r$; and (c) for a connected component C of $G[V_r]$, if $C \cap V(F) \neq \emptyset$ then $C \subseteq V(F)$.	

We say a set of edges F is a *solution* to instance $(G, (V_r, V_b), k, d)$ if the number of edges in F is at most k, the maximum degree of graph G/F is at most d, $V(F) \subseteq V_r$ and for a connected component C of $G[V_r]$, if $C \cap V(F) \neq \emptyset$ then $C \subseteq V(F)$.

It is easy to see that if $(G, (V_r, V_b), k, d)$ is a Yes instance of Labeled-MDC then (G, k, d) is a Yes instance of MDC. Let $\mathcal{U}$ be the family of all subsets of $V(G)$. If (G, k, d) is a Yes instance of MDC then $(G, (V_r, V(G) \setminus V_r), k, d)$ is a Yes instance of Labeled-MDC for some set V_r in $\mathcal{U}$. We use *universal sets* to construct a 'small' family of subsets of $V(G)$ that suffices for our purpose. We assume that there is a unique integer in $[n]$ for every vertex in $V(G)$. We use a subset of $[n]$ and a corresponding subset of $V(G)$ interchangeably.

▶ **Definition 6** (Universal Sets). *An (n, l)-universal set is a family $\mathcal{U}$ of subsets of $[n]$ such that for any $S \subseteq [n]$ of size l, the family $\{A \cap S \mid A \in \mathcal{U}\}$ contains all subsets of S.*

▶ **Proposition 7** ([5]). *For any $n, l \geq 1$ one can construct an (n, l)-universal set of size $2^{\mathcal{O}(l)} \cdot \log(n)$ in time $2^{\mathcal{O}(l)} \cdot n \log(n)$.*

In the following lemma, we argue that an FPT algorithm for Labeled-MDC leads to an FPT algorithm for MDC.

▶ **Lemma 8.** *Suppose there is an algorithm that given an instance $(G, (V_r, V_b), k, d)$ of Labeled-MDC runs in time $f(k, d) \cdot n^{\mathcal{O}(1)}$ and correctly determines whether it is a Yes instance. Then, there is an algorithm that given an instance (G, k, d) of MDC runs in time $2^{\mathcal{O}(dk)} \cdot f(k, d) \cdot n^{\mathcal{O}(1)}$ and correctly determines whether it is a Yes instance.*

Proof. Let $\mathcal{A}$ be an algorithm that given an instance $(G, (V_r, V_b), k, d)$ of Labeled-MDC runs in time $f(k, d) \cdot n^{\mathcal{O}(1)}$ and correctly determines whether it is a Yes instance. We first describe an algorithm for MDC that uses $\mathcal{A}$ as a subroutine. For the input (G, k, d), the algorithm constructs a $(U, 2k + kd)$-universal family $\mathcal{U}$ using Proposition 7. For every set V_r in $\mathcal{U}$, the algorithm runs Algorithm $\mathcal{A}$ with input $(G, (V_r, V(G) \setminus V_r), k, d)$. The algorithm returns Yes if Algorithm $\mathcal{A}$ returns Yes for one of these inputs otherwise it returns No. This completes the description of the algorithm. The running time of the algorithm follows from the description and Proposition 7. In the remaining proof, we argue the correctness of the algorithm. More precisely, we prove that (G, k, d) is a Yes instance of MDC if and only if there is a subset V_r in $\mathcal{U}$ such that $(G, (V_r, V(G) \setminus V_r), k, d)$ is a Yes instance of Labeled-MDC.

Suppose that (G, k, d) is a Yes instance of MDC and let F be a solution to it. Note that $|V(F)| \leq 2k$. We first argue that the number of vertices in $N(V(F))$ is at most kd. Let $\mathcal{W}$ be the G/F-witness structure of G and $\psi : V(G) \to V(G/F)$ be the corresponding function. Consider an arbitrary vertex v in $N(V(F))$. As v is not in $V(F)$, $\psi(v)$ corresponds to a small witness set in $\mathcal{W}$. As v is in $N(V(F))$, $\psi(v)$ is adjacent to a vertex in G/F that corresponds to a big witness set in $\mathcal{W}$. As $|F| \leq k$, there are at most k big witness sets in $\mathcal{W}$. Since the maximum degree of G/F is at most d, there are at most kd small witness sets in $\mathcal{W}$ that are adjacent with some big witness set. Hence, there are at most kd vertices in $N(V(F))$. As $\mathcal{U}$ is a $(n, 2k + dk)$-universal set and $|N[V(F)]| \leq 2k + dk$, there exists a set A in $\mathcal{U}$ such that the family $\{A \cap N[V(F)] \mid A \in \mathcal{U}\}$ contains all subsets of $N[V(F)]$. This implies, there exists a set, say V_r, such that $V_r \cap N[V(F)] = V(F)$. We argued that $(G, (V_r, V(G) \setminus V_r), k, d)$ is a Yes instance of Labeled-MDC.

Note that G/F has maximum degree at most d and $V(F) \subseteq V_r$. We need to prove that for a connected component C of $G[V_r]$ if $C \cap V(F) \neq \emptyset$ then $C \subseteq V(F)$. Assume that there exits a connected component C of $G[V_r]$ such that $C \cap V(F) \neq \emptyset$ and $C \setminus V(F) \neq \emptyset$. As C is a connected component and $C \cap V(F) \neq \emptyset$, there exists a vertex v in $C \setminus V(F)$ that is adjacent with some vertex in $V(F)$. Hence, there is a vertex in $N(V(F)) \cap V_r$. This contradicts the fact that $V_r \cap N[V(F)] = V(F)$. Hence, our assumption is wrong and $C \setminus V(F)$ is an empty set. This implies $(G, (V_r, V(G) \setminus V_r), k, d)$ is a YES instance of LABELED-MDC. As mentioned before, it is easy to see that if $(G, (V_r, V_b), k, d)$ is a YES instance of LABELED-MDC then (G, k, d) is a YES instance of MDC. This concludes the proof of the lemma. ◀

In the remaining section, we present a recursive algorithm for LABELED-MDC. We start with the following simple reduction rules.

▶ **Reduction Rule 9.** *For an instance $(G, (V_r, V_b), k, d)$, if the maximum degree of vertices in G is at most d and $k \geq 0$ then return a YES instance.*

It is easy to see that the first reduction rule is safe. Recall that a set of edges F is called solution to $(G, (V_r, V_b), k, d)$ if the number of edges in F is at most k, the maximum degree of graph G/F is at most d, $V(F) \subseteq V_r$, and for a connected component C of $G[V_r]$, if $C \cap V(F) \neq \emptyset$ then $C \subseteq V(F)$. Consider a connected component C of $G[V_r]$. If $|C| = 1$ then no solution edge can be incident to it. Also, if $|C| \geq 2k + 1$ then because of the last property and the fact that $|V(F)| \leq 2k$, no solution edge can be incident to vertices in C. These simple observations prove that the following reduction rule is safe.

▶ **Reduction Rule 10.** *For an instance $(G, (V_r, V_b), k, d)$, if there is a connected component, say C, of $G[V_r]$ such that $|C| = 1$ or $|C| \geq 2k + 1$ then move C from V_r to V_b i.e. return instance $(G, (V_r \setminus C, V_b \cup C), k, d)$.*

By Observation 5, vertex v in V_b can be adjacent to at most $d + k$ vertices in V_r. The following reduction rule ensures that the neighbors of v in V_r are not spread across many connected components.

▶ **Reduction Rule 11.** *For an instance $(G, (V_r, V_b), k, d)$, if there exists a vertex, say v, in V_b for which $N_G(v)$ intersects with $d+1$ different connected components of $G[V_r]$ then return a NO instance.*

▶ **Lemma 12.** *Reduction Rule 11 is safe.*

Proof. Assume that $(G, (V_r, V_b), k, d)$ is a YES instance. Let F be its solution and it contracts G to G/F via mapping ψ. Suppose $C_1, C_2, \ldots, C_{d+1}$ are connected components of $G[V_r]$ such that $C_i \cap N(v) \neq \emptyset$ for $i \in [d + 1]$. For every i, consider a vertex, say u_i, in $C_i \cap N(v)$. Let $U = \{u_1, u_2, \ldots, u_{d+1}\}$. Define $\psi(U) = \bigcup_{u \in U} \psi(u)$. For $i, j \in [d + 1]$, $i \neq j$ implies $\psi(u_i) \neq \psi(u_j)$ as C_i and C_j are two different connected components of $G[V_r]$ and $V(F) \subseteq V_r$. This implies $|\psi(U)| = |U| = d + 1$. As $V(F) \subseteq V_r$ and $v \in V_b$, F does not contain an edge incident on v. Hence, $\psi(v) \neq \psi(u_i)$ for any $i \in [d + 1]$. As $\psi(U) \subseteq N_{G/F}(\psi(v))$ and $|\psi(U)| \geq d + 1$, vertex $\psi(v)$ is adjacent with $d + 1$ or more vertices in G/F. This contradicts the fact that the maximum degree of vertices in G/F is at most d. Hence, our assumption was wrong and $(G, (V_r, V_b), k, d)$ is a NO instance. ◀

The algorithm exhaustively applies the reduction rules mentioned above. On a reduced instance, the algorithm creates multiple instances using the following subroutine. For an instance $(G, (V_r, V_b), k, d)$, a subset R of V_r, and a $(d + 1)$-coloring of R, the subroutine

creates a new instance by contracting each colored component of R into a single vertex, and (re-)label it blue. We need the notion of 'valid coloring' to filter out colorings that will not produce a 'smaller' instance. For graph H, a vertex coloring $\phi : V(H) \to [d+1]$ is said to be a *valid coloring* if every monochromatic connected component is of size at least two. We now describe the subroutine.

Subroutine `Colorwise-Contraction`**.** This subroutine takes as an input an instance $(G, (V_r, V_b), k, d)$ of Labeled-MDC, a non-empty subset R of V_r, and a valid coloring ϕ of $G[R]$. It returns another instance of Labeled-MDC. It initializes $G' = G$, $V'_r = V_r$, $V'_b = V_b$, and $k' = k$. For a monochromatic connected component C of $G[R]$, the subroutine finds a spanning tree of $G[C]$ and contracts all edges in it. Let v_C be the vertex obtained at the end of this series of edge contractions. It updates $V'_r = V_r \setminus C$, $V'_b = V_b \cup \{v_C\}$ and reduces k by $|C| - 1$. The subroutine repeats this procedure for every monochromatic connected component of $G[R]$. It returns $(G', (V'_r, V'_b), k', d)$ as instance of Labeled-MDC. This completes the description of the subroutine.

It is easy to verify that (V'_r, V'_b) is a partition of $V(G')$. As ϕ is a valid coloring of $G[R]$, a union of spanning trees of all monochromatic connected components of $G[R]$ contains at least $|R|/2$ edges. Hence, the subroutine contracts at least $|R|/2$ edges. This small observation will be helpful to get a bound on the running time of the algorithm.

▶ Remark 13. $k' \leq k - |R|/2$.

Let $\texttt{CC}[(G, (V_r, V_b), k, d); R; \phi]$ denote the instance returned by the subroutine when the input is $(G, (V_r, V_b), k, d)$, R, and ϕ. In the following lemma, we prove if the original instance is a Yes instance than at least one of the reduced instances is a Yes instance.

▶ **Lemma 14.** *Consider a* Yes *instance $(G, (V_r, V_b), k, d)$ of* Labeled-MDC*. Let R be a union of some connected components of $G[V_r]$. Suppose there is solution F to $(G, (V_r, V_b), k, d)$ such that $R \subseteq V(F)$. Then, there is a valid coloring $\phi : R \to [d+1]$ of $G[R]$ for which $\texttt{CC}[(G, (V_r, V_b), k, d); R; \phi]$ is a* Yes *instance.*

Proof. Let $H = G/F$. Consider the H-witness structure $\mathcal{W}$ of G and let G be contracted to H via ψ. Define a subset $\mathcal{W}_R$ of $\mathcal{W}$ as the collection of witness sets that intersects R. Formally, $\mathcal{W}_R = \{W \in \mathcal{W} \mid W \cap R \neq \emptyset\}$. Let $\mathcal{W}_R = \{W_1, W_2, \ldots, W_q\}$. For every $i \in [q]$, let h_i be the vertex corresponding to W_i. In other words, $W_i = \{v \in V(G) \mid \psi(v) = h_i\}$. Let $R_H = \{h_1, h_2, \ldots, h_q\}$.

Let F_1 be the collection of edges in F that are incident to some vertex in R. Hence, $R \subseteq V(F_1)$. As R is a union of connected components in $G[V_r]$ and $V(F_1) \subseteq V(F) \subseteq V_r$, we can conclude that $R = V(F_1) = \bigcup_{i \in [q]} W_i$. Hence, $\{W_1, W_2, \ldots, W_q\}$ is a partition of R. As there is a solution edge incident to every vertex in R, every witness set in $\mathcal{W}_R$ is a big witness set. This implies for every $i \in [q]$, there is a subset F_i of F such that $W_i = V(F_i)$. As the maximum degree of vertices in graph H is at most d, there is a proper $(d+1)$-coloring, say γ, of H. For $i, j \in [q]$, if (h_i, h_j) is an edge in H then $\gamma(h_i) \neq \gamma(h_j)$. Define a coloring $\phi : R \to [d+1]$ as follows. For $v \in R$, $\phi(v) = \gamma(h_i)$ where $v \in W_i$. As $\{W_1, W_2, \ldots, W_q\}$ is a partition of R, function ϕ is well defined. Since W_i is a big witness set, ϕ is a valid coloring.

By the construction of ϕ, any witness set in $\mathcal{W}$ is monochromatic. Since γ is a proper coloring of H, any two witness sets adjacent to each other have distinct colors. Hence, every witness set in $\mathcal{W}_R$ is a monochromatic connected component of coloring ϕ. As algorithm constructs every valid coloring of R, it also consider this coloring and create instance $(G', (V'_r, V'_b), k', d) = \texttt{CC}[(G, (V_r, V_b), k, d), R; \phi]$. For every $i \in [q]$, let F°_i be edges in a spanning tree of $G[W_i]$. Define $F^\circ = \bigcup_{i \in [q]} F^\circ_i$. As $W_i = V(F_i)$, graphs G/F_1 and G/F° are identical. Also, $|F^\circ_i| \leq |F_i|$ which implies $|F^\circ| \leq |F_1|$. Define $F^\star = (F \setminus F_1) \cup F^\circ$. It is easy to verify that $F^\star$ is also a solution to $(G, (V_r, V_b), k, d)$.

We now argue that $F^\star \setminus F^\circ$ is a solution to $(G', (V_r', V_b'), k', d)$. By the description of the algorithm, $k' = k - |F^\circ|$. As $|F^\star| \leq |F| \leq k$ and $F^\circ \subseteq F^\star$, we have $|F^\star \setminus F^\circ| \leq |F^\star| - |F^\circ| \leq k'$. Note that $G/F^\star = (G/F^\circ)/(F^\star \setminus F^\circ) = G'/(F^\star \setminus F^\circ)$ as $G' = G/F^\circ$. This implies the maximum degree of $G'/(F^\star \setminus F^\circ)$ is at most d. The only thing that remains to argue is that $V(F^\star \setminus F^\circ)$ is contained in V_r'. By construction, $F \setminus F_1 = F^\star \setminus F^\circ$. As F_1 is the set of edges in F that were incident to R, we can conclude that no edge in $F^\star \setminus F^\circ$ is incident to R. Recall that $V_r' = V_r \setminus R$. Hence, $V(F^\star \setminus F^\circ) \subseteq V_r'$. This implies that $F^\star \setminus F^\circ$ is a solution to $(G', (V_r', V_b), k', d)$ and concludes the proof of the lemma. ◀

In the above lemma, instead of considering any arbitrary subset V_r we only consider a subset that is the union of one or more connected components of $G[V_r]$. This suffices for our purpose as the algorithm calls the subroutine only on such subsets of V_r. Also, note that we do not need to know the solution F explicitly to apply the above lemma. It suffices to know that such a solution exists. We are now able to present an algorithm for Labeled-MDC

Algorithm for Labeled-MDC. The algorithm takes as input an instance $(G, (V_r, V_b), k, d)$ of Labeled-MDC and returns Yes or No. If $k < 0$ then the algorithm returns No. If $k = 0$ then it finds the maximum degree of G. If it is at most d then the algorithm returns Yes otherwise it returns No. The algorithm exhaustively applies Reduction Rules 9, 10, and 11. If the reduced instance is a trivial Yes (resp. No) instance then the algorithm returns Yes (resp. No). Otherwise, it creates multiple instances and makes recursive calls on these instances. The algorithm returns Yes if one of the recursive calls returns Yes, otherwise; it returns No.

We now describe the procedure used by the algorithm to create new instances. Let $(G, (V_r, V_b), k, d)$ be the instance on which reduction rules are not applicable. The algorithm finds a vertex, say v, in G such that $\deg_G(v) \geq d + 1$. It considers the following two cases.

1. (Vertex v is in V_r) Let R be the connected component of $G[V_r]$ that contains v. The algorithm constructs all valid colorings $\phi : R \to [d + 1]$ of $G[R]$. For each coloring, the algorithm calls subroutine `Colorwise-Contraction` with input $(G, (V_r, V_b), k, d)$, R, and ϕ. The algorithm calls itself with the instances returned by this subroutine as the input.
2. (Vertex v is in V_b) Let $C_1, C_2, \ldots, C_q$ be the connected components of $G[V_r]$ such that $N(v) \cap C_i \neq \emptyset$ for every $i \in [q]$. For a non-empty subset $I \subseteq [q]$, define $R_I := \bigcup_{i \in I} C_i$. For every non-empty subset $I \subseteq [q]$, the algorithm proceeds as follows. If $|R_I| \geq 2k + 1$, the algorithm discards this choice of I and moves to the next one. Otherwise, the algorithm constructs all valid coloring $\phi : R_I \to [d + 1]$ of $G[R_I]$. For each coloring, the algorithm calls subroutine `Colorwise-Contraction` with input $(G, (V_r, V_b), k, d)$, R_I, and ϕ. The algorithm calls itself with the instance returned by this subroutine as input.

This completes the description of the algorithm.

In the following lemma, we prove that the algorithm described above is correct and runs in the desired time.

▶ **Lemma 15.** *There is an algorithm that given an instance $(G, (V_r, V_b), k, d)$ of Labeled-MDC runs in time $2^{(d+2)k} \cdot (d + 1)^{2k} \cdot n^{\mathcal{O}(1)}$ and correctly determines whether it is a Yes instance.*

Proof. We argue that the algorithm described above solves Labeled-MDC in the desired time. We prove this lemma by the induction over the solution size k.

Consider the base case when the solution size is zero. Here, the algorithm finds a maximum degree of the graph and depending on its value returns Yes or No. It is easy to see that the lemma holds in this case. Assume that the lemma is true when the solution size is at most $k - 1$.

We first prove that given a YES instance the algorithm returns YES. Suppose $(G, (V_r, V_b), k, d)$ is a YES instance of LABELED-MDC and let F be its solution. Note that this implies that F is a solution to (G, k, d). If the algorithm returned YES because Reduction Rule 9 returned a YES instance then the lemma is vacuously true. By Lemma 12, Reduction Rule 11 is not applicable on the input. Consider the instance obtained by the exhaustive application Reduction Rules 9 and 10 on the input instance. For notational convenience, we denote this reduced instance by $(G, (V_r, V_b), k, d)$. As Reduction Rule 9 is not applicable, there is a vertex in G that has degree at least $d+1$. Let v be the vertex of degree at least $d+1$ found by the algorithm. By Observation 5, $V(F)$ intersects with $N[v]$.

Consider the case when v is in V_r and let R be the connected component of $G[V_r]$ that contains v. Since $V(F) \subseteq V_r$, we have $R \cap V(F) \neq \emptyset$. As F is a solution to $(G, (V_r, V_b), k, d)$, $R \cap V(F) \neq \emptyset$ implies $R \subseteq V(F)$. Instance $(G, (V_r, V_b), k, d)$, subset R of V_r, and solution F satisfies the premise of Lemma 14. Hence, there is a valid coloring $\phi : R \to [d+1]$ of $G[R]$ such that $\texttt{CC}[(G, (V_r, V_b), k, d), R; \phi]$ is a YES instance. As $R \neq \emptyset$, Remark 13 implies that $k' < k$. By the induction hypothesis, the algorithm correctly returns YES when the input is $(G', (V_r', V_b'), k', d)$. As one of the recursive calls returns YES, the algorithm returns YES when the input is $(G, (V_r, V_b), k, d)$ and v is in V_r.

Consider the case when v is in V_b. Let $C_1, C_2, \ldots, C_q$ be the connected components of $G[V_r]$ such that $N(v) \cap C_i \neq \emptyset$ for every $i \in [q]$. Recall that for a non-empty subset $I \subseteq [q]$, $R_I = \bigcup_{i \in I} C_i$. As $V(F)$ intersects $N[v]$ and $V(F) \subseteq V_r$, there exists a non-empty subset $I' \subseteq [q]$ such that for $i \in [q]$, $C_i \cap N(v) \neq \emptyset$ if and only if $i \in I'$. As F is a solution to $(G, (V_r, V_b), k, d)$, $C_i \cap V(F) \neq \emptyset$ implies $C_i \subseteq V(F)$. Hence, $R_{I'} \subseteq V(F)$. As $|V(F)| \leq 2k$, $|R_{I'}| \leq 2k$. For every non-empty subset $I \subseteq [q]$ for which $|R_I| \leq 2k$, the algorithm constructs all valid coloring $\phi : R_I \to [d+1]$ of $G[R_I]$ and calls `Colorwise-Contraction`. Instance $(G, (V_r, V_b), k, d)$, subset $R_{I'}$ of V_r, and solution F satisfies the premise of Lemma 14. Hence, there is a valid coloring $\phi : R_{I'} \to [d+1]$ of $G[R_{I'}]$ such that $(G', (V_r', V_b'), k', d) = \texttt{CC}[(G, (V_r, V_b), k, d), R_{I'}, \phi]$ is a YES instance. As $R \neq \emptyset$, Remark 13 implies that $k' < k$. By the induction hypothesis, the algorithm correctly returns YES when the input is $(G', (V_r', V_b'), k', d)$. As one of the recursive calls returns YES, the algorithm returns YES when the input is $(G, (V_r, V_b), k, d)$ and v is in V_b. This implies that if $(G, (V_r, V_b), k, d)$ is a YES instance then the algorithm returns YES.

We now prove that if the algorithm returns YES on instance $(G, (V_r, V_b), k, d)$ then it is a YES instance of LABELED-MDC. If the algorithm returned YES because Reduction Rule 9 returned a YES instance then the lemma is vacuously true. Otherwise, there is a newly created instance, say $(G', (V_r', V_b'), k', d)$, on which the recursive call of the algorithm returned YES. Let R be the subset of V_r and ϕ be its valid coloring such that `Colorwise-Contraction` returned this instance when input was $(G, (V_r, V_b), k, d)$, R, and ϕ. Let F° be the edges in G contracted by the subroutine to contract G'. In other words, F° is a collection of spanning trees of connected monochromatic components of $G[R]$. Note that $|F^\circ| = k - k'$. The algorithm calls `Colorwise-Contraction` only on non-empty subsets R. Hence, by Remark 13, $k' < k$. By the induction hypothesis, $(G', (V_r', V_b'), k', d)$ is a YES instance of LABELED-MDC. It is easy to see that if F' is a solution to $(G', (V_r', V_b'), k', d)$ then $F' \cup F^\circ$ is a solution to $(G, (V_r, V_b), k, d)$. This concludes the proof of the correctness of the algorithm.

We now bound the running time of the algorithm. The algorithm can apply all the reduction rules in polynomial time. It creates new instances only when none of the reduction rules are applicable. As Reduction Rules 10 is not applicable, any connected component of $G[V_r]$ has at least two and at most $2k$ vertices. In Case (1), the algorithm creates at most $(d+1)^{|R|}$ many instances. By Remark 13 and the induction hypothesis, the time taken by the algorithm in this case is

$$(d+1)^{|R|} \cdot 2^{(d+2)(k-|R|/2)} \cdot (d+1)^{2(k-|R|/2)} \cdot n^{\mathcal{O}(1)} \leq 2^{(d+2)k} \cdot (d+1)^{2k} \cdot n^{\mathcal{O}(1)}.$$

As Reduction Rule 11 is not applicable, for any vertex v in V_b, there are at most d connected components of $G[V_r]$ that intersects $N(v)$. In Case (2), the algorithm constructs all valid partitions of R_I only when $|R_I| \leq 2k$. Hence, in this case, the algorithm creates $2^d \cdot (d+1)^{|R|}$ many instances. By Remark 13 and the induction hypothesis, the time taken by the algorithm in this case is

$$2^d \cdot (d+1)^{|R|} \cdot 2^{(d+2)(k-|R|/2)} \cdot (d+1)^{2(k-|R|/2)} \cdot n^{\mathcal{O}(1)} \leq 2^{(d+2)k} \cdot (d+1)^{2k} \cdot n^{\mathcal{O}(1)}.$$

As $|R| \geq 2$, we have $2^d \cdot 2^{(d+2)(-|R|/2)} \leq 1$. This completes the proof of the lemma. ◀

The correctness of Theorem 2 immediately follows from Lemma 8 and Lemma 15.

5 No Polynomial Kernel

In this section, we present a sketch of a reduction from Red Blue Dominating Set (RBDS). In this problem, an input is comprised of a bipartite graph H with a bipartition (R, B) of $V(H)$, and a positive integer l. The question is, does there exist a subset R' of R of size at most l such that $N(R') = B$?

In this problem, an input comprises a bipartite graph H with a bipartition (R, B) of $V(H)$, and a positive integer l. The question is, does there exist a subset R' of R of size at most l such that $N(R') = B$? Without loss of generality, we can assume that $l + 3 < |B|$ and no vertex in R is adjacent to all but one vertices in B. We know the following result about the compression of the problem. See, for example, Theorem 15.18 in [14].

▶ **Proposition 16.** *Unless* NP ⊆ coNP/*poly, RBDS, parameterized by* $|B|$*, does not admit a polynomial compression.*

If $|R| > 2^{|B|}$ then there are at least two different vertices, say r_1, r_2 such that $N(r_1) = N(r_2)$. It is easy to see that it is safe to delete one of these two vertices. In this case, we can ensure, in polynomial time, that $|R| \leq 2^{|B|}$ by repeating the above process. This implies $\log_2 |R| \leq |B|$. Hence, we get the following corollary of Proposition 16.

▶ **Corollary 17.** *Unless* NP ⊆ coNP/*poly, RBDS, parameterized by* $|B| + \log_2 |R|$*, does not admit a polynomial compression.*

For the sake of clarity, we use both $|B|$ and $\log_2 |R|$ as parameters instead of replacing $\log_2 |R|$ by the larger parameter $|B|$. For notational convenience, we assume that $\log_2 |R|$ is an integer. If this is not the case, one can add some isolated vertices in R to ensure that $\log_2 |R|$ is an integer. This results in at most doubling of the number of vertices in it.

We first present an overview of the reduction. Consider an instance (H, R, B, l) of RBDS. See Figure 2 for an illustration. The reduction makes a copy of R and two copies of B, say B^1, B^2. For every vertex b in B, we denote its two copies in B^1, B^2 by b^1, b^2, respectively. For every edge (r, b), the reduction adds edges (r, b^1) and (r, b^2). It adds two independent sets U^1, U^2. For every vertex $u \in U^1 \cup U^2$, it adds some pendent vertices adjacent to it. The reduction adds all edges to make a complete bipartite graph with (B^1, U^1) as its bipartition. Similarly, it adds all edges to make a complete bipartite graph with (B^2, U^2) as its bipartition. For every vertex b in B, it adds a set of independent vertices X_b. For every x in X_b, it adds some pendent vertices adjacent to it and adds edges (b^1, x), (b^2, x). We briefly present an intuition behind the construction before presenting the last step. Let G be the graph constructed so far and k, d be two integers whose values depend only on $|B|, \log_2 |R|$. Suppose the reduction returns (G, k, d) as an instance of MDC.

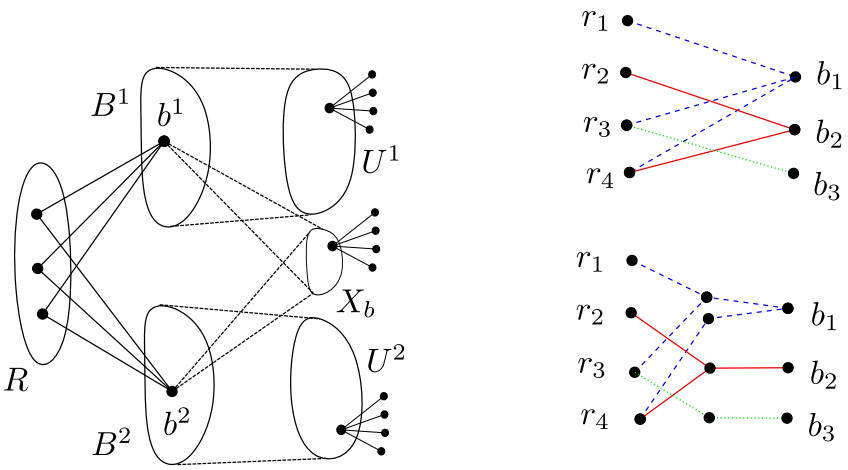


Figure 2 (Left) Overview of the reduction. The doted lines indicate that there is a complete bipartite graph cross two sets. (Right) The operation of replacing edges incident to vertex in B by a tree rooted at that vertex.

We set the value of d and the number of pendant vertices such that it is ensured that the only vertices in $U^1 \cup U^2 \cup X_b$ have degree more than d in G. We fix k and the sizes of sets U^1, U^2, X_b to ensure that any solution for the reduced instance of MDC satisfy the following properties.

1. It does not include an edge with one of its endpoints in $B^1 \cup B^2$ and another in $U^1 \cup U^2$.
2. For any b in B, it does not include an edge with one of its endpoints in $\{b^1, b^2\}$ and another in X_b.
3. It spans all vertices in $B^1 \cup B^2$. In other words, $B^1 \cup B^2 \subseteq V(F)$.
4. There are at most l witness sets in the G/F-witness structure of G that contain vertices in B^1 (similarly in B^2).
5. For every b in B, F includes b^1, b^2 in the same witness set.

Property (4) ensures that the degree constraints for the vertices in U^1 (similarly in U^2) are satisfied. Property (5) ensures that for every b in B, the degree constraints for the vertices in X_b are satisfied. Because of Property (1) and (2), only the vertices in R can make a witness set connected. Hence, each witness set should contain at least one vertex from R. We set the budget k such that each witness set contains exactly one vertex from R. To prove connectivity to witness set, this vertex needs to be adjacent to all vertices in that witness set. Hence, the set of endpoints of edges in a solution to (G, k, d) contains at most l vertices in R that dominates B. This naturally leads to a solution to $(H.R, B, l)$.

We now present the last step in the construction. The degree of the vertices formed by contracting a witness set can be larger than d. To avoid this, we replace star centered at b and whose leaves are in R by a binary tree rooted at that vertex. We ensure that for every edge incident b, there is a unique root-to-leaf path in the binary tree rooted at b and vice versa.

We present a formal reduction and its proof of correctness in the full version of the paper. This reduction combined with the fact that unless NP $\subseteq$ coNP/*poly*, RBDS, parameterized by $|B|$, does not admit a polynomial compression leads to a proof of Theorem 3.

6 Conclusion

In this article, we studied Maximum Degree Contraction problem. We prove that a simple brute force algorithm for this problem is optimal under ETH. This lower bound also implies that the known FPT algorithm with running time $(d+k)^k \cdot n^{\mathcal{O}(1)}$ is also optimal under the same hypothesis. We compliment this result by presenting another FPT algorithm with running time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$. While these two FPT algorithms are incomparable, our algorithm runs faster for smaller values of d, for which the problem still remains NP-Hard. We also prove that unless NP $\subseteq$ coNP/*poly*, the problem does not admit a polynomial compression when parameterized by $k + d$.

Most of the $\mathcal{H}$-Contraction problems do not admit a polynomial kernel under the same complexity conjecture. For some graph classes like trees, cactus, cliques, splits graphs, such negative results have been complimented by establishing a *lossy kernel* of polynomial size for these problems. There are also examples like Chordal Contraction, s-Club Contraction (for $s \geq 2$) for which we know that lossy kernel of polynomial size do not exist. We conclude this article with following open question: Does Maximum Degree Contraction admit a lossy kernel of polynomial size?

References

1 Akanksha Agarwal, Saket Saurabh, and Prafullkumar Tale. On the parameterized complexity of contraction to generalization of trees. *Theory of Computing Systems*, 63(3):587–614, 2019.

2 Akanksha Agrawal, Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Prafullkumar Tale. Path contraction faster than 2^n. *SIAM Journal on Discrete Mathematics*, 34(2):1302–1325, 2020.

3 Akanksha Agrawal, Lawqueen Kanesh, Saket Saurabh, and Prafullkumar Tale. Paths to trees and cacti. In *International Conference on Algorithms and Complexity*, pages 31–42. Springer, 2017.

4 Akanksha Agrawal, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Split contraction: The untold story. *ACM Transactions on Computation Theory (TOCT)*, 11(3):1–22, 2019.

5 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

6 Takao Asano and Tomio Hirata. Edge-contraction problems. *Journal of Computer and System Sciences*, 26(2):197–208, 1983.

7 Balabhaskar Balasundaram, Shyam Sundar Chandramouli, and Svyatoslav Trukhanov. Approximation algorithms for finding and partitioning unit-disk graphs into co-k-plexes. *Optimization Letters*, 4(3):311–320, 2010.

8 Rémy Belmonte, Petr A. Golovach, Pim Hof, and Daniël Paulusma. Parameterized complexity of three edge contraction problems with degree constraints. *Acta Informatica*, 51(7):473–497, 2014.

9 Nadja Betzler, Robert Bredereck, Rolf Niedermeier, and Johannes Uhlmann. On bounded-degree vertex deletion parameterized by treewidth. *Discrete Applied Mathematics*, 160(1-2):53–60, 2012.

10 Hans L Bodlaender and Babette van Antwerpen-de Fluiter. Reduction algorithms for graphs of small treewidth. *Information and Computation*, 167(2):86–119, 2001.

11 Andries Evert Brouwer and Henk Jan Veldman. Contractibility and NP-completeness. *Journal of Graph Theory*, 11(1):71–79, 1987.

12 Leizhen Cai and Chengwei Guo. Contracting few edges to remove forbidden induced subgraphs. In *International Symposium on Parameterized and Exact Computation*, pages 97–109. Springer, 2013.

13 Zhi-Zhong Chen, Michael Fellows, Bin Fu, Haitao Jiang, Yang Liu, Lusheng Wang, and Binhai Zhu. A linear kernel for co-path/cycle packing. In *International Conference on Algorithmic Applications in Management*, pages 90–102. Springer, 2010.

14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

15 Anders Dessmark, Klaus Jansen, and Andrzej Lingas. The maximum k-dependent and f-dependent set problem. In *International Symposium on Algorithms and Computation*, pages 88–97. Springer, 1993.

16 Michael R Fellows, Jiong Guo, Hannes Moser, and Rolf Niedermeier. A generalization of nemhauser and trotter's local optimization theorem. *Journal of Computer and System Sciences*, 77(6):1141–1158, 2011.

17 Fedor V. Fomin, Daniel Lokshtanov, Ivan Mihajlin, Saket Saurabh, and Meirav Zehavi. Computation of Hadwiger Number and Related Contraction Problems: Tight Lower Bounds. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, pages 49:1–49:18. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

18 Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. In *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

19 Petr A Golovach, Marcin Kaminski, Daniël Paulusma, and Dimitrios M Thilikos. Increasing the minimum degree of a graph by contractions. *Theoretical computer science.*, 481:74–84, 2013.

20 Petr A Golovach, Pim van't Hof, and Daniël Paulusma. Obtaining planarity by contracting few edges. *Theoretical Computer Science*, 476:38–46, 2013.

21 Sylvain Guillemot and Dániel Marx. A faster fpt algorithm for bipartite contraction. *Information Processing Letters*, 113(22-24):906–912, 2013.

22 Spoorthy Gunda, Pallavi Jain, Daniel Lokshtanov, Saket Saurabh, and Prafullkumar Tale. On the parameterized approximability of contraction to classes of chordal graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*, 2020 (To Appear).

23 Pinar Heggernes, Pim Van'T Hof, Daniel Lokshtanov, and Christophe Paul. Obtaining a bipartite graph by contracting few edges. *SIAM Journal on Discrete Mathematics*, 27(4):2143–2156, 2013.

24 Pinar Heggernes, Pim Van't Hof, Benjamin Lévêque, Daniel Lokshtanov, and Christophe Paul. Contracting graphs to paths and trees. *Algorithmica*, 68(1):109–132, 2014.

25 Christian Komusiewicz, Falk Hüffner, Hannes Moser, and Rolf Niedermeier. Isolation concepts for efficiently enumerating dense subgraphs. *Theoretical Computer Science*, 410(38-40):3640–3654, 2009.

26 R Krithika, Pranabendu Misra, and Prafullkumar Tale. An fpt algorithm for contraction to cactus. In *International Computing and Combinatorics Conference*, pages 341–352. Springer, 2018.

27 Ramaswamy Krithika, Pranabendu Misra, Ashutosh Rai, and Prafullkumar Tale. Lossy kernels for graph contraction problems. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

28 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. *SIAM Journal on Computing*, 47(3):675–702, 2018.

29 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. On the hardness of eliminating small induced subgraphs by contracting edges. In *International Symposium on Parameterized and Exact Computation*, pages 243–254. Springer, 2013.

30 Barnaby Martin and Daniël Paulusma. The computational complexity of disconnected cut and 2k2-partition. *Journal of combinatorial theory, series B*, 111:17–37, 2015.

31 Naomi Nishimura, Prabhakar Ragde, and Dimitrios M Thilikos. Fast fixed-parameter tractable algorithms for nontrivial generalizations of vertex cover. *Discrete Applied Mathematics*, 152(1-3):229–245, 2005.

32 Saket Saurabh, Uéverton dos Santos Souza, and Prafullkumar Tale. On the parameterized complexity of grid contraction. In *17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

33 Toshimasa Watanabe, Tadashi Ae, and Akira Nakamura. On the removal of forbidden graphs by edge-deletion or by edge-contraction. *Discrete Applied Mathematics*, 3(2):151–153, 1981.

34 Toshimasa Watanabe, Tadashi Ae, and Akira Nakamura. On the np-hardness of edge-deletion and-contraction problems. *Discrete Applied Mathematics*, 6(1):63–78, 1983.

PACE Solver Description: Fluid

Max Bannach
Institute for Theoretical Computer Science, Universität zu Lübeck, Germany
bannach@tcs.uni-luebeck.de

Sebastian Berndt
Institute for IT Security, Universität zu Lübeck, Germany
s.berndt@uni-luebeck.de

Martin Schuster
Institute for Epidemiology, Kiel University, Germany
martin.schuster@epi.uni-kiel.de

Marcel Wienöbst
Institute for Theoretical Computer Science, Universität zu Lübeck, Germany
wienoebst@tcs.uni-luebeck.de

Abstract

This document describes the heuristic for computing treedepth decompositions of undirected graphs used by our solve fluid. The heuristic runs four different strategies to find a solution and finally outputs the best solution obtained by any of them. Two strategies are score-based and iteratively remove the vertex with the best score. The other two strategies iteratively search for vertex separators and remove them. We also present implementation strategies and data structures that significantly improve the run time complexity and might be interesting on their own.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases treedepth, heuristics

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.27

Supplementary Material

Repository	`github.com/maxbannach/Fluid`
Release	pace-2020
doi	`10.5281/zenodo.3871709`

1 Introduction

A treedepth decomposition T of an connected, undirected graph $G = (V, E)$ is a rooted tree such that G is a subgraph of the closure of T. Such a decomposition can be obtained iteratively by taking a vertex $v \in V$ as root of T. Its children are then the decompositions of the connected components of $G[V \setminus \{v\}]$. Our heuristic iteratively removes vertices or sets of vertices to obtain a treedepth decomposition in this top-down fashion. Different strategies for choosing these vertices are used and the best solution over all these strategies is presented as output.

2 Score-Based Strategies

Our first two strategies are based on score function on the vertices, i. e., we iteratively choose a vertex with the best score, remove it from G, insert it in T, and update the scores of the other vertices. We use the following two score functions:

Degree-Score: $\text{score}_{\text{d}}(v) = |N(v)|$;
Fill-In-Score: $\text{score}_{\text{fi}}(v) = |\{\, \{x, y\} \mid x, y \in N(v) \wedge x \neq y \wedge \{x, y\} \notin E \,\}|$.

A naive way of implementing a score-based strategy is to recursively take the best vertex, remove it from the graph, and recompute the connected components. However, in this way, computing the connected components alone would require time $O(|V| \cdot |E|)$. For larger graphs, such a non-linear running time is not acceptable.

Instead of modifying the graph, we use the score functions to compute an *elimination ordering* $\pi = (v_1, v_2, \ldots, v_n)$ of G, that is, a permutation of the vertices such that v_i has the highest score in $G[V \setminus \{v_1, \ldots, v_{i-1}\}]$. Updating the scores is comparatively simple – for instance, computing an elimination ordering for score_{d} can be done in time $O(|E|)$.

Given an elimination ordering, we now face the new problem of obtaining a treedepth decomposition from it. We could, of course, compute the tree vertex-by-vertex by iterating over the vertices in the order of π. But then, we would have to recompute the connected components again – yielding a run time of $O(|V| \cdot |E|)$. To overcome this issue, we use the algorithm presented in Listing 1, which avoids the recursive recomputation of connected components by processing π in reversed order using a union-find data structure [1].

Listing 1 An efficient algorithm that computes a treedepth decomposition from a given elimination ordering in time $O(|E| \log^* |E|)$. The algorithm builds the tree in reversed order and maintains a union-find data structure in order to find the roots of subtrees efficiently.

```
1  INPUT:  graph G and elimination ordering π = (v_1, v_2, ..., v_n)
2  OUTPUT: elimination tree T
3  T ← ∅
4  uf ← ∅ // union-find structure with root pointer for each set
5  for v ← v_n, v_{n-1}, ..., v_1 do
6    Insert v as new singleton subtree in T.
7    Insert {v} as new set in uf. // set root pointer to v
8    for each w with (v, w) ∈ G and w ∈ T do
9      if v and w are not in the same subtree in T then
10       Let r be the root of the subtree in T containing w.
11       Insert an edge from r to v.
12       Join v and w in uf; update root pointer.
13 return T.
```

The advantage of implementing the score-based algorithm in two phases is that we can check many elimination orderings efficiently. We utilise this idea by repeatedly adding random perturbations to the score functions and running the algorithm multiple times. This leads to a large collection of treedepth decompositions, from which we output the best one.

3 Separator-Based Strategies

Instead of removing one vertex at a time, we may also remove a whole vertex separator at once and then recur into the new connected components immediately. Our two separator-based strategies iteratively search for such vertex separators, remove them from the graph and, then, proceed on the connected components separately.

3.1 Searching Separators Greedily

The first strategy finds the separator using a greedy algorithm. We maintain three sets A, B, and C such that no vertex in A is connected to any vertex in B. Initially, an arbitrary vertex v is chosen and A is initialized as $\{v\}$. The neighbors of A are inserted into C, all other vertices of V go to B. Now, we iteratively choose the vertex $v \in C$ that has the least number of edges to B, move v to A and put the neighbors of v that are still in B into C.

The algorithm will return a set C of minimal size that separates the graph into A and B within some balanced range, e. g., both at least $1/4$ of $|V|$. We observed the following extension to be helpful: When the subgraph induced by B contains a small connected component, we move the entire component with all its neighbors to A. This operation decreases the size of C while not changing the ratio of $|A|$ and $|B|$ too much.

Finally, we swap the role of A and B whenever $|B|$ is decreased to $1/4$ of $|V|$. In this way, we let B grow and A shrink and are often able to find smaller separators or separators with a better balance.

3.2 Search for Separators using Community Detection

Our second strategy runs the *asynchronous fluid communities algorithm* for community detection [3]. The algorithm computes a partition of V into two sets A and B. These sets are not necessarily of the same size, but are likely to be communities, i. e., it should be easy to separate A from B, but not so easy to separate the graph within A or within B.

In order to find a separator between A and B that we can use for our treedepth decomposition, we construct an auxiliary bipartite graph with one shore being A and the other being B. The edges of this bipartite graph are just the edges of the input graph G with one endpoint in A and one in B. We compute a maximum matching M in the bipartite graph and, using the König-Egervary Theorem [2], transform M into a minimum vertex cover S of the bipartite graph. This set S is then the sought separator in the original graph G.

We remark that, even though our algorithm has its name from the fluid community detection, it turned out that score-based heuristics or the greedy separator strategy is often superior compared to the fluid algorithm – both in quality and speed. However, there were a few instances in the test set on which this strategy was notably better than the others.

References

1 Bernard A. Galler and Michael J. Fischer. An improved equivalence algorithm. *Commun. ACM*, 7(5):301–303, 1964.

2 Denes König. Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. *Mathematische Annalen*, 77(4):453–465, 1916. doi:10.1007/BF01456961.

3 Ferran Parés, Dario Garcia Gasulla, Armand Vilalta, Jonatan Moreno, Eduard Ayguadé, Jesús Labarta, Ulises Cortés, and Toyotaro Suzumura. Fluid communities: A competitive, scalable and diverse community detection algorithm. In Chantal Cherifi, Hocine Cherifi, Márton Karsai, and Mirco Musolesi, editors, *Complex Networks & Their Applications VI*, pages 229–240, Cham, 2018. Springer International Publishing.

PACE Solver Description: PID*

Max Bannach
Institute for Theoretical Computer Science, Universität zu Lübeck, Germany
bannach@tcs.uni-luebeck.de

Sebastian Berndt
Institute for IT Security, Universität zu Lübeck, Germany
s.berndt@uni-luebeck.de

Martin Schuster
Institute for Epidemiology, Kiel University, Germany
martin.schuster@epi.uni-kiel.de

Marcel Wienöbst
Institute for Theoretical Computer Science, Universität zu Lübeck, Germany
wienoebst@tcs.uni-luebeck.de

Abstract

This document provides a short overview of our treedepth solver PID* in the version that we submitted to the exact track of the PACE challenge 2020. The solver relies on the positive-instance driven dynamic programming (PID) paradigm that was discovered in the light of earlier iterations of the PACE in the context of treewidth. It was recently shown that PID can be used to solve a general class of vertex pursuit-evasion games – which include the game theoretic characterization of treedepth. Our solver PID* is build on top of this characterization.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases treedepth, positive-instance driven

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.28

Supplementary Material

Repository	`github.com/maxbannach/PID-Star`
Release	pace-2020
doi	`10.5281/zenodo.3871800`

1 Introduction to Positive-Instance Driven Dynamic Programming

Many graph decompositions have game theoretic characterizations in the form of *vertex pursuit-evasion games*. Such games, which are also known as *graph searching* or *cops and robber,* are played by two players on an undirected graph $G = (V, E)$. In the version of the game that corresponds to treedepth, the first player places a team of k *searchers* on the vertices of the graph, while the second player controls a single *fugitive* that hides in a connected component of the graph. The game is played in rounds as follows [3]: Initially, the fugitive picks one connected component C of G. The game is continued only on $G[C]$ and we say that C is *contaminated.* In each round, both players perform one action:

1. The searchers pick a vertex $v \in C$ on which they want to place the next searcher. We say they *clean* the vertex v.
2. The fugitive responds by picking a component C' of $G[C \setminus \{v\}]$. The contaminated area is reduced to C' and the game proceeds only on this subgraph.

The game ends when the contaminated area shrinks to the empty set, or if the searchers have placed all k members of their team and C is still non-empty. In the first case the graph was *cleaned* and the fugitive was *caught,* in the second case the fugitive *escaped.* The searchers

win if they catch the fugitive, otherwise she wins. Note that in this version of the game, the searchers are not allowed to remove an already placed searcher from the graph. The game is therefore monotone and always ends after at most k rounds. Further observe that the fugitive is *visible* in the sense that the searchers know in which connected component she hides – in contrast, an invisible fugitive could hide in subgraphs that are not connected.

We call the configurations of this game *blocks,* which are tuple (C, ρ) with $\rho \in \mathbb{N}$ and $C \subseteq V$ being a connected subgraph with $|N(C)| + \rho \leq k$. Informally, C is the contaminated area (which is connected), and ρ is the number of remaining searchers. We require $|N(C)| + \rho \leq k$ as the neighborhood of C has to be cleaned in order to have C as contaminated area. Let us denote the set of all blocks of the game played on a graph G with a team of k searchers by $\mathcal{B}(G, k)$. Two blocks (C_1, ρ_1) and (C_2, ρ_2) *intersect* if $N[C_1] \cap C_2 \neq \emptyset$. The *start configuration* of the game is the block (V, k) and the *winning configurations* for the searchers are $(\emptyset, \rho \geq 0)$. We say the searchers have a *winning strategy* on a block (C, ρ) if they can ensure to reach a winning configuration no matter how the fugitive acts. The set of such blocks is the *winning region* of the searchers, which we denote by $\mathcal{R}(G, k) \subseteq \mathcal{B}(G, k)$. Every block in $\mathcal{R}(G, k)$ is called *positive*.

It is known that a graph has treedepth at most k if, and only if, k searchers have a winning strategy in the game defined above. In our notation we can express this fact as:

▶ **Fact 1** ([3]). *Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. Then $(V, k) \in \mathcal{R}(G, k) \iff \mathrm{td}(G) \leq k$.*

Fact 1 tells us that, in order to check whether the treedepth of a graph G is at most k, it is sufficient to compute the set $\mathcal{R}(G, k)$. One way of doing so would be to first compute $\mathcal{B}(G, k)$, then build an auxiliary graph on top of this set, and finally compute $\mathcal{R}(G, k)$ by solving reachability queries on this auxiliary graph. We can estimate the number of configurations with $|\mathcal{B}(G, k)| \leq (k + 1) \cdot n^{k+1}$, as there are n^k possible ways of placing k searchers on an n-vertex graph; at most n connected components adjacent to a separator; and since $\rho \in \{0, \ldots, k\}$. Therefore, the sketched algorithm achieves a run time of $O\big(n^{c \cdot k}\big)$ for a constant c, which is not feasible in practice for even moderate values of k.

In order to make the game theoretic approach feasible, we present an *output-sensitive* algorithm that computes just $\mathcal{R}(G, k)$ – without "touching" the rest of $\mathcal{B}(G, k)$. Such an algorithm is called *positive-instance driven.* This algorithmic technique was invented by Hisao Tamaki in the context of treewidth computations [5] and was recently shown to be able to solve a general class of graph searching games [1] – PID* is based on this version.

2 Description of the Core Algorithm

Before we describe the algorithm formally, let us build some intuition about how to compute the set $\mathcal{R}(G, k)$. Surely, we can not start at some block, say (V, k), and just simulate the game – we might touch a lot of blocks in $\mathcal{B}(G, k) \setminus \mathcal{R}(G, k)$ without even noticing it. After all, we do not know whether $(V, k) \in \mathcal{R}(G, k)$. We do know, however, that $(\emptyset, 0)$ is a winning configuration. So let us start with the set $\mathcal{R} = \{ (\emptyset, 0) \}$ and then try to grow it to $\mathcal{R}(G, k)$. We can first ask which configurations of the game lead to $(\emptyset, 0)$, i. e., what are configurations in which the searchers immediately win in the next round? These are the configurations $(\{v\}, 1)$ with $|N(v)| < k$, as in these the searchers can surround the fugitive and have a searcher left to place it on top of her in the next round. Now assume that we currently have a set $\mathcal{R} \subseteq \mathcal{R}(G, k)$ that did already grow a little. How does a configuration $(C, \rho) \in \mathcal{R}(G, k) \setminus \mathcal{R}$ that is "close to" $\mathcal{R}$ look like? The set C is connected by definition, and since the searchers have a winning strategy from (C, ρ), there is a vertex $v \in C$ such that $G[C \setminus \{v\}]$ has connected

components $C_1, \dots, C_q$ ($q = 1$ is possible) with $(C_i, \rho - 1) \in \mathcal{R}$ for all $i \in \{1, \dots, q\}$. To find these configurations, we scan through the blocks (C, ρ) in $\mathcal{R}$, guess a neighbor $v \in N(C)$ (the last cleaned vertex), and guess a set $X \subseteq \{ (C', \rho') \in \mathcal{R} \mid v \in N(C') \wedge N[C] \cap C' = \emptyset \wedge \rho' \leq \rho \}$ of pairwise non-intersecting blocks – the other configurations the fugitive could choose. Then the new block $(C \cup \bigcup_{(C',\rho') \in X} C' \cup \{v\}, \rho + 1)$ is positive and added to $\mathcal{R}$ if it has at most $k - \rho - 1$ neighbors. The complete algorithm is presented in Listing 1.

Listing 1 The core positive-instance driven algorithm tailored towards treedepth. We assume that the set $\mathcal{R}'$, the priority queue, and some data structure to mark already explored subgraphs C (for instance a hash set) are available in global memory.

```
1  INPUT:  graph G = (V,E) and number k ∈ ℕ
2  OUTPUT: a set R = R(G,k)
3
4  // in global memory
5  R' ← empty set of blocks
6  queue ←  priority queue of blocks (C,ρ) ordered by ρ
7
8  function pid()
9    // configurations leading to (∅,0)
10   for v in V do
11     if |N(v)| < k then
12       insert ({v},1) into queue
13     end
14   end
15   // compute the set R' ⊆ R(G,k)
16   while queue is not empty do
17     (C,ρ) ← extract a block from the queue
18     if C was already visited then
19       skip (C,ρ) and continue the while-loop
20     end
21     mark C as visited
22     // compute predecessor configurations
23     for v in N(C) do
24       for X ⊆ {(C',ρ') ∈ R' | v ∈ N(C') ∧ N[C] ∩ C' = ∅ ∧ ρ' ≤ ρ} do
25         // assert: blocks in X are pairwise non-intersecting
26         if |N(C ∪ ⋃_{(C',ρ')∈X} C' ∪ {v})| ≤ k − ρ − 1 then
27           insert (C ∪ ⋃_{(C',ρ')∈X} C' ∪ {v}, ρ + 1) into queue
28         end
29       end
30     end
31     R' ← R' ∪ {(C,ρ),}
32   end
33   // compute R from R'
34   R ← ⋃_{(C,ρ)∈R'} {(C,ρ') | ρ' ≥ ρ ∧ |N(C)| + ρ' ≤ k}
35 end
```

▶ **Theorem 2.** *Let $\mathcal{R}$ be the output of the algorithm in Listing 1 on input of a graph $G = (V, E)$ and a number $k \in \mathbb{N}$. Then $\mathcal{R} = \mathcal{R}(G, k)$.*

3 Preprocessing and Pruning Rules

To compute the treedepth of a graph $G = (V, E)$, we use the algorithm from the previous section for $k = 1, 2, \dots, \text{opt}$, i. e., we increase a lower bound until we reach the first positive instance. To each such instance (G, k), we apply the following reduction rules in advance:

▶ **Rule 1** (Leaf Rule [2]). *Let $v, w, w' \in V$ with $w, w' \in N(v)$ and $|N(w)| = |N(w')| = 1$, then delete w'.*

▶ **Rule 2** (Improvement Rule [4]). *Let $u, v \in V$ with $\{u, v\} \notin E$ and $|N(u) \cap N(v)| \geq k$, then add the edge $\{u, v\}$.*

▶ **Rule 3** (Simplical Rule [4]). *Let $u \in V$ be simplical such that $|N(v)| > k$ for all $v \in N(u)$, then delete u.*

To increase the performance of the algorithm from Listing 1, we apply the following pruning rules. We say a winning strategy of the searchers has a *conflict* if there are two vertices $u, v \in V$ with $N(u) \setminus \{v\} \subsetneq N(v) \setminus \{u\}$ such that the searchers clean u before v.

▶ **Lemma 3.** *If k searchers have a winning strategy on a graph $G = (V, E)$, then they also have a conflict free winning strategy on G.*

We can adapt the rules of our game with the lemma, without losing Fact 1. The new game simply forbids that the searchers clean a vertex u as long as there is a contaminated vertex v with $N(u) \setminus \{v\} \subsetneq N(v) \setminus \{u\}$. We define the following sets for every vertex $v \in V$:

$$\text{descendants}(v) = \{\, u \mid \{u, v\} \in E \wedge N[u] \subsetneq N[v] \,\},$$
$$\text{non-ancestors}(v) = \{\, u \mid \{u, v\} \notin E \wedge N(u) \subsetneq N(v) \,\}.$$

Assume the algorithm generates a new block (C, ρ) by gluing previously discovered blocks $(C_1, \rho_1), \ldots, (C_q, \rho_q)$ at some vertex $x \in V$, i. e., $C = \{x\} \cup \bigcup_{i=1}^{q} C_i$ (see line 27 in Listing 1). We check whether we have $\text{descendants}(x) \subseteq C$ and $x \notin \bigcup_{y \in C \setminus \{x\}} \text{non-ancestors}(y)$. If this is not the case, we discard the block.

Our second pruning rule avoids the expensive glue operation in line 24. Let (C, ρ) be a block and $v \in N(C)$. We say v is *covered* if $N(v) \subseteq N[C]$ and we call v an *attachment* if $\text{td}(G[C]) = \text{td}(G[C \cup \{v\}])$ and $|N(C)| = |N(C \cup \{v\})|$. One can show that we can, in both cases, greedily add v to C and proceed with $(C \cup \{v\}, \rho + 1)$ without further handling (C, ρ).

References

1 Max Bannach and Sebastian Berndt. Positive-instance driven dynamic programming for graph searching. In *Proceedings of the 16th International Symposium on Algorithms and Data Structures*, 2019. doi:10.1007/978-3-030-24766-9_4.

2 Robert Ganian, Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. SAT-Encodings for Treecut Width and Treedepth. In *Proceedings of the 21th Workshop on Algorithm Engineering and Experiments*, 2019. doi:10.1137/1.9781611975499.10.

3 Archontia C. Giannopoulou, Paul Hunter, and Dimitrios M. Thilikos. Lifo-search: A min-max theorem and a searching game for cycle-rank and tree-depth. *Discret. Appl. Math.*, 160(15):2089–2097, 2012. doi:10.1016/j.dam.2012.03.015.

4 Yasuaki Kobayashi and Hisao Tamaki. Treedepth Parameterized by Vertex Cover Number. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation*, 2016. doi:10.4230/LIPIcs.IPEC.2016.18.

5 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *J. Comb. Optim.*, 37(4):1283–1311, 2019.

PACE Solver Description: tdULL

Ruben Brokkelkamp
Centrum Wiskunde & Informatica (CWI), The Netherlands
ruben.brokkelkamp@cwi.nl

Raymond van Venetië
Korteweg–de Vries Institute, University of Amsterdam, The Netherlands
r.vanvenetie@uva.nl

Mees de Vries
University of Amsterdam, The Netherlands
meesdevries@protonmail.com

Jan Westerdiep
Korteweg–de Vries Institute, University of Amsterdam, The Netherlands
j.h.westerdiep@uva.nl

Abstract

We describe tdULL, an algorithm for computing treedepth decompositions of minimal depth. An implementation was submitted to the exact track of PACE 2020. tdULL is a branch and bound algorithm branching on inclusion-minimal separators.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Algorithm design techniques; Theory of computation → Graph algorithms analysis

Keywords and phrases PACE 2020, treedepth, treedepth decomposition, vertex ranking, minimal separators, branch and bound

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.29

Supplementary Material The source code can be found on `https://github.com/mjdv/tdULL` and `https://doi.org/10.5281/zenodo.3881472`

1 Introduction

The treedepth of an undirected graph is a measure of the complexity of the graph. Informally, it measures how resistant the graph is to being disconnected by removing vertices. All graphs have a treedepth between 1 and their number of vertices. Star graphs $K_{n,1}$, which can be completely disconnected by removing a single vertex, have treedepth 2. Trees have a treedepth at most logarithmic in their size. Complete graphs K_n have full treedepth of n. In general, computing the treedepth of a graph is NP-complete.

The PACE 2020 challenge consists of implementing an algorithm that is capable of computing treedepth. In the exact track, to which the solver tdULL was submitted, the goal was to compute the exact treedepth of 100 graphs from an unknown set, with a time limit of 30 minutes per graph. In order to test implementations, a set of 100 different but representative graphs was published at the start of the contest.

There are many equivalent definitions of treedepth. The following is useful for our purposes. For S a set of vertices of G, we write $G \setminus S$ for the graph obtained from G by removing the vertices from S and any incident edges. For a graph G, we write $\mathrm{cc}(G)$ for the set of its connected components.

▶ **Definition 1** (Treedepth). *The treedepth* $\mathrm{td}(G)$ *of a connected graph* $G = (V, E)$ *is recursively defined as*

$$\mathrm{td}(G) := \begin{cases} 1 & \text{if } |V| = 1, \\ \min\limits_{v \in V} \max\limits_{H \in \mathrm{cc}(G \setminus \{v\})} 1 + \mathrm{td}(H) & \text{else.} \end{cases}$$

The minimizing vertex v in the recursion can be taken as the root of a tree, with the connected components of $G \setminus \{v\}$ its children; this way, the computation of treedepth gives rise to a tree, the *treedepth decomposition* of G. The depth of this tree is the treedepth of G.

If removal of v does not cause the graph to be disconnected, the next (recursive) step is again the removal of a single vertex, until enough vertices have been removed to disconnect the graph. This leads to the following alternative definition. A set S of vertices of a connected graph G is called a *separator* of G if $G \setminus S$ is disconnected. Such a set S is called *inclusion minimal* if there is no $S' \subsetneq S$ which separates G. Write $\mathrm{sep}(G)$ for the inclusion-minimal separators of a graph G.

▶ **Definition 2** (Treedepth). *The treedepth* $\mathrm{td}(G)$ *of a connected graph* $G = (V, E)$ *is recursively defined as*

$$\mathrm{td}(G) := \begin{cases} |V| & \text{if } G \cong K_{|V|}, \\ \min\limits_{S \in \mathrm{sep}(G)} \max\limits_{H \in \mathrm{cc}(G \setminus S)} |S| + \mathrm{td}(H) & \text{else.} \end{cases}$$

With this definition, tdULL can be described in one sentence as a branch-and-bound algorithm based on Definition 2, with heuristics, clever data structures and exact special cases used for speed-up. The advantage of recurring on separators rather than on single vertices is avoiding duplicate branches: if we recur on a separator of size n, a vertex-based recursion may have $n!$ branches leading to that same point.

2 Algorithm

2.1 Branch and bound

To prune the search tree we use branch and bound. If we have found a treedepth decomposition of depth d for a particular graph, then any further search for a treedepth decomposition of that graph need not continue with any attempt that will have treedepth at least d. Similarly, if we have picked a separator S, and for $H \in \mathrm{cc}(G \setminus S)$ we have that $\mathrm{td}(H) = d$, then there is no need for us to find a treedepth decomposition for $H' \in \mathrm{cc}(G \setminus S)$ of depth lower than d, since it will not make the depth of the full decomposition any lower.

To this end, the main component of tdULL is a function `treedepth`, whose arguments are a graph G, as well as a *search lower bound* (SLB) and a *search upper bound* (SUB). The interval between these two bounds is the interval of treedepths dat are still relevant. The function `treedepth` returns a lower bound l and an upper bound u on the treedepth of G, for which at least one of the following three things is true:

- $u <$ SLB: the treedepth of G is so low we do not need the exact value;
- $l >$ SUB: the treedepth of G is so high we do not need the exact value;
- $u = l$: the treedepth of G is equal to $u = l$.

The main loop of `treedepth` generates the separators of the graph G, and recursively calls itself on the components left after removing the separator. The search bounds allow us to skip computation that is provably not going to improve the treedepth of the graph.

2.1.1 Lower bounds

Upper bounds on the treedepth of the graph G can be found directly, by completing branches of the recursion. Lower bounds are harder to find: the only sure way to find a lower bound is to recur on all possible separators, and conclude that a smaller treedepth cannot be realized.

To skip as many of the branches as possible, it is therefore important that we obtain good lower bounds as quickly as possible. Our main tool is the fact that the treedepth of a graph is *minor monotone*: if H is a graph that can be obtained from G by removing vertices and edges and contracting edges, then $\mathrm{td}(H) \leq \mathrm{td}(G)$. We apply this principle in a number of ways.

Any lower bound on the treedepth of a subgraph returned by the recursion is also a lower bound on $\mathrm{td}(G)$. In rare cases, this lower bound may actually be equal to the treedepth of G, allowing us to short-circuit the rest of the computation.

We pick specific subgraphs of G of which we compute the treedepth. The most important of these is a *core*: we take the vertex of lowest degree of G, and iteratively remove all vertices from G that have at most that degree. In the resulting graph every vertex has higher degree. Heuristically, this core should be the "toughest subset" of G, and thus provide good lower bounds.

If this core is empty, then we compute a contraction of the graph G instead, and try to find its treedepth to use as a lower bound. Specifically, we contract the edge between a vertex of minimal degree and a neighboring vertex for which the overlap of common neighbors is minimal. We experimented with other contraction strategies, but they proved less effective.

2.2 Separators

In order to use the recursion suggested by Definition 2, we need to be able to generate inclusion-minimal separators of the graph G. Analysis of the runtime of tdULL suggests that most of the runtime is spent on this generation process.

We essentially use the algorithm from [1]. This is an algorithm that produces minimal separators, which are subtly different from inclusion-minimal separators. A separator S is called minimal if there are vertices $v, w \in G \setminus S$ such that v, w are in different components after removing S, and there are no $S' \subsetneq S$ which separate v, w. It is easy to check whether a minimal separator S is also inclusion minimal: S is inclusion minimal if and only if each vertex in S has an edge to each connected component of $G \setminus S$. Thus we can use the algorithm for minimal separators, and filter out those which are not inclusion minimal.

We do not generate all separators of a graph at once, but in batches of 10,000. This occasionally helps us to avoid having to compute all the separators, by finding both an optimal decomposition and an optimal lower bound early. We tried several batch size strategies, both fixed and dynamic, and this one worked best.

A batch of separators is not tried in arbitrary order: we sort the separators by the size of the largest component that remains after removing the separator. This prioritizes separators that appear to be efficient at disconnecting the graph, which are heuristically more likely to lead to optimal solutions. We tried several sorting strategies, this one appeared to work best.

If G has a leaf (degree one vertex), then the one neighbor of that leaf forms an inclusion-minimal separator by itself. If a graph has a leaf attached to (nearly) every other vertex, our inclusion-minimal separators are (almost) the same as the vertices of degree greater than one. Then the recursion turns into the one from Definition 1, which is much less efficient. To avoid this problem, we actually compute separators on the graph with all leaves removed. Since leaves are never useful as the root of a treedepth decomposition, these separators suffice.

2.3 Special cases

There are a few cases in which we can quickly find an optimal treedepth decomposition of a graph G. We can easily recognize these cases and apply the faster direct algorithm.

- $G \cong K_n$ for some n;
- $G \cong C_n$ for some n;
- G is a tree (with the algorithm described in [2]).

2.4 Cache

Removing the same set of vertices in a different order results in the same graph. To avoid double work we store all lower and upper bounds on the treedepth of subgraphs in a cache. To quickly add, update and retrieve items from the cache we make use of a *SetTrie* data structure, as described in [3]. Furthermore, this data structure allows for efficient retrieval of subsets and hence can be used to compute lower bounds: because $\mathrm{td}(H) \leq \mathrm{td}(G)$ if $H \subseteq G$, we can use a lower bound on the treedepth of H as a lower bound on the treedepth of G.

This cache also works with contractions: every vertex created by a contraction has a canonical representation as the set of vertices which are contracted. When a new combination of vertices is used for a contraction, it gets a new global index to use in the SetTrie cache.

3 Discussion

The above algorithm is the result of a trial-and-error process, where a big subset of the public instances was used to compare versions. Comparing the number of cases solved by us and by the submissions above us on both the public and private test sets, it seems we may have suffered from some overfitting: we made algorithmic choices that performed better on the public instances of the problem than on the hidden instances.

Many ideas did not make it to the final algorithm. One notable omission is using the symmetry of a graph. Neither using graph automorphisms to reduce the number of separators nor trying to use a cache that works up to isomorphism yielded any improvement.

tdULL spends a lot of time on generating separators. If a graph has many – in the worst case, there may be exponentially many – this pushes us quickly over the time limit. Small improvements such as only using separators of the graph where all leaves are removed certainly helped, but we we have not found a way to substantially reduce the number of separators. Switching from a single vertex recursion to a separator-based recursion has been the biggest performance improvement, though.

Another improvement which yielded big performance improvements was finding better subgraphs with which to compute lower bounds. Even when a graph has a lot of separators, if for some separator the upper bound from the recursion matches the lower bound found before, the algorithm finishes fast because of an early exit.

References

1 Anne Berry, Jean-Paul Bordat, and Olivier Cogis. Generating all the minimal separators of a graph. In *Graph-Theoretic Concepts in Computer Science*, pages 167–172, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

2 Ananth V. Iyer, H. Donald Ratliff, and G. Vijayan. Optimal node ranking of trees. *Inf. Process. Lett.*, 28(5):225–229, August 1988. doi:10.1016/0020-0190(88)90194-9.

3 Iztok Savnik. Index data structure for fast subset and superset queries. In *International Conference on Availability, Reliability, and Security*, pages 134–148. Springer, 2013.

PACE Solver Description: SMS

Tuukka Korhonen
Department of Computer Science, University of Helsinki, Finland
`https://tuukkakorhonen.com`
tuukka.m.korhonen@helsinki.fi

Abstract

We describe SMS, our submission to the exact treedepth track of PACE 2020. SMS computes the treedepth of a graph by branching on the **S**mall **M**inimal **S**eparators of the graph.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Treedepth, PACE 2020, SMS, Minimal separators

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.30

Related Version A version of this solver description with an appendix containing additional details is available in `https://arxiv.org/abs/2006.07302`.

Supplementary Material The source code of SMS is available in [7] and in `https://github.com/Laakeri/pace2020-treedepth-exact`.

Funding This work has been financially supported by Academy of Finland (grant 322869).

1 Overview

SMS is an exact algorithm implementation for computing treedepth. SMS was developed for the 5th Parameterized Algorithms and Computational Experiments challenge (PACE 2020). The main algorithm implemented in SMS is a recursive procedure that branches on minimal separators [4]. Two variants of the branching algorithm are implemented, one with a heuristic algorithm for enumerating minimal separators and one with an exact algorithm [9]. Several lower bound techniques are implemented within the branching algorithm. Before applying the branching algorithm, preprocessing techniques are applied and a heuristic upper bound for treedepth is computed.

2 Notation

Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. The graph $G[X]$ is the induced subgraph of G with vertex set X. The set $N(v)$ is the neighborhood of a vertex v and $N(X)$ is the neighborhood of a vertex set X. The treedepth of G is denoted by $\mathtt{td}(G)$. A minimal a,b-separator of G is a subset-minimal vertex set S such that the vertices a and b are in different connected components of $G[V(G) \setminus S]$. The set of minimal separators of G for all pairs $a, b \in V(G)$ is denoted by $\Delta(G)$ and the set of minimal separators with size at most k by $\Delta_k(G)$. The set of vertex sets of connected components of G is denoted by $\mathcal{C}(G)$.

3 The Algorithm

3.1 Branching

SMS is based on the following characterization of treedepth.

▶ **Proposition 1** ([4]). *Let G be a graph. If G is complete then $\mathtt{td}(G) = |V(G)|$. Otherwise*

$$\mathtt{td}(G) = \min_{S \in \Delta(G)} \left(|S| + \max_{C \in \mathcal{C}(G[V(G) \setminus S])} \mathtt{td}(G[C]) \right).$$

Proposition 1 is implemented as a recursive algorithm that takes a vertex set X as input and computes $\mathtt{td}(G[X])$ by first enumerating the minimal separators of $G[X]$ and then branching from each minimal separator S to smaller induced subgraphs $G[C]$ for each component $C \in \mathcal{C}(G[X \setminus S])$. We make use of upper bounds by implementing Proposition 1 as a decision procedure which, given a vertex set X and a number k, decides if $\mathtt{td}(G[X]) \leq k$. Clearly, in this case we may consider only the minimal separators in $\Delta_{k-1}(G[X])$. Moreover, we handle the minimal separators with sizes $k-1$ and $k-2$ as special cases and thus consider only the minimal separators in $\Delta_{k-3}(G[X])$ in the main recursion. A minimal separator S with $|S| = k-1$ such that $\mathtt{td}(G[X \setminus S]) = 1$ must be a vertex cover of $G[X]$ and therefore is a neighborhood of a vertex. A minimal separator S with $|S| = k-2$ such that $\mathtt{td}(G[X \setminus S]) \leq 2$ has also a somewhat special structure, and we handle them with a modification of Berry's algorithm [1] for enumerating minimal separators.

3.2 Enumerating Small Minimal Separators

SMS spends most of its runtime in a subroutine which given a number k and a graph G enumerates $\Delta_k(G)$. To make use of the fact that heuristic enumeration of small minimal separators is more efficient than exact enumeration, two variants of the main branching algorithm are ran: first a variant using a heuristic minimal separator enumeration algorithm and then a variant using an exact minimal separator enumeration algorithm.

The heuristic enumeration algorithm is a simple modification of Berry's algorithm [1]. The modification prunes all minimal separators with more than k vertices immediately during the execution, outputting a set $\Delta'_k \subseteq \Delta_k(G)$ in $O(|\Delta'_k|n^3)$ time. As observed in [9], there are cases in which $\Delta'_k \neq \Delta_k(G)$. However, in practice the algorithm seems to often find all small minimal separators on the values of k that are relevant.

As an exact small minimal separator enumeration algorithm we implement the algorithm of Tamaki [9], including also the optimizations discussed in the paper. To the best of our knowledge there are no better bounds than $n^{k+O(1)}$ for the runtime of this algorithm. In practice it appears to usually have only a factor of 2-10 runtime overhead compared to the heuristic algorithm.

In cases when $G[C]$ is a child of G in the recursion, obtained by branching on a minimal separator $N(C) \in \Delta(G)$, and $|C| > |V(G)|/2$ we make use of the small minimal separators of G to enumerate the small minimal separators of $G[C]$. In particular, for all minimal separators $S \in \Delta_k(G[C])$, there exists a minimal separator $S' \in \Delta_{k+|N(C)|}(G)$ such that $S = C \cap S'$. Note that in this case $|N(C)|$ is exactly the difference in the values of k in recursive calls on $G[C]$ and G, and therefore $\Delta_{k+|N(C)|}(G)$ is already enumerated.

3.3 Lower Bounds

To avoid unnecessary re-computation, the known upper and lower bounds for $\mathtt{td}(G[X])$ are stored for each handled induced subgraph $G[X]$. To this end, an open addressing hashtable with linear probing is implemented. Also, we implement an ad-hoc data structure so that given a vertex set X, a vertex set $X' \subset X$ with the highest known lower bound for $\mathtt{td}(G[X'])$ can be found. This data structure uses the idea of computing subset-preserving hashes by using the intersection $X \cap V'$, where V' is a subset of vertices with size $O(\log n)$, where n is

the number of elements in the data structure. Other implemented algorithms for computing lower bounds on $\texttt{td}(G[X])$ are the MMD+ algorithm [3] which finds large clique minors, a depth-first search algorithm which finds long paths and cycles, and a graph isomorphism hashtable which finds already processed induced subgraphs $G[X']$ that are isomorphic to $G[X]$ and applies the lower bounds of $G[X']$ to $G[X]$.

3.4 Preprocessing Techniques

The preprocessing techniques implemented in SMS are *tree elimination* and the kernelization procedures described in [6]. Tree elimination finds a subgraph $G[T]$ such that $G[T]$ is a tree and $|N(V(G) \setminus T)| = 1$, i.e., the subgraph is attached to the rest of the graph only on a single vertex. Then it uses an exact algorithm to compute a list of length $\texttt{td}(G[T])$ that characterizes the behavior of $G[T]$ with respect to treedepth of G [8], and replaces $G[T]$ with a construction of $O(\texttt{td}(G[T])^2)$ vertices whose behavior is the same. The simplicial vertex kernelization rule from [6] is implemented as it is described there, but the shared neighborhood rule is generalized. In particular, if there are two non-adjacent vertices $u, v \in V(G)$, and the minimum u, v-vertex cut is at least k, where k is an upper bound for treedepth, then an edge can be added between u and v.

3.5 Upper Bounds

To compute upper bounds on treedepth we implement a novel heuristic algorithm. The algorithm first finds a triangulation (chordal completion) H of G using the LB-Triang algorithm [2] with a heuristic aiming to minimize the number of fill-edges in each step. Then it uses the branching algorithm, with some additional heuristics making it non-exact, to compute a treedepth decomposition of H. Any treedepth decomposition of H is also a treedepth decomposition of G. The properties of chordal graphs interplay nicely with the branching algorithm: chordal graphs have a linear number of minimal separators and the treewidth of a chordal graph can be computed in linear time [5]. Moreover, there exists a triangulation H of G with $\texttt{td}(H) = \texttt{td}(G)$, because treedepth can be formulated as a completion problem to a graph class that is a subset of chordal graphs [4].

References

1 Anne Berry, Jean-Paul Bordat, and Olivier Cogis. Generating all the minimal separators of a graph. *International Journal of Foundations of Computer Science*, 11(3):397–403, 2000. doi:10.1142/S0129054100000211.

2 Anne Berry, Jean-Paul Bordat, Pinar Heggernes, Geneviève Simonet, and Yngve Villanger. A wide-range algorithm for minimal triangulation from an arbitrary ordering. *Journal of Algorithms*, 58(1):33–66, 2006. doi:10.1016/j.jalgor.2004.07.001.

3 Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations II. Lower bounds. *Information and Computation*, 209(7):1103–1119, 2011. doi:10.1016/j.ic.2011.04.003.

4 Jitender S. Deogun, Ton Kloks, Dieter Kratsch, and Haiko Müller. On the vertex ranking problem for trapezoid, circular-arc and other graphs. *Discrete Applied Mathematics*, 98(1-2):39–63, 1999. doi:10.1016/S0166-218X(99)00179-1.

5 Philippe Galinier, Michel Habib, and Christophe Paul. Chordal graphs and their clique graphs. In Manfred Nagl, editor, *Proceedings of the 21st International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 1017 of *Lecture Notes in Computer Science*, pages 358–371. Springer, 1995. doi:10.1007/3-540-60618-1_88.

6 Yasuaki Kobayashi and Hisao Tamaki. Treedepth parameterized by vertex cover number. In Jiong Guo and Danny Hermelin, editors, *Proceedings of the 11th International Symposium on Parameterized and Exact Computation*, volume 63 of *LIPIcs*, pages 18:1–18:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.18.

7 Tuukka Korhonen. PACE 2020 exact treedepth submission: SMS, 2020. doi:10.5281/zenodo.3872898.

8 Alejandro A. Schäffer. Optimal node ranking of trees in linear time. *Information Processing Letters*, 33(2):91–96, 1989. doi:10.1016/0020-0190(89)90161-0.

9 Hisao Tamaki. Computing treewidth via exact and heuristic lists of minimal separators. In Ilias Kotsireas, Panos M. Pardalos, Konstantinos E. Parsopoulos, Dimitris Souravlias, and Arsenis Tsokas, editors, *Proceedings of the Special Event on Analysis of Experimental Algorithms*, volume 11544 of *Lecture Notes in Computer Science*, pages 219–236. Springer, 2019. doi:10.1007/978-3-030-34029-2_15.

PACE Solver Description: Computing Exact Treedepth via Minimal Separators

Zijian Xu
The University of Tokyo, Japan
xuzijian@ms.k.u-tokyo.ac.jp

Dejun Mao
The University of Tokyo, Japan
maodejun001@is.s.u-tokyo.ac.jp

Vorapong Suppakitpaisarn
The University of Tokyo, Japan
vorapong@is.s.u-tokyo.ac.jp

Abstract

This is a description of team xuzijian629's treedepth solver submitted to PACE 2020. As we use a top-down approach, we enumerate all possible minimal separators at each step. The enumeration is sped up by several novel pruning techniques and is based on our conjecture that we can always have an optimal decomposition without using separators with size larger than treewidth. Although we cannot theoretically guarantee that our algorithm based on the unproved conjecture can always give an optimal solution, it can give optimal solutions for all instances in our experiments. The algorithm solved 68 private instances and placed 5th in the competition.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Treedepth, Minimal Separators, Experimental Algorithm

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.31

Supplementary Material The solver submitted to the competition is available at `https://doi.org/10.5281/zenodo.3870624` and the repository is `https://github.com/xuzijian629/pace2020`.

1 Preliminaries

1.1 Notations

In this paper, G denotes undirected unweighted graph. V or $V(G)$ denote the vertex set. We use n and m for the number of nodes and edges, respectively. The treewidth and treedepth of G are expressed as $tw(G)$ and $td(G)$, respectively. $N(v)$ or $N_G(v)$ denote the open neighbors. For a vertex set $S \subseteq V$, $G[S]$ is the subgraph induced by S. We use $G \backslash S$ for the graph obtained from G by removing S, that is, $G \backslash S = G[V \backslash S]$. Also, we use $\mathcal{C}(G)$ to denote the connected components of G. S is called an a-b separator if $a, b \in V$ are not connected in $G \backslash S$. An a-b separator is called minimal if none of its proper subset is an a-b separator. Finally, S is called a minimal separator if S is a minimal a-b separator for some $a, b \in V$.

1.2 Computing Treedepth via Minimal Separators

The recursive formula we use for computing treedepth is a variant of the following theorem.

▶ **Theorem 1** ([4]).

$$td(G) = \begin{cases} |V| & \text{if } G \text{ is complete} \\ \min\limits_{S \in \Delta_G} \left(|S| + \max\limits_{H \in \mathcal{C}(G \backslash S)} td(H) \right) & \text{otherwise} \end{cases} \tag{1}$$

where Δ_G is a collection of all minimal separators of G.

Algorithm 1 MinDegree and MinFill.

Require: graph G
Ensure: upper bound of treewidth ub
1: $H := G$
2: **while** G is not empty **do**
3: $v :=$ a vertex with minimum degree (MinDegree)/with minimum fill (MinFill)
4: $F := \{(a, b) \mid a, b \in N_G(v) \text{ and } (a, b) \notin E(G)\}$
5: add edges in F to both G and H
6: remove v from G
7: **end while**
8: assert H is a chordal completion of G
9: $ub :=$ treewidth of H

The bottleneck of computing treedepth by Theorem 1 is the computation of Δ_G, since a graph may have an exponential number of minimal separators with respect to n.

To calculate an optimal treedepth decomposition from Equation (1), the authors begin by finding the set Δ_G. Then, for each minimal separator $S \in \Delta_G$ and for each connected component $H \in \mathcal{C}(G \backslash S)$, they recursively calculate $td(H)$. By that, they can obtain a set $S^* \in \arg\min_{S \in \Delta_G} \left(|S| + \max_{H \in \mathcal{C}(G \backslash S)} td(H) \right)$. An optimal treedepth decomposition obtained from the algorithm is a tree which:

1. the top of the tree is a simple path consisting of all nodes in S^*;
2. the bottom end of the simple path have several branches, each of the branches is connected to the root of an optimal treedepth decomposition for $H \in \mathcal{C}(G \backslash S)$, which can be computed recursively by the same algorithm.

2 Conjecture

In order to reduce the amount of the minimal separators that we have to enumerate, we conjectured that it suffices to enumerate minimal separators that have size at most treewidth. Formally,

▶ **Conjecture 2.** *Let Δ_G^p be a set of separators no larger than p, i.e. $\Delta_G^p := \{S \in \Delta : |S| \leq p\}$. Define $td^{(p)}(G)$ as follows:*

$$td^{(p)}(G) = \begin{cases} |V| & \text{if } G \text{ is complete} \\ \min_{S \in \Delta_G^p} \left(|S| + \max_{H \in \mathcal{C}(G \backslash S)} td(H) \right) & \text{otherwise} \end{cases} \tag{2}$$

Then, for any graph G and for any $p \geq tw(G)$, $td^{(p)}(G) = td(G)$.

It is known that we can efficiently calculate an upper bound of treewidth, denoted by $\overline{tw(G)}$ by taking the minimum of MinDegree heuristic and MinFill heuristic (Algorithm 1).

Then, if the conjecture is correct, we can replace the set S^* in the previous section with $\hat{S} \in \arg\min_{S \in \Delta_G^p} \left(|S| + \max_{H \in \mathcal{C}(G \backslash S)} td(H) \right)$. By that, our search space size is reduced from $|\Delta_G|$ to $\left|\Delta_G^{\overline{tw(G)}}\right|$, and we can significantly speed up the algorithm.

Correctness of the Conjecture and Solver

We cannot prove the correctness of Conjecture 2 for general graphs. Hence, we cannot theoretically guarantee that our solver based on the conjecture can always give an optimal solution. However, for all public and private instances we could solve, the solutions were optimal. Some theoretical aspects of this conjecture are discussed in [8].

3 Pruning Rules

In addition to the conjecture in the previous section, we speed up our solvers using several pruning rules.

Our solver handles the decision version of the treedepth problem. It computes $solve(G, k)$, checking if $td(G) \leq k$, for $k = 1, 2, \ldots$. When G is separated by a minimal separator S, $solve(G, k)$ recursively checks $solve(H, k - |S|)$ for each connected component $H \in \mathcal{C}(G \backslash S)$. If we have a method to efficiently calculate a lower bound of $td(H)$ for each H, we can immediately return `false` if the lower bound is larger than $k - |S|$. We can significantly prune the search space if the lower bound is tight. Therefore, we use several lower bounds of $td(H)$ in our solvers.

We compute the following six lower bounds from the first one. The first two lower bounds are usually not as effective as others, but the computation is much simpler.

Simple Bound. Consider a treedepth decomposition T such that it has k leaves and the depth is d. If T is a treedepth decomposition of G, then G may have at most $(k-1)(n-k/2)$ edges. The maximum case is obtained when all k leaves are at depth d and for $1 \leq i \leq d-1$, there is only one node at depth i. Therefore, we have the following lower bound for treedepth:

▶ **Proposition 3.** *Let k' be the smallest integer k such that $m \leq (k-1)(n-k/2)$, then $td(G) \geq k'$.*

Maximum Degree Bounds [7].

▶ **Proposition 4.** *Let $b > 0$ be the maximum degree of G. Then, $td(G) \geq lb(n)$, where*

$$lb(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 + lb(\lceil (n-1)/b \rceil) & \text{otherwise.} \end{cases}$$

Degeneracy Bound. Let G be a graph. The degeneracy of G is defined as the maximum of minimum degrees among all subgraphs. Degeneracy is a lower bound of treewidth [2] and can be computed in linear time [6] and often works as the most effective bound especially for small graphs. Since $td(G) \geq tw(G) + 1$, degeneracy plus one is a lower bound of treedepth.

Path-length Bound [7]. Let P be a path in G. Then, $td(G) \geq \lceil \log_2(|P| + 1) \rceil$, where $|P|$ is the number of nodes in the path P.

Contraction Degeneracy Bound [3, 5]. Contraction degeneracy is a stronger lower bound of treewidth, compared to degeneracy. However, since its computation takes time, we use this bound only when degeneracy is slightly smaller than k in $solve(G, k)$. The algorithm is described in Algorithm 2.

Pruning by Blocks. We observed that the above lower bounds can effectively prune our search only when n is as small as 100. For larger n, we introduce a novel pruning heuristic. As a preprocessing, we take various induced subgraphs of the input graph. These subgraphs are called blocks. Let `Blocks`$[i]$ be the collection of blocks with size i. For each i in ascending order, we compute the exact treedepth of the induced subgraphs and sort

Algorithm 2 Contraction Degeneracy Heuristic [3, 5].

Require: graph G
Ensure: lower bound of treewidth lb
1: $lb := 0$
2: **while** G has more than one vertices **do**
3: $v :=$ a vertex with minimum degree
4: $lb \leftarrow \max(lb, degree(v))$
5: $u :=$ a vertex in $N(v)$ such that $|N(u) \cap N(v)|$ is minimum
6: contract (u, v)
7: **end while**

them in descending order of treedepth. In $solve(H, k - |S|)$, we scan each blocks from smaller i and then from larger treedepth, and, if there is a block $B \subseteq H$ and $V(B) \cap S = \emptyset$ such that $td(B) > k - |S|$, we can immediately return `false`.

Block Selection in Preprocessing

To obtain various blocks, we need a quick randomized heuristic. We combine the following two partitioning heuristics to recursively decompose the input graph.

Partitioning Heuristic 1. Select two random vertices u and v in G and divide $V(G)$ into two groups, denoted U and V. A node v' is put in U if the shortest path length from v' to u is smaller than the shortest path length from v' to v. Otherwise v' is put in V. We then consider $G[U]$ and $G[V]$ as two blocks in the preprocessing process.

Partitioning Heuristic 2. A small vertex separator S is computed by [1]. We then consider each of the components of $G \backslash S$ as blocks.

To compute exact treedepth for the elements in `Blocks`$[i]$, `Blocks`$[j]$ for $j < i$ is used for pruning. The preprocessing is terminated either if it finishes the computation for all blocks or it exceeded the time limit for preprocessing. This pruning was so effective that it allows us to solve almost 20 public instances which we could not solve without it.

References

1 Haeder Y Althoby, Mohamed Didi Biha, and André Sesboüé. Exact and heuristic methods for the vertex separator problem. *Computers & Industrial Engineering*, 139:106135, 2020.
2 Hans L Bodlaender and Arie MCA Koster. Treewidth computations ii. lower bounds. *Information and Computation*, 209(7):1103–1119, 2011.
3 Hans L Bodlaender, Arie MCA Koster, and Thomas Wolle. Contraction and treewidth lower bounds. In *European Symposium on Algorithms*, pages 628–639. Springer, 2004.
4 Dariusz Dereniowski and Adam Nadolski. Vertex rankings of chordal graphs and weighted trees. *Information Processing Letters*, 98(3):96–100, 2006.
5 Vibhav Gogate and Rina Dechter. A complete anytime algorithm for treewidth. *arXiv preprint*, 2012. `arXiv:1207.4109`.
6 David W Matula and Leland L Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427, 1983.
7 James Trimble. An algorithm for the exact treedepth problem. *arXiv preprint*, 2020. `arXiv:2004.08959`.
8 Zijian Xu and Vorapong Suppakitpaisarn. On the size of minimal separators for treedepth decomposition, 2020. `arXiv:2008.09822`.

PACE Solver Description: Tree Depth with FlowCutter

Ben Strasser
Independent Researcher, Germany
acedemia@ben-strasser.net

Abstract

We describe the FlowCutter submission to the PACE 2020 heuristic tree-depth challenge. The task of the challenge consists of computing an elimination tree of small height for a given graph. At its core our submission uses a nested dissection approach, with FlowCutter as graph bisection algorithm.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases tree depth, graph algorithm, partitioning

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.32

Supplementary Material Our source code is available at `https://github.com/ben-strasser/flow-cutter-pace20` and `https://doi.org/10.5281/zenodo.3870928`.

Acknowledgements We thank the organizers for their work in making the 2020 PACE challenge a success.

1 Introduction

Using the original FlowCutter code base, we implemented a program to compute elimination trees of small height. We submitted it to the PACE 2020 heuristic tree-depth implementation challenge. In this paper, we describe our submission.

The objective of the challenge is to implement an algorithm that, given an undirected graph, computes an elimination tree of small height within a fixed amount of time. The height of the computed tree is used to score the competing implementations. Our source code is available at [13, 14]. The core of our program is very similar to the FlowCutter submissions to the PACE 2016 and 2017 heuristic tree decomposition [8, 12].

The FlowCutter algorithm was introduced in [6, 7]. Our submission is based on the original code. Two other independent PACE 2020 competitors make use of reimplementations of the FlowCutter algorithm. These are ExTREEm [16] and Sallow [17]. ExTREEm won the first place and Sallow the third. We won the second place. The FlowCutter algorithm is thus used in all top three submissions.

FlowCutter is a balanced graph bisection algorithm. The base version computes balanced edge cuts. There exists an extension that computes node separators. For a detailed description of FlowCutter, we refer to [6, 7]. The idea of FlowCutter is to compute a maximum flow and then to derive a minimum cut from it. If the derived cut is balanced, we are finished. Otherwise, additional source or target nodes are added and the flow intensity is increased. The minimum cut derived from the new flow has a better balance due to the additional source and target nodes. This is repeated until the desired balance is reached.

FlowCutter does not only compute a single cut. It computes a sequence of cuts that optimize balance and cut size in the Pareto-sense. Using this sequence, solutions to more complex problems can be found. For example, the cut expansion can be maximized. The expansion of a cut is its size divided by the number nodes on the smaller side.

FlowCutter has been developed to analyze road networks with the goal of accelerating shortest path queries. FlowCutter is used to compute an elimination tree, which is used as input to an algorithm called Customizable Contraction Hierarchy (CCH) [1, 2, 15], which can quickly compute shortest paths. In the CCH context, elimination orders are called contraction orders.

2 Pace 2020 Submission

The core of our submission is a combination of nested dissection [4] and FlowCutter. We use it to compute an elimination order, from which we derive an elimination tree. Nested dissection is a recursive scheme to compute an elimination order. First, a node separator is computed. This separator splits the graph into parts. Next, elimination orders are recursively computed for every part. The final elimination order is the concatenation of the parts' elimination orders followed by the nodes of the separator. The base case for our recursion are clique and tree graphs. For cliques, any order is optimal and thus the problem is trivial. For tree graphs, we use the optimal algorithm of [11]. If the graph is not a tree nor a clique, we compute a separator with a large expansion and a bounded imbalance using FlowCutter.

FlowCutter usually finds good separators regardless of the graph structure. However, it has a running time proportional to the product of the number of edges and the size of the separator. If there are small separators that FlowCutter can find, then FlowCutter has a nearly linear running time. Unfortunately, if the separator sizes found by FlowCutter are linear in the input graph size, then FlowCutter has quadratic running time. As a result, the time limit is reached before FlowCutter finishes.

To mitigate these problems, we implemented two additional approaches. One is based on a scheme inspired by the minimum degree heuristic [9, 5] and the Contraction Hierarchy node ordering heuristic [3]. The details are described in Section 3.

The other is very loosely based on label propagation [10], which tends to work well on graphs with large separators. It computes edge cuts. From these, we derive nodes separators by picking the nodes incident to the cut on one side. Together with nested dissection, an elimination order can be computed with this scheme. The edge cut algorithm is described in Section 4. We only use it, when we suspect that the graph has large separators.

In multiple steps, we try finding solutions using various approaches. First, we run the minimum degree heuristic variant to assure that we find some solution. If the achieved depth is above a threshold, we run the label propagation approach. Next, we run the edge cut variant of FlowCutter from which we derive node separators and apply nested dissection. The rational is that the edge cut FlowCutter variant is faster than the node variant.

Finally, we run the node separator variant of FlowCutter combined with nested dissection. This step is repeated until the timeout is reached. In every step, the random seed and tuning parameters are changed. For details about the tuning parameters, we refer to the code. We abort a nested dissection if the resulting tree would be higher than the best known tree.

3 Minimum Degree Heuristic

Most minimum degree heuristics use a node rating function $r(x)$. All nodes placed into a minimum priority queue ordered by $r(x)$. Nodes are then iteratively removed from the queue. After removing a node x, it is eliminated from the graph, i.e., edges among x's neighbors are inserted. Afterwards, the rating value $r(y)$ of all neighbors y of x must be recomputed. We update the positions of the neighbors y in the queue. This is repeated until the queue is empty. The order in which the nodes are removed from the queue is the computed elimination order. Usually, the rating function estimates something related to the height of a node.

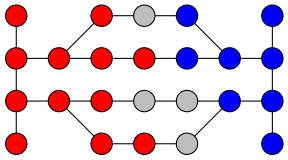


Figure 1 Example of Local Optimum. Initially, red nodes are left and blue and gray nodes are right. Grey nodes are moved by move-to-left round.

Our rating function is $r(x) = d(x) + 8 \cdot \ell(x)$. It has two terms $d(x)$ and $\ell(x)$. $d(x)$ is the degree of x in the graph after eliminating all nodes removed from the queue. $\ell(x)$ is an estimation of the height of x that is induced by the eliminated nodes. Initially, $\ell(x)$ is zero. When a node x is eliminated, we set $\ell(y)$ to $\max\{\ell(y), \ell(x) + 1\}$ for all neighbors y of x.

Inserting edges between all neighbors of a node x is slow if x has a high degree. We therefore abort the queue-based algorithm if all remaining nodes have a degree ≥ 150. If this happens, we sort the remaining nodes x by $r(x)$ and place them into the elimination order.

4 Label Propagation

We implement a graph bisection algorithm that is loosely based on label propagation. First, we describe the high-level label propagation scheme. Next, we explain the details of our algorithm. Finally, we discuss why this setup works on certain graphs.

Label propagation algorithms start with a simple and fast method to find an initial cut. This initial cut can be very large. It is then refined in many rounds. In every round, we iterate over all nodes x incident to the cut in random order. Using local information, we determine whether x should change sides. Rounds are executed until almost convergence is reached, i.e., the number of nodes changing side is small.

We refer to the sides as *left* and *right*. The *imbalance* of a cut is the size of the larger side divided by the node count. We aim to find a cut with few edges and at most an imbalance of 2/3. First, we pick two random seed nodes. From these, two simultaneous breath first searches are run. A node x is on the side whose search first reaches x.

We use three types of rounds named *decrease-cut-size*, *decrease-imbalance*, and *move-to-left*. We repeat all rounds until near convergence. The rounds only differ with respect to when a node moves between sides. In a decrease-cut-size round, a node moves if moving it decreases the cut size and the imbalance is at most 2/3. The objective is to only perform moves that immediately improve the cut size. In a decrease-imbalance round, a node additionally moves, if moving decreases the imbalance without increases the cut size. Finally, in a move-to-left round, a node moves to the left side, if the cut size remains the same and the imbalance is at most 2/3. In this round, no node moves to the right. We start by performing decrease-cut-size rounds, then do decrease-imbalance rounds, and finally do move-to-left rounds. This is repeated for a fixed number times. We finish with a decrease-imbalance round.

The decrease-cut-size and improve-balance rounds try to greedily improve our optimization criteria. However, they can run into local optima. One such situation is depicted by Figure 1. Initially, the red nodes form the left side and the gray and blue nodes the right side. The cut goes through path subgraphs. No single move improves balance or decreases cut size. However, a lot of nodes exist that can be moved without causing harm. Moving them randomly will move nodes aimlessly between left and right. If the paths are long enough, no progress is made with sufficient probability. To make progress, we need to tie-break these moves. We do this by moving as many nodes as possible to the left while maintaining

a 2/3-balance. This has a high chance of moving the grey nodes. In the next decrease-cut-size rounds, the cut size shrinks. The next improve-balance rounds find the symmetric structure on the top of the figure example. This concludes the description of our label propagation graph bisection algorithm.

5 Conclusion

We described our FlowCutter submission to the PACE 2020 tree-depth challenge. It is based on the original FlowCutter code. Two other independent competitors make use of reimplementations of the FlowCutter algorithm. These are Sallow and ExTREEm. Together these three submissions managed to win first, second, and third place. This shows that FlowCutter is a powerful tool to compute eliminations trees.

References

1 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. In *Symposium Experimental Algorithms SEA 2014*, volume 8504, pages 271–282. Springer, 2014. doi:10.1007/978-3-319-07959-2_23.

2 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. *ACM Journal of Experimental Algorithmics JEA*, 21(1):1.5:1–1.5:49, 2016. doi:10.1145/2886843.

3 Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Workshop on Experimental Algorithms, WEA 2008*, volume 5038, pages 319–333. Springer, 2008. doi:10.1007/978-3-540-68552-4_24.

4 Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.

5 Alan George and Joseph WH Liu. The evolution of the minimum degree ordering algorithm. *Siam review*, 31(1):1–19, 1989.

6 Michael Hamann and Ben Strasser. Graph bisection with pareto-optimization. In Michael T. Goodrich and Michael Mitzenmacher, editors, *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2016, Arlington, Virginia, USA, January 10, 2016*, pages 90–102. SIAM, 2016. doi:10.1137/1.9781611974317.8.

7 Michael Hamann and Ben Strasser. Graph bisection with pareto optimization. *ACM Journal of Experimental Algorithmics JEA*, 23, 2018. doi:10.1145/3173045.

8 Michael Hamann and Ben Strasser. Correspondence between multilevel graph partitions and tree decompositions. *Algorithms*, 12(9):198, 2019. doi:10.3390/a12090198.

9 Harry M Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3(3):255–269, 1957.

10 Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.

11 Alejandro A. Schäffer. Optimal node ranking of trees in linear time. *Inf. Process. Lett.*, 33(2):91–96, 1989. doi:10.1016/0020-0190(89)90161-0.

12 Ben Strasser. Computing tree decompositions with flowcutter: PACE 2017 submission. *CoRR*, abs/1709.08949, 2017. arXiv:1709.08949.

13 Ben Strasser. Flowcutter pace 2020 submission, 2020. URL: https://github.com/ben-strasser/flow-cutter-pace20.

14 Ben Strasser. Flowcutter pace 2020 submission, 2020. doi:10.5281/zenodo.3870928.

15 Ben Strasser and Dorothea Wagner. Graph fill-in, elimination ordering, nested dissection and contraction hierarchies. In *Gems of Combinatorial Optimization and Graph Algorithms*, pages 69–82. Springer, 2015. doi:10.1007/978-3-319-24971-1_7.

16 Sylwester Swat. Extreem source code. doi:10.5281/zenodo.3873126.

17 Marcin Wrochna. Sallow: a heuristic algorithm for treedepth decompositions. *CoRR*, abs/2006.07050, 2020. arXiv:2006.07050.

PACE Solver Description: Finding Elimination Trees Using ExTREEm - a Heuristic Solver for the Treedepth Decomposition Problem

Sylwester Swat
Institute of Computing Science, Poznan University of Technology, Poland
sylwester.swat@put.poznan.pl

Abstract

This article briefly describes the most important algorithms and techniques used in the treedepth decomposition heuristic solver called "ExTREEm", submitted to the 5th Parameterized Algorithms and Computational Experiments Challenge (PACE 2020) co-organized with the 15th International Symposium on Parameterized and Exact Computation (IPEC 2020).

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Treedepth decomposition, elimination tree, separator, PACE 2020

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.33

Supplementary Material The source code of ExTREEm solver is available at `https://doi.org/10.5281/zenodo.3873126`.

1 Problem description

A treedepth decomposition of a connected graph $G = (V, E)$ is a rooted tree $T = (V, E_T)$ such that every edge of G connects a pair of nodes that have an ancestor-descendant relationship in T. The solver briefly described here is a heuristic approach to the treedepth decomposition problem, where the goal is to find a treedepth decomposition for a given graph with as small height as possible.

2 Solver description

In this paper, we provide a short description of the most important algorithms implemented in solver ExTREEm. Due to many parameters used in the implementation and a lot of edge-cases that need to be taken into account, this description may not contain full information about the algorithms behavior in every possible situation.

Before we proceed to the actual description, let us fix some natural notations. For a given graph G we denote by $T(G)$ its treedepth decomposition. For a subset $S \subseteq V$ we denote by $C(G, S)$ the set of connected components of $G \setminus S$. For $a \in V$ we define $N(a) = \{v \in V : \{a, v\} \in E\}$ and for $A \subseteq V$ we take $N(A) = \bigcup_{v \in A} N(v)$.

Given a graph G, we find a separator S of G, then recursively obtain treedepth decompositions for components in $C(G, S)$, and finally merge separator S and found decompositions into an elimination tree of G. At the end, we apply some additional improvements to $T(G)$.

3 Separator evaluation

To assess the quality of a separator S we need to store values $mn(G, S)$ and $me(G, S)$ denoting, respectively, the maximum number of nodes and the maximum number of edges of a graph from $C(G, S)$. Now let us define

$$score_n(S) = |S| \cdot \frac{1 - \beta^{-\frac{\log |V|}{\log \beta}}}{1 - \beta}, \quad \text{where } \beta = \frac{mn(G,S)}{|V|}$$

$$score_e(S) = |S| \cdot \frac{1 - \gamma^{-\frac{\log |E|}{\log \gamma}}}{1 - \gamma}, \quad \text{where } \gamma = \frac{me(G,S)}{|E|}$$

$$score(S) = \alpha \cdot score_n(S) + (1 - \alpha) \cdot score_e(S), \quad \text{where } \alpha \in [0,1] \text{ is a parameter.}$$

A separator S is balanced if $mn(G,S) < b \cdot |V|$ or $me(G,S) < b \cdot |E|$, where $b \in (0,1)$ is a parameter. For given two balanced or two unbalanced separators S_1 and S_2, we say that S_1 is better than S_2 if $score(S_1) < score(S_2)$. A balanced separator is always better than an unbalanced one.

4 Preprocessing

The preprocessing phase consists of two steps. The first step consists in detecting some induced subgraphs of a given graph G that are isomorphic to a cactus graph. We do that by repeatedly choosing a node of degree at most 2, removing it from the graph and adding edges connecting its neighbors (unless already present). Considering the initial graph G induced by the set of removed nodes, we observe that all its connected components are cacti. For each such cactus component we find a treedepth decomposition using recursively the Articulation Point Separator Creator method (see 5.1). Roots of those decompositions are attached to a proper node in a decomposition of the remaining graph.

The second step of the preprocessing is based on finding a maximum independent set in a subgraph of G induced by a set containing all nodes that are "center nodes" of induced claw-subgraphs as well as all nodes of degree four whose neighborhood induces a connected graph in G. Each node from the found independent set is removed from G and all pairs of its neighbors are connected by an edge. After finding a decomposition of the obtained graph, nodes from the found independent set are attached to the deepest of their neighbors.

5 Separator creation

After preprocessing, we try to find a good separator. After generating candidates, we select five best ones (with respect to their scores) that are further subjected to a refinement process, called minimization. As a final separator we take the best one after the minimization.

5.1 Articulation Point Separator Creator

In this method, we find all articulation points (cut vertices) of a given graph. Then, for each articulation point v, we find values $mn(G,\{v\})$ and $me(G,\{v\})$. This is done in $O(|E|)$ time using an algorithm similar to Tarjan's algorithm for finding bridges and articulation points.

5.2 BFS Separator Creator

Given a set $B \subseteq V$, we run a standard breadth-first-search with source-nodes set B. By L_i we denote the i-th BFS layer, that is a set containing all vertices that are at distance i from the set B. For each of those layers we consider a graph $G_i = G[V \setminus (L_0 \cup \ldots \cup L_{i-1})]$. Then, we divide nodes in L_i into blocks, two nodes belong to the same block if they belong to the same connected component of G_i. For each such block X we find a minimum vertex cover of a bipartite graph induced by the edges between sets X and $(N(X) \cap L_{i+1})$. All blocks and vertex covers are treated as different separator candidates and are found in time $O(|E||V|^{\frac{1}{2}})$.

5.3 Component Expansion Separator Creator

We take some subset $B \subseteq V$ and then iteratively expand B by adding to it one node from $N(B) \setminus B$. In the first variant, we select a node with the tightest connection to B. In the second one, we select a node from B that has the least number of neighbors outside B, then add these neighbors to B in an arbitrary order. We obtain a sequence $(v_1, \dots v_n)$ of added nodes called an *expansion order*. We now consider separators $S_i = \{v_j : j \leq i, N(v_j) \cap (V \setminus \{v_1, \dots, v_i\}) \neq \emptyset\}$ and try to improve the given expansion order by taking smaller connected components of the partitioned graph before the larger ones. If we have already constructed separator S_i and we add v_{i+1} to that separator (and probably do some more changes) to obtain S_{i+1}, we afterwards rearrange all remaining nodes $v_{i+2}, \dots, v_n$ in such a way that nodes belonging to smaller connected components of $G[\{v_{i+2}, \dots, v_n\}]$ occur before all nodes that belong to larger connected components. All separators S_i are found in time $O(|E| \log |V|)$.

5.4 Flow Separator Creator

At the beginning, we select two sets of nodes that are possibly far from each other. This is done by creating a "landmark set", that is a set L obtained by first selecting a random node and then iteratively adding to L any node that is farthest from L. Then, we select two random nodes u and v from L and consider sets of the form $B = N(N(N(u)))$ and $E = N(N(N(v)))$. Afterwards, we find a maximal set of node-disjoint paths that begin in B and end in E. As a separator candidate we take the union of those paths, then minimize it with Greedy Minimizer. This method of creating separators works best in the context of Flow Minimizer.

5.5 FlowCutter Separator Creator

In this method, we use our own implementation of a slightly modified version of the FlowCutter algorithm (see [1]). The main idea remains the same - to expand the set of sources or targets and to avoid augmenting paths. As initial source nodes and target nodes we consider, as in 5.4, pairs of nodes from the "landmark set". Additionally, we admit certain imbalance in the source-reachable and target-reachable node sets only after the size of sources and targets reaches some fixed fraction of $|V|$. After a last node is marked as a source or a target, we consider four different expansion orders based on the order of adding graph nodes to sources and targets, and for each order we find separators using the Component Expansion Separator Creator method (5.3). We also consider as a separator a vertex cover of a graph induced by edges between the final sets of sources and targets.

6 Separator minimization

After creating separator candidates, we proceed to the refinement step - for each candidate S we exhaustively try to minimize the value of $score(S)$ using following methods:

1. Vertex Cover Minimizer – we find a vertex cover of a bipartite graph induced by edges between sets S and $N(S) \cap C_d$, where C_d is some subset of $C(G, S)$.
2. BFS Minimizer and Component Expansion Minimizer – we find separators using the methods from 5.2 and 5.3, respectively, with the initial source-set S.
3. Greedy Minimizer – we iteratively remove nodes from S, each time selecting a node which removal results in the minimal total size of the connected components adjacent to that node.

4. Flow Minimizer and FlowCutter Minimizer – we find separators using the methods from 5.4 and 5.5, respectively, with the initial sets of sources and targets set to some subsets of nodes that lie at a fixed distance from S.

7 Subtrees merging

After recursively finding decompositions for all components in $C(G, S) = \{c_1, \ldots, c_k\}$, we need to merge the results to obtain $T(G)$. To do that, we sort all components according to the nonincreasing depths of their decompositions. Then, we create a sequence S' by iteratively adding to S' nodes from $S \cap N(C_i)$ that were not added to S' earlier. As the tree $T(G)$ we initially take the path represented by sequence S', then we attach each tree $T(G[C_i])$ to the last node from sequence S' that occurs in $N(C_i)$. The whole procedure takes time $O(|E| \log |V|)$.

8 Tree improvements

When the tree $T(G)$ is created, we try to improve it by performing some structure-based changes. Those improvements are based on pivot-like operations.

8.1 Block pivots

By the block of $v \in V$ in a tree $T(G)$ we mean a maximal path in $T(G)$ which contains v, such that each node on that path, apart from the deepest one, has at most one son. We construct a separator S of G by taking all nodes from the root-block of T and recursively doing the same for the highest tree in the decomposition of $G \setminus S$, until $|S| > d \cdot H$, where $d \in [0, 1]$ is a parameter and H is the height of $T(G)$. Then, we merge separator S and all trees just as described in Section 7.

8.2 Hall-set pivots

Let us fix any block in $T(G)$ that lies on a longest leaf-root path P and let v be the topmost node in that block. Let $U(v)$ be the set of nodes on that path from the root to the parent of v and $D(v)$ be the remaining nodes on path P. By T_v we denote the subtree of T with root in v. We now consider a maximum matching M in a bipartite graph induced by edges between sets $U(v)$ and $N(U(v)) \cap R$, where R is either T_v or $T_v \setminus D(v)$.

If possible, we take a set $H_M \subseteq U(v)$ with the property $|H_M| > |N(H_M) \cap R|$ and check a treedepth decomposition obtained from T by removing nodes that belong to $N(H_M) \cap R$, setting those nodes as the root-block and performing some other necessary structural changes.

9 Availability

The source code of ExTREEm solver is available at `https://doi.org/10.5281/zenodo.3873126`.

References

1 Michael Hamann and Ben Strasser. Graph bisection with pareto-optimization. *CoRR*, abs/1504.03812, 2015. `arXiv:1504.03812`.

PACE Solver Description: Bute-Plus: A Bottom-Up Exact Solver for Treedepth

James Trimble
School of Computing Science, University of Glasgow, Scotland, UK
j.trimble.1@research.gla.ac.uk

Abstract

This note introduces *Bute-Plus*, an exact solver for the treedepth problem. The core of the solver is a positive-instance driven dynamic program that constructs an elimination tree of minimum depth in a bottom-up fashion. Three features greatly improve the algorithm's run time. The first of these is a specialised trie data structure. The second is a domination rule. The third is a heuristic presolve step can quickly find a treedepth decomposition of optimal depth for many instances.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Algorithm design techniques

Keywords and phrases Treedepth, Elimination Tree, Graph Algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.34

Related Version `https://arxiv.org/abs/2006.09912`

Supplementary Material DOI of code submitted to PACE Challenge: `https://doi.org/10.5281/zenodo.3881441`. The latest version of the code can be found on GitHub: `https://github.com/jamestrimble/pace2020-treedepth-solvers`.

Funding *James Trimble*: This work was supported by the Engineering and Physical Sciences Research Council (grant number EP/R513222/1).

Acknowledgements I would like to thank the PACE 2020 program committee and the optil.io team for a well-organised and enjoyable contest.

1 Introduction

A treedepth decomposition of graph G is a rooted forest F, such that if G has edge $\{u, v\}$ then either u is an ancestor of v or v is an ancestor of u in F. The treedepth problem is to determine, for a given graph G, the minimum depth of a treedepth decomposition of G, where depth is defined as the maximum number of vertices on a root-leaf path.

An *elimination tree* of a connected graph G is a special type of treedepth decomposition, defined recursively as follows. If G has a single vertex, its elimination tree equals G. Otherwise, let v be a vertex in G and let F be a forest consisting of an elimination tree for each component of $G - v$. Then an elimination tree of G is formed by making v the parent of every root of F.

For every connected graph G, there exists an elimination tree whose depth equals the treedepth of G ([5], chapter 6). To solve the treedepth problem, it is therefore sufficient to find an elimination tree of minimum depth. That is the approach taken by the Bute-Plus solver, which this paper introduces. The solver uses a positive-instance driven dynamic programming algorithm, which seeks sets of vertices that induce low-treedepth subgraphs of the input graph. Three additional features improve the performance of the algorithm: a specialised trie data structure, a domination rule based on a rule by Ganian et al. [2], and a heuristic presolver that quickly finds an optimal solution for many of the PACE Challenge instances.

The author submitted two other exact solvers to the PACE Challenge: Bute (which is Bute-Plus without the heuristic presolve step) and Bute-Plus-Plus (which spends additional time on the heuristic presolve and has a minor modification to the trie data structure). An earlier algorithm by the author [9], which constructs a treedepth decomposition from the top down, is very memory-efficient but is typically much slower than Bute-Plus.

2 A brief description of the algorithm

This section presents an outline of the Bute-Plus algorithm. We assume that the vertex set of a graph G, denoted $V(G)$, contains only integers. The neighbourhood of vertex v is denoted by $N(v)$. For a set of vertices S, $N(S)$ denotes the set of vertices that are not in S but are adjacent to some member of S.

The algorithm takes as input a connected graph G and returns the treedepth of G along with a treedepth decomposition of that depth. The optimisation problem is solved as a sequence of decision problems. The solver attempts to find an elimination tree of depth 1, then of depth 2, and so on until it is successful. (Typically, the higher-numbered decision problems are by far the most time consuming.)

For the decision problem of whether an elimination tree of depth k exists, the algorithm works downwards for $i = k, \ldots, 1$, finding all subsets S of $V(G)$ such that (1) S induces a subgraph of G with treedepth no greater than $k - i + 1$, (2) the neighbourhood of S has fewer than i vertices, and (3) the subgraph of G induced by S is connected. This collection of sets of vertices is called $\mathcal{S}_i^k$; it includes the vertex set of every subtree whose root is at depth i of an elimination tree of depth k of G.

The algorithm uses a positive-instance driven (PID) [7] approach to constructing the $\mathcal{S}_i^k$: rather than generating all subsets of $V(G)$ and checking if each one satisifies the three required properties, the elements of $\mathcal{S}_i^k$ are generated by joining together elements of $\mathcal{S}_{i+1}^k$. To be more precise, sets in $\mathcal{S}_i^k$ are constructed in two ways; a sketch of these follows. The first is simply by choosing vertices with a sufficiently small neighbourhood (since clearly each of these induces a connected subgraph of treedepth 1). The second is by finding a nonempty sub-collection $\mathcal{S} \subseteq \mathcal{S}_{i+1}^k$ and a vertex v satisfying the following conditions. The elements of $\mathcal{S}$ must be pairwise disjoint, and moreover there must not be an edge between vertices in any two distinct members of $\mathcal{S}$. Furthermore, v must have an edge to at least one vertex in each member of $\mathcal{S}$. These conditions guarantee that the set $\bigcup \mathcal{S} \cup \{v\}$ induces a connected subgraph of G of that admits an elimination tree with root v of depth no more than $k - i + 1$.

It is easy to verify using the definition of $\mathcal{S}_i^k$ that an instance of the decision problem is satisfiable if and only if $\mathcal{S}_1^k$ is non-empty (in which case $\mathcal{S}_1^k$ will have $V(G)$ as its only element). A small amount of extra bookkeeping allows the solver to output an optimal elimination tree.

Bute-Plus is not the first PID algorithm for treedepth. Bannach and Berndt [1] present a PID framework for computing a range of graph parameters including treedepth, treewidth, and pathwidth. Although their paper describes the family of algorithms in terms of a game theoretic characterisation of each problem, their algorithm for treedepth has a similar overall approach to that of Bute-Plus: both algorithms build up sets of vertices by combining one or more existing sets with a root vertex. Bannach and Berndt use a queue when combining sets whereas Bute-Plus uses a stack; a second difference is that the algorithm of Bannach and Berndt is not restricted to finding only elimination trees. The framework of Bannach and Berndt generalises a PID algorithm for treewidth by Tamaki which won the exact treewidth track of PACE 2016;[1] a second PID algorithm for treewidth by Tamaki [7] performed strongly in PACE 2017.

[1] `https://github.com/TCS-Meiji/treewidth-exact`

3 Improvements to the algorithm

The Bute-Plus solver has three additional features which greatly reduce run time on many instances. Two of these – a trie data structure and a domination rule – are described in the following two subsections. The third feature is a heuristic solver which is run for the first minute with the hope of finding a treedepth decomposition of optimal depth; this uses the Tweed-Plus solver which was an entry by the author in the heuristic track of PACE 2020 and is described in its own paper in this volume.

3.1 Trie data structure

Recall from Section 2 that the algorithm generates sets in the collection $\mathcal{S}_i^k$ by finding a subset of $\mathcal{S}_{i+1}^k$ along with a vertex v that together satisfy certain properties. For some of the PACE Challenge instances, $\mathcal{S}_{i+1}^k$ can contain millions of sets, and the task of finding appropriate subsets of the collection becomes intractable without a specialised data structure.

Bute-Plus's data structure supports two operations. The first is to add a $(S, N(S))$ pair – a set of vertices and its neighbourhood – to the collection. The second is a query operation which takes a set of vertices Q and an integer i. This returns all sets S in the collection such that both (1) $|N(S) \cup N(Q)| < i$ and (2) $(Q \cup N(Q)) \cap S = \emptyset$.

The data structure is implemented as a trie. When $(S, N(S))$ is inserted, $N(S)$ is sorted in ascending order and viewed as a string over the alphabet $V(G)$, then added to the trie. This approach has been used for the similar problem of superset queries several times in the past, for example in Savnik's Set-Trie [6].

To sketch the query operation: the algorithm performs a depth-first traversal of the trie, backtracking when it becomes clear that no value in the subtree is acceptable. For efficiency, each node of the trie stores the intersection of $N(S)$ values in the subtree rooted at that node; this idea is from a data structure for superset queries posted on Stack Overflow by Ben Tilly [8].

The task carried out by Bute-Plus's data structure is similar to the task of the *block sieve* designed by Tamaki for a PID treewidth solver [7]. Although both data structures are based on tries, their designs differ in several respects; for example, the block sieve data structure comprises a collection of tries rather than just one.

3.2 Domination rule

As discussed in Section 1, to find a minimum-depth treedepth decomposition it is sufficient to restrict attention to elimination trees. We can speed the algorithm up further by placing additional restrictions on acceptable elimination trees, if it can be shown that at least one tree in the restricted class has optimal depth.

For this purpose, the Bute algorithm uses a domination-breaking rule that extends a rule by Ganian et al. [2]. For distinct vertices v, w, we say that v *dominates* w if either of the following two conditions holds: (1) $N(v) \setminus \{w\} \supset N(w) \setminus \{v\}$; (2) $N(v) \setminus \{w\} = N(w) \setminus \{v\}$ and $w < v$. It is always possible to construct an elimination tree of minimum depth such that no vertex dominates any of its ancestors.

This rule allows us to further restrict each collection $\mathcal{S}_i^k$ to include only sets of vertices S such that no vertex in S dominates any member of $N(S)$.

4 Implementation details

The Bute-Plus solver is written in C. Sets of vertices are stored using bitsets; code from Nauty 2.6r12 [4] is used for the bitset data structure.[2] The Tweed-Plus heuristic presolver also uses code from Nauty for for the random number generator, and uses Metis 5.1.0 [3] to find nested dissection orderings.[3]

References

1 Max Bannach and Sebastian Berndt. Positive-instance driven dynamic programming for graph searching. In *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, volume 11646 of *Lecture Notes in Computer Science*, pages 43–56. Springer, 2019. doi:10.1007/978-3-030-24766-9_4.

2 Robert Ganian, Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. SAT-encodings for treecut width and treedepth. In *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019, San Diego, CA, USA, January 7-8, 2019.*, pages 117–129. SIAM, 2019. doi:10.1137/1.9781611975499.10.

3 George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20(1):359–392, 1998. doi:10.1137/S1064827595287997.

4 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

5 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

6 Iztok Savnik. Index data structure for fast subset and superset queries. In *Availability, Reliability, and Security in Information Systems and HCI - IFIP WG 8.4, 8.9, TC 5 International Cross-Domain Conference, CD-ARES 2013, Regensburg, Germany, September 2-6, 2013. Proceedings*, volume 8127 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2013. doi:10.1007/978-3-642-40511-2_10.

7 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *J. Comb. Optim.*, 37(4):1283–1311, 2019. doi:10.1007/s10878-018-0353-z.

8 Ben Tilly. Fast data structure for finding strict subsets (from a given list). Stack Overflow. Version: 2011-06-29. URL: https://stackoverflow.com/a/6514445/3347737.

9 James Trimble. An algorithm for the exact treedepth problem. In *18th International Symposium on Experimental Algorithms, SEA 2020, June 16-18, 2020, Catania, Italy*, volume 160 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SEA.2020.19.

[2] Nauty is available at http://pallini.di.uniroma1.it/

[3] Metis is available at http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

PACE Solver Description: Tweed-Plus: A Subtree-Improving Heuristic Solver for Treedepth

James Trimble
School of Computing Science, University of Glasgow, Scotland, UK
j.trimble.1@research.gla.ac.uk

Abstract

This paper introduces Tweed-Plus, a heuristic solver for the treedepth problem. The solver uses two well-known algorithms to create an initial elimination tree: nested dissection (making use of the Metis library) and the minimum-degree heuristic. After creating an elimination tree of the entire input graph, the solver continues to apply nested dissection and the minimum-degree heuristic to parts of the graph with the aim of replacing subtrees of the elimination tree with alternatives of lower depth.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Algorithm design techniques

Keywords and phrases Treedepth, Elimination Tree, Heuristics

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.35

Supplementary Material DOI of code submitted to PACE Challenge: `https://doi.org/10.5281/zenodo.3881441`. The latest version of the code can be found on GitHub: `https://github.com/jamestrimble/pace2020-treedepth-solvers`.

Funding *James Trimble*: This work was supported by the Engineering and Physical Sciences Research Council (grant number EP/R513222/1).

Acknowledgements Many thanks to the program committee of PACE 2020 and the optil.io team for an enjoyable contest.

1 Introduction

A *treedepth decomposition* of graph $G = (V, E)$ is a rooted forest F with node set V, such that for every edge $\{u, v\} \in E$ we have either that u is an ancestor of v in F or v is an ancestor of u in F. The *depth* of a treedepth decomposition is the maximum number of vertices in a path from the root to a leaf. For example, if G is the graph at the left of Figure 1 then each of the two trees in the figure is a treedepth decomposition of G; the first has depth 5 and the second has depth 4.

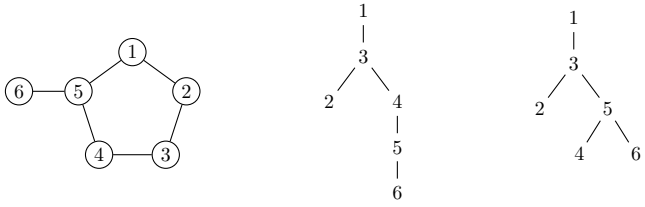


Figure 1 A graph G and two treedepth decompositions of G.

In his PhD dissertation, Fernando Sánchez Villaamil suggests as future work the following strategy for reducing the depth of a treedepth decomposition. "It would be possible to throw away a part of the decomposition and compute a new treedepth decomposition for this part of the graph. ... This suggests a straightforward way of using different heuristics on different parts of the graph." ([5], page 118)

This paper introduces the heuristic solver *Tweed-Plus*, which seeks to find a treedepth decomposition of low depth for a given connected graph. The solver's overall strategy is exactly the one proposed by Sánchez Villaamil, with the parts of the decomposition that are thrown away and replaced being subtrees.

I will use the term *subtree replacement* to denote this process of replacing a subtree of a treedepth decomposition with a new decomposition of the corresponding part of the input graph. To give an example of this process, let G be the graph in Figure 1, and let T_0 and T_1 be the two treedepth decompositions of G shown beside it. To transform T_0 into T_1, the subtree rooted at vertex 4 – which has vertex set $\{4, 5, 6\}$ – is replaced with a different tree on the same set of vertices; moreover, this tree is a treedepth decomposition of the subgraph of G induced by $\{4, 5, 6\}$. In general, the process of subtree replacement is as follows. First, a subtree T rooted at some non-root vertex v is removed from the initial treedepth decomposition. Next, a new forest T' on the vertex set of T is found; this forest must be a treedepth decomposition of the subgraph of G induced by the vertex set of T. Finally, each root of T' is made a child of the vertex that was v's parent in the initial treedepth decomposition.

The algorithm described in this paper depends on the fact, formalised in the following proposition, that the process of subtree replacement maintains the treedepth decomposition property.

▶ **Proposition 1.** *Let F_0 be a treedepth decomposition of a graph G, and let F_1 be the result of applying a subtree replacement to F_0. Then F_1 is also a treedepth decomposition of G.*

Proof. Let $\{u, v\}$ be an edge in G. Without loss of generality, assume that u is an ancestor of v in F_0. There are three cases. We show that in each one, u and v have an ancestor-descendant relationship in F_1; thus, F_1 is a treedepth decomposition of G.

- *Case 1: Neither u nor v is in the replaced part of the decomposition.* The path between u and v in the decomposition is not changed by the subtree replacement. Therefore u remains an ancestor of v in F_1.
- *Case 2: Vertex v is in the replaced part of the decomposition, but u is not.* Let w be the parent in F_0 of the root of the replaced subtree. In F_1, vertex u remains either equal to w or an ancestor of w, since both u and w are in the unchanged part of the decomposition. There must also be a path from w to v in F_1. Therefore u is an ancestor of v in F_1.
- *Case 3: Both u and v are in the replaced part of the decomposition.* By the definition of subtree replacement, the new part of the decomposition is itself a treedepth decomposition of part of G containing u, v, and an edge between these two vertices. Therefore, either u is an ancestor of v in F_1 or vice versa. ◀

If a subtree replacement replaces a subtree with a forest of strictly smaller depth, we call this replacement a *subtree improvement*.

Tweed-Plus makes use of two solvers – which will be referred to as the *sub-solvers* – each of which is itself a heuristic for the treedepth problem. The first sub-solver is a a variant of the well-known *minimum-degree heuristic*; the second is a small wrapper around the Metis library [2]. The overall strategy of the Tweed-Plus solver is to generate an initial treedepth decomposition of the input graph using one of the sub-solvers, then to make repeated additional calls to the sub-solvers with the aim of making subtree improvements.

2 The Sub-Solvers

Recall that each sub-solver called by Tweed-Plus is itself a heuristic algorithm for finding a treedepth decomposition. Both of the sub-solvers produce an *elimination tree* of the input graph. This is valid because every elimination tree is also a treedepth decomposition ([4], chapter 6). Note that the Tweed-Plus algorithm's overall strategy would still produce valid treedepth decompositions even if a different collection of sub-solvers were used, some or all of which produced treedepth decompositions that were not also elimination trees.

To give one of the several equivalent definitions, an elimination tree of connected graph G is any tree created by the following procedure. Visit the vertices of G one by one. On visiting vertex v, carry out the following two steps. First, modify G by adding all of the edges required to make the set of unvisited neighbours of v a clique. Second, find the set of visited neighbours of v that do not yet have a parent in the elimination tree, and make v the parent in the tree of each member of this set.

The first sub-solver uses the well-known minimum-degree heuristic [1], which constructs an elimination tree using the process described in the definition above. At each step, one of the vertices of minimum degree in the current graph is selected at random to be visited. Unlike the standard version of the algorithm, my implementation includes edges incident to previously-visited vertices in the degree calculation. (I have not carried out a rigorous investigation into the effect of this change.) Another small tweak is that on some calls to the minimum-degree algorithm, degrees are divided by a small integer and rounded down, in order to enlarge the set of "minimum degree" vertices that may be chosen.

The second sub-solver calls the Metis library [2] which constructs an elimination tree in a top-down fashion by nested dissection (a recursive process of finding small vertex separators to partition the graph). The Metis library returns an ordering of vertices for the elimination tree; the sub-solver simply constructs an elimination tree using this ordering.

3 The Tweed-Plus Algorithm

Let G be the input graph. The Tweed-Plus algorithm performs the following steps. First, one of the sub-solvers is called to produce an initial treedepth decomposition T of G. (Since G is assumed to be connected, T must be a tree rather than a forest composed of multiple trees.) Following this, the algorithm finds a subtree T' of T that contains a leaf of maximum depth in T; the subtree T' is a candidate for subtree improvement. One of the sub-solvers is then called in an attempt to find a replacement for T' that has lower depth. If such a subtree is found, this is used to replace T' in the overall decomposition T.

The algorithm periodically discards T completely, computes a new initial treedepth decomposition, and again carries out the process of attempted subtree improvements. Whenever the current decomposition has lower depth than any decomposition previously found, this decomposition is recorded as the incumbent best solution.

The algorithm as implemented is intended to be run for 30 minutes, as this is the time limit of the PACE Challenge. Two minutes before this time limit is reached, the best treedepth decomposition found so far is reloaded from memory, and additional subtree improvements are attempted until the time limit is reached. This simple strategy of returning to the best solution found so far for a final, relatively long improvement phase appears to be useful in some cases. For example, it brings the best depth found down from 138 to 135 on PACE Challenge public instance 069; no other solver entered into the challenge gives as good a result on this instance.

There are many small details of the algorithm that have not yet been described: How are candidate subtrees for improvement chosen? How many attempts at improvement are carried out before beginning with a new initial decomposition? When is each sub-solver used? Each of these decisions was made by hand-tuning to the PACE Challenge public instances, and further improvements are without doubt possible. To give a sketch of the choices made: Both sub-solvers are used (in different iterations) for producing an initial decomposition, and both are also used for attempting subtree improvements. Candidate subtrees for improvement always contain the lowest-indexed vertex of maximum depth, and have depth ranging from 1 to one less than the depth of the full treedepth decomposition. On early iterations, subtree improvements are not attempted at all; on later iterations, an increasing number of attempts at subtree improvements are made.

4 Implementation details

The Tweed-Plus algorithm is implemented in C. Two different representations of vertex neighbourhoods are used in an effort to save memory and reduce run time: small adjacency lists (up to 32 vertices) are stored as unsorted lists, while larger adjacency lists are stored as bitsets. Nevertheless, the implementation of the minimum-degree heuristic lacks many of the improvements that have been proposed in the literature [1].

The solver uses code from Nauty 2.6r12 [3] for the bitset data structure and random number generator (the latter of which is based in turn on code by Donald Knuth).[1] Metis 5.1.0 [2] is used to find a nested dissection ordering.[2]

5 Conclusion

This paper has briefly introduced the Tweed-Plus solver, which uses a portfolio of heuristic treedepth algorithms both for constructing an initial solution and for seeking subtree improvements. An interesting direction for future research would be to add several of the leading solvers from the PACE 2020 heuristic track as additional sub-solvers.

References

1 Alan George and Joseph W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989. doi:10.1137/1031001.

2 George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20(1):359–392, 1998. doi:10.1137/S1064827595287997.

3 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

4 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

5 Fernando Sánchez Villaamil. About treedepth and related notions. PhD thesis, RWTH Aachen University, Germany, 2017. URL: https://tcs.rwth-aachen.de/~sanchez/about-treedepth-and-related-notions.pdf.

[1] Nauty is available at http://pallini.di.uniroma1.it/.
[2] Metis is available at http://glaros.dtc.umn.edu/gkhome/metis/metis/overview/.

PACE Solver Description: Sallow: A Heuristic Algorithm for Treedepth Decompositions

Marcin Wrochna
University of Oxford, UK
mwrochna@gmail.com

Abstract

We describe a heuristic algorithm for computing treedepth decompositions, submitted for the PACE 2020 challenge. It relies on a variety of greedy algorithms computing elimination orderings, as well as a Divide & Conquer approach on balanced cuts obtained using a from-scratch reimplementation of the 2016 FlowCutter algorithm by Hamann & Strasser [2].

2012 ACM Subject Classification Theory of computation → Discrete optimization; Mathematics of computing → Solvers; Mathematics of computing → Graph theory

Keywords and phrases treedepth, decomposition, heuristic, weak colouring numbers

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.36

Related Version A full version of the paper is available at `https://arxiv.org/abs/2006.07050`.

Supplementary Material Code: `http://doi.org/10.5281/zenodo.3870565`.

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 714532, PI: Stanislav Živný).

Acknowledgements The author is very grateful to the PACE 2020 organizers at the University of Warsaw, organizers of past editions, and the OPTIL.io team at the Poznań University of Technology for making this challenge possible.

1 Orderings and elimination

We start by recalling a few useful notions and facts (experts will recognize we are essentially describing the well-known statement that $\mathrm{td}(G) = \mathrm{wcol}_\infty(G)$, see e.g. [3, Lemma 6.5]).

Treedepth has many equivalent definitions. Small treedepth can be certified as usual by a treedepth decomposition (also known as a Trémaux tree) – it suffices to specify the `parent` of each vertex in the tree. A corresponding `ordering` is any linear ordering of vertices such that parents come before children – it can be obtained from a `parent` vector by any topological sorting algorithm, for example. In turn, any linear `ordering` of vertices can be turned into a treedepth decomposition by an elimination process: repeatedly remove the last vertex in the ordering and turn its neighbourhood into a clique. The `parent` of the removed vertex is set to the latest vertex in the neighbourhood.

It is easy to check this results in a new valid treedepth decomposition. Moreover, turning a decomposition into an ordering and back cannot increase the depth. To see this, observe that by induction, at any point in the elimination process, the neighbourhood of the removed vertex consists only of its ancestors (in the original decomposition), because all later vertices were removed, hence all introduced edges are still in the ancestor-descendant relationship. In particular the new parent of each vertex is an ancestor in the original decomposition.

A more static look at the elimination process is through *strongly* and *weakly* reachable vertices. Fix a vertex v. A vertex x is in the neighbourhood of v at the moment v is eliminated if and only if x is earlier in the ordering and can be reached, in the original

graph, via a path whose internal vertices are later than v in the ordering (= have been eliminated). We say x is *strongly reachable* from v (in the given graph and ordering). So this neighbourhood is the set of strongly reachable vertices (in the graph G with ordering L), usually denoted $\mathrm{SReach}_\infty[G, L, v]$. Similarly x is *weakly reachable* from v if x is earlier and can be reached via a path whose internal vertices are later than x (instead of "later than v"). Equivalently, this relation is the transitive closure of strong reachability. The set of weakly reachable vertices is denoted $\mathrm{WReach}_\infty[G, L, v]$ – it is equal to the set of ancestors of v in the treedepth decomposition obtained by elimination.

To give some context, the maximum size of $\mathrm{SReach}_\infty[G, L, v]$ over vertices v is the *strong ∞-colouring number* of (G, L) and its minimum over all orderings L is equal to $\mathrm{tw}(G) + 1$ [1, Theorem 3.1]. The maximum size of $\mathrm{WReach}_\infty[G, L, v]$ over vertices v (hence the depth of the resulting treedepth decomposition) is the *weak ∞-colouring number* of (G, L) and its minimum over all orderings L is equal to $\mathrm{td}(G)$. The ∞ here is customary because using paths of length at most $k < \infty$ leads to other notions important in sparse graph theory, see e.g. [6].

A third, more efficient look at the elimination process comes from the observation that descendants of v, in the resulting treedepth decomposition, are exactly vertices reachable in the subgraph induced by vertices later than or equal to v (in the ordering). We can thus define a *building process* on an ordering as follows: we process vertices starting from the last, maintaining connected components of the subgraph induced by vertices processed so far. This is sufficient to build the same treedepth decomposition, without changing the graph: we maintain the treedepth decomposition of the subgraph induced by processed vertices (using a `parent` vector) and represent each component by the root of the corresponding tree (equivalently, the earliest vertex of the component). When processing a vertex v, for each neighbour y later than v in the ordering, to update components we only have to merge y's component with v (initially a singleton). To update the decomposition, we find the root of y's component and make it a child of v, which thus becomes the new root. Thus y's parent is the latest weakly reachable vertex, as expected.

In other words, the building process consider vertices in the same order as the elimination process. However, in the elimination process we maintain a graph on unprocessed vertices and when eliminating the next vertex, we replace its neighbourhood by a clique, which is somewhat costly. Instead, in the building process we maintain a graph on more and more processed vertices (hence the name), or rather a structure to represent its connected components.

2 Greedy algorithms

The above processes suggest simple heuristics: we can start with vertices ordered decreasingly by any notion of centrality (e.g., the degree in the original graph), since we expect higher vertices in optimal treedepth decompositions to be more "central". Moreover, we can update this "centrality score" of unprocessed vertices on the fly: we maintain a heap of unprocessed vertices with the minimum score at the top, popping and processing a vertex until the heap is empty.

By elimination

In the elimination process, we update the score of a vertex v based on a linear combination of: 1. its height ($1+\max$ height of neighbours eliminated so far; once we decide to eliminate v this becomes the height of the subtree rooted at v in the resulting decomposition); 2. its degree (in the partially eliminated graph; once we decide to eliminate v this becomes $|\mathrm{SReach}_\infty[G, L, v]|$ in the resulting ordering); 3. some initial, static score.

Consider a graph with n vertices, m edges, and suppose we stop when unable to obtain a decomposition of depth better than d. In the elimination process, we can then assume that the neighbourhood of every vertex at every step is smaller than d. Simulating it then requires $\Theta(nd^2)$ time in many cases: e.g. if most vertices have a neighbourhood of size $\Theta(d)$ at the time they are processed (as in $K_{d,n}$), ensuring that this neighbourhood becomes a clique takes $\Theta(d^2)$ time. This bound is also sufficient; to do the simulation we maintain neighbourhood lists of the partially eliminated graph as `std::vector`s sorted by vertex name (not by the ordering we're about to compute), so that the union of neighbourhoods can be computed by merge-sort (when turning v's neighbourhood into a clique). Note however that we cannot compute parents on the fly, since the ordering of unprocessed vertices is not yet decided; we do this in a second pass, using the faster building approach once the ordering is fixed. See Algorithm 1 in the full version.

The memory requirement is $\Theta(nd)$ in the worst case and this cannot be improved to $\mathcal{O}(m)$, because a random 3-regular graph on d vertices will have, after eliminating half of its vertices, a clique on $\Theta(d)$ vertices and $\Theta(d^2)$ edges (due to its expansion properties). This means memory usage can also be prohibitive for large graphs with expected treedepth d much larger than the average degree.

By building

The $\Theta(nd^2)$ time and $\Theta(nd)$ space bound of the elimination process can be prohibitive for huge graphs of large treedepth. Instead, the building process can be simulated in $\mathcal{O}(\min(m \cdot \alpha(n), nd))$ time and $\mathcal{O}(m)$ space by maintaining components with the classic union-find data structure (where α is the inverse Ackerman function [5] and the latter bound follows from the fact that for each vertex, its pointer in the structure only goes up the tree). We note that this also allows to check the correctness of a treedepth decomposition (by replacing the assignment $\texttt{parent}[y] := v$ with whatever the original parent was and checking it is a descendant of v) in the same running time; this can be significantly faster than the straightforward $\mathcal{O}(md)$ method when d is large.

The details are similar as in Algorithm 1, see Algorithm 2 in the full version. One change is that $\texttt{g}[v]$ does not represent the neighbourhood of a vertex after elimination; instead, it represents the graph after contracting processed components. For a root vertex r of a component C (starting from singleton components), $\texttt{g}[r]$ stores the neighbours of that component. For a non-root vertex $\texttt{g}[v]$ is cleared: this guarantees that the total size never increases. This also means the α part of the score is less meaningful; using $\alpha \cdot |g[x]|$ below would be the same as $\alpha \cdot |N_G(x)|$ (the original degree, since x is not processed yet). Instead we use $\alpha \cdot \max(|g[x]|, |g[v]|)$ as a slightly better heuristic. This does result in noticeably worse results compared to the elimination version.

Moreover, we maintain a union-find structure with pointers $\texttt{ancestor}[v]$. We decided not to balance unions by size or rank, instead of opting for a simpler and more natural choice: $\texttt{ancestor}[v]$ is always some ancestor in the treedepth decomposition computed so far ($\texttt{ancestor}[v]$ is $\perp$ for unprocessed vertices) – we expect these to be shallow anyway.

Fast versions with lookahead

In Algorithm 2 the cost of computing $\texttt{g}[v]$ is still significant (though much lower compared to the elimination version). A super-fast version can be obtained by removing $\texttt{g}[v]$; however the `height` of unprocessed vertices cannot be maintained exactly then. In that case the heap is useless and we can simply do the building process with a fixed ordering by initial score.

However, we can significantly improve this super-fast version with a simple lookahead. Instead of processing the last unprocessed vertex, we check the last ℓ unprocessed vertices, compute what their height would be at this point, and choose the minimum height. For $\ell = 2$ this is almost as fast as a DFS; for $\ell = 64$ this is still faster than other versions and results in significantly better depth than DFS, often giving a reasonable ballpark estimate. Nevertheless we essentially always use the full version of Algorithm 2 as well, unless we know that the super-fast estimates are good enough (e.g. in recursive runs where other branches are already deeper).

A similar idea can be used to get the best of the elimination and building versions. The problem with the building version is that we do not have access to the degree of a vertex after eliminations, for evaluating the heuristic score. A work-around is to do this evaluation exactly (by computing unions of neighbourhoods) for a few vertices close to the top of the heap. In fact, a re-evaluation can only increase the score, pushing a vertex down, so it suffices to re-evaluate and update the top vertex of the heap some constant ℓ number of times (completely forgetting the computed unions of neighbourhoods afterwards). We can stop as soon as the top vertex stays at the top after re-evaluation, so even high constants ℓ turn out to be quite affordable. For $\ell = 1024$, this results in an algorithm that seems just as good as greedy by elimination, yet avoids the heavy memory usage in huge graphs of large treedepth.

3 Divide & Conquer

The other main component of the submitted algorithm is to divide & conquer: find a possibly small balanced cut, remove it, recurse into connected components, and output a treedepth decomposition with the cut arranged in a line above the recursively obtained decompositions. To find balanced cuts we use the FlowCutter algorithm submitted for the PACE 2016 challenge by Ben Strasser. It is a crucial part here as well, but the idea and details are already very well described in a paper by Hamman and Strasser [2] (see also some further details in [4]).

A final feature is that of cutoffs. A bad cutoff d means we abandon any attempt that won't lead to a decomposition of depth strictly smaller than d. A good cutoff d means we return as soon as it can output a decomposition of depth at most d. We use e.g. the best know decomposition's depth as a bad cutoff and the maximum depth in sibling branches computed so far as a good cutoff.

Further ideas and details are deferred to the full version (`arXiv:2006.07050`).

References

1 Stefan Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability – a survey. *BIT Numerical Mathematics*, 25(1):2–23, 1985. `doi:10.1007/BF01934985`.

2 Michael Hamann and Ben Strasser. Graph bisection with pareto optimization. *ACM Journal of Experimental Algorithmics*, 23, 2018. `doi:10.1145/3173045`.

3 Jaroslav Nešetřil and Patrice Ossona de Mendez. Bounded height trees and tree-depth. In *Sparsity*, pages 115–144. Springer, 2012. `doi:10.1007/978-3-642-27875-4_6`.

4 Ben Strasser. Computing tree decompositions with flowcutter: PACE 2017 submission. *CoRR*, abs/1709.08949, 2017. `arXiv:1709.08949`.

5 Robert Endre Tarjan and Jan van Leeuwen. Worst-case analysis of set union algorithms. *J. ACM*, 31(2):245–281, 1984. `doi:10.1145/62.2160`.

6 Jan van den Heuvel, Patrice Ossona de Mendez, Daniel Quiroz, Roman Rabinovich, and Sebastian Siebertz. On the generalised colouring numbers of graphs that exclude a fixed minor. *Eur. J. Comb.*, 66:129–144, 2017. `doi:10.1016/j.ejc.2017.06.019`.

The PACE 2020 Parameterized Algorithms and Computational Experiments Challenge: Treedepth

Łukasz Kowalik
Institute of Informatics, University of Warsaw, Poland
kowalik@mimuw.edu.pl

Marcin Mucha
Institute of Informatics, University of Warsaw, Poland
mucha@mimuw.edu.pl

Wojciech Nadara
Institute of Informatics, University of Warsaw, Poland
w.nadara@mimuw.edu.pl

Marcin Pilipczuk
Institute of Informatics, University of Warsaw, Poland
malcin@mimuw.edu.pl

Manuel Sorge
Institute of Informatics, University of Warsaw, Poland
manuel.sorge@mimuw.edu.pl

Piotr Wygocki
Institute of Informatics, University of Warsaw, Poland
p.wygocki@mimuw.edu.pl

Abstract

This year's Parameterized Algorithms and Computational Experiments challenge (PACE 2020) was devoted to the problem of computing the treedepth of a given graph. Altogether 51 participants from 20 teams, 12 countries and 3 continents submitted their implementations to the competition.

In this report, we describe the setup of the challenge, the selection of benchmark instances and the ranking of the participating teams. We also briefly discuss the approaches used in the submitted solvers and the differences in their performance on our benchmark dataset.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases computing treedepth, contest, implementation challenge, FPT

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.37

Funding This work has been supported by the ERC Starting Grant TOTAL, Grant Agreement No 677651 (Ł.K., M.M), the ERC Starting Grant CUTACOMBS, Grant Agreement No 714704 (W.N., M.P., M.S.) and by the Polish National Science Center Grant PRELUDIM 2018/29/N/ST6/00676 (P.W.). The publication of proceedings of PACE 2020 has been supported by the University of Warsaw and the European Research Council (ERC) via European Union's Horizon 2020 research and innovation programme Grant Agreement no. 714704.

Acknowledgements The PACE challenge was supported by Networks [35] and University of Warsaw. The prize money (4000€) was given through the generosity of Networks [35]. We are grateful to the whole `optil.io` team, led by Szymon Wasik, and especially to Jan Badura for the fruitful collaboration and for hosting the competition at `optil.io` on-line judge system. Special thanks go to Felix Reidl, who designed the eye-catching PACE 2020 poster.

1 Introduction

The Parameterized Algorithms and Computational Experiments Challenge (PACE) is an annually held algorithm engineering competition conceived in Fall 2015 to deepen the relationship between parameterized algorithms and practice.

So far, four iterations of PACE were organized. In each iteration, one or two NP-hard computational problems were chosen and the goal was to prepare an implementation which is able to solve (either exactly or approximately) challenging instances from various application domains in a decent time. The problems included treewidth [12,13], minimum feedback vertex set [12], minimum fill-in [13], Steiner tree [7], vertex cover [16], and hypertree width [16]. These challenges have a significant impact on the research community. Indeed, according to Google Scholar previous PACE reports are cited more than 90 times, in particular by research articles based on concrete implementations competing in previous editions of PACE, published in conferences like ALENEX, SEA, WADS, and ESA (where the best paper award was given in 2017 to a PACE-related work of H. Tamaki [50]).

In this article, we report on the fifth iteration of PACE. The topic of PACE 2020 was computing the treedepth of a graph (see Section 2 for a definition). The challenge was partitioned into two tracks. In the *exact track*, the implementations were supposed to return only optimal solutions and the goal was to maximize the number of solved instances (with total computation time as a tiebreaker). In the *heuristic track*, the implementations were supposed to solve larger instances than in the exact track, but non-optimal solutions were allowed and the goal was to provide solutions which are, on average, better than the solution of others.

The PACE 2020 challenge was announced on 25th October 2019. On December 16th public instances were made available and beginning from 13th March 2020 it was possible to test solutions on the public instances via the `optil.io` platform, which provided also a provisional ranking. The final version of the submissions was due on 1st June 2020. The results were announced on 24th June 2020. The award ceremony is going to take place during the International Symposium on Parameterized and Exact Computation (IPEC 2020) which was supposed to take place in Hong Kong, but due to the COVID-19 pandemic will be held online.

For the first time, short descriptions of the top five solvers in each track are contained as standalone documents in the proceedings of IPEC. Some of them will likely inspire or evolve into research papers. Indeed, at the moment of writing this report, we were already able to identify two such cases, see [53] and [62].

2 Computing Treedepth: Theory and Practice

Treedepth plays a major role in structural graph theory, in particular, the theory of sparse graph classes [31–33]. It is more restrictive than its more well-known counterparts *treewidth* and *pathwidth*, but still graphs of bounded treedepth form quite a rich family of graphs.

Definition

Remarkably, treedepth admits a number of equivalent definitions. Probably the best known is the one using embeddings into a rooted forest. A *rooted forest* is a graph whose every connected component is a tree with a designated root and the *depth* of a rooted forest is the maximum number of vertices on a root-to-leaf path. For a graph G, a *treedepth decomposition* of G consists of a rooted forest F and a bijection $\phi : V(G) \to V(F)$ such that for every

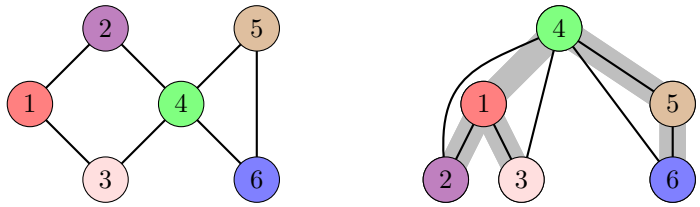


Figure 1 The thick gray tree on the right is an exemplary treedepth decomposition (aka elimination tree) of the graph on the left. Note that all graph edges go bottom-up in the tree.

$uv \in E(G)$, $\phi(u)$ and $\phi(v)$ are in descendant-ancestor relation in F. The *depth* of a treedepth decomposition (F, ϕ) is the depth of F and the *treedepth* of a graph G, denoted $\mathrm{td}(G)$, is the minimum possible depth of its treedepth decomposition.

A basic but important observation about treedepth is that if (F, ϕ) is a treedepth decomposition of a graph G and T is a tree in F with root r, then removing $\phi^{-1}(r)$ from G disconnects vertices whose images under ϕ are in different subtrees of T rooted in the children of r. This leads to the following equivalent recursive definition of treedepth:

$$\mathrm{td}(G) = \begin{cases} 0 & \text{if } V(G) = \emptyset, \\ 1 + \min_{v \in V(G)} \mathrm{td}(G - \{v\}) & \text{if } G \text{ is connected}, \\ \max_{C \in \mathtt{cc}(G)} \mathrm{td}(C) & \text{if } G \text{ is disconnected}. \end{cases}$$

Here, $\mathtt{cc}(G)$ denotes the family of connected components of G.

The above definition can be also interpreted as a game between two players, say Breaker and Chooser. The arena of the game is an induced subgraph of the input graph G, initially the whole graph G. At each round, Breaker first deletes a vertex of the current graph, and then Chooser restricts the arena to one of the connected components of the current graph. The game ends when the current graph becomes empty; Breaker wants to end the game in the minimum number of rounds, and Chooser in the maximum number of rounds. It is immediate from the above recursive definition of treedepth that, if both players play optimally, the game will end in exactly $\mathrm{td}(G)$ rounds.

Because of the above definition and the intuition of treedepth as an "elimination game", where one can pay 1 to delete a vertex from the graph and then recurse independently over connected components, treedepth decompositions are sometimes called also *elimination forests* (or *elimination trees* if G is not connected).

Two related equivalent definitions of treedepth come from the theory of sparse graph classes. Let G be a graph. A function $\alpha : V(G) \to \mathbb{N}$ is a *centered coloring* if for every connected subgraph H of G, H contains a vertex of unique color, i.e., there is $i \in \mathbb{N}$ with $|\alpha^{-1}(i) \cap V(H)| = 1$. Furthermore, α is a *vertex ranking* if this unique color i is actually equal to $\max\{\alpha(v) \mid v \in V(H)\}$. In the context of a centered coloring, the values $\alpha(v)$ are called *colors* and in the context of a vertex ranking, they are called *ranks*. It is not difficult to see that the minimum number of colors used for a centered coloring of G and the minimum number of ranks used for a vertex ranking are both equal and equal to the treedepth of G.

Algorithms

From the theory point of view, the complexity of computing or approximating treedepth is much less understood than for treewidth.

The recursive definition of treedepth can be also interpreted as a recipe for a dynamic programming algorithm computing the treedepth of G. This yields a very simple $2^n \cdot n^{\mathcal{O}(1)}$-time algorithm.

Reidl et al. [37] described a dynamic programming algorithm that, given a graph G and a tree decomposition of G of width t, finds $\mathrm{td}(G)$ in time $2^{\mathcal{O}(\mathrm{td}(G)\cdot t)}n^{\mathcal{O}(1)}$. One can pipeline this algorithm with an approximation algorithm for treewidth, say constant-factor approximation algorithm running in time $2^{\mathcal{O}(\mathrm{tw}(G))}n^{\mathcal{O}(1)}$ [38] where $\mathrm{tw}(G)$ denotes the treewidth of G, obtaining an exact algorithm running in time $2^{\mathcal{O}(\mathrm{td}(G)\mathrm{tw}(G))}n^{\mathcal{O}(1)}$. This is the state-of-the-art as far as parameterized exact algorithms for treedepth (in theory) are concerned.

For practical approaches to computing treedepth exactly, we mention a work by Ganian, Lodha, Ordyniak, and Szeider [18] that experimented with encoding computing treedepth as SAT instances and using SAT solvers.

For approximation, a folklore observation (see [26] for a full proof) is that given a graph G and a tree decomposition of G of width t with tree T, one can in polynomial time find a treedepth decomposition of G of depth at most $(t+1)\mathrm{td}(T)$. Since every tree decomposition of G can be simplified to use $\mathcal{O}(|V(G)|)$ nodes in its tree and every tree T has treedepth $\mathrm{td}(T) \leq \log_2(|V(T)|+1)$ (which is easy to deduce from the recursive formula mentioned earlier), we obtain

$$\mathrm{td}(G) \leq (\mathrm{tw}(G)+1)\cdot\mathcal{O}(\log_2(|V(G)|).$$

Combining the above with known treewidth approximation algorithms, one can obtain a polynomial-time $\mathcal{O}(\mathrm{tw}(G)^2 \log \mathrm{tw}(G))$-approximation for treedepth. The study of forbidden structures characterizations for treedepth led to an improved approximation guarantee of $\mathcal{O}(\mathrm{tw}(G)\log^{3/2}\mathrm{tw}(G))$ [9, 26]. Obtaining a constant-factor approximation for treedepth running in time say $2^{\mathcal{O}(\mathrm{td}(G))}n^{\mathcal{O}(1)}$ remains a challenging open problem.

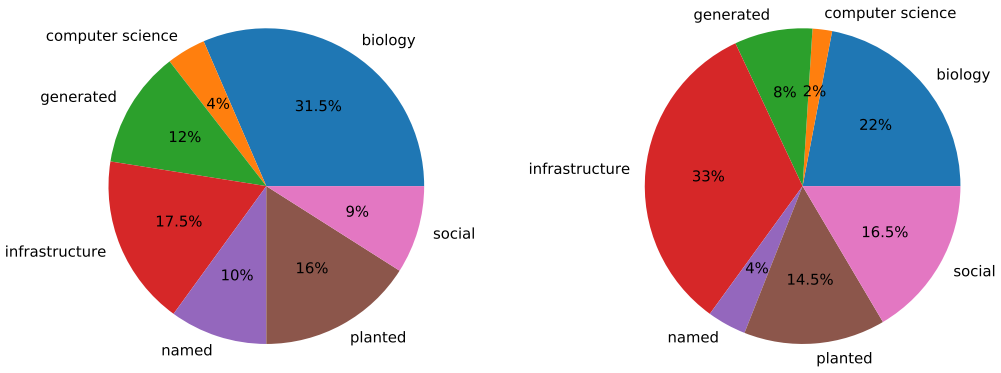
3 The challenge setup

For each track, the PC selected 200 instances. The instances in each track were ordered lexicographically by non-decreasing (n, m) where n is the number of vertices and m is the number of edges. The odd-numbered instances were known to the participants five months before the submission deadline. The even-numbered instances were used to create the official ranking and they were secret until the results of the challenge were announced.

Both in the testing phase and for the final evaluation, the implementations were run for 30 minutes per instance using the `optil.io` on-line judge system [57]. For each instance, the available memory was limited to 8 GB.

In the exact track, the contestants were ranked by the number of instances solved and the total time required for the solved instances as a tiebreaker. Submissions for the exact track were supposed to be based on a provably optimal algorithm, although it was not a formal requirement. Instead, if a submission halted on some instance within the allotted time and output a solution that was worse than the best-known solution (from the PC's solver, other participants, or the way in which the instance was generated) then the submission was disqualified.

In the heuristic track, the goal was to maximize the total score and the score for a single instance for an implementation which returned a result of value d was determined by the formula $100 \cdot \min / d$, where min is the best (though not necessarily optimal) value obtained by any participating team. The reasons for selecting the formula were a) it does not award minor (say, additive) improvements too much (as compared to, e.g., ranking-based methods) and b) the score is within $(0, 100]$ always, so one very bad result does not make the submission to lose (as it could happen, say, for the inverse $d/\min$).



Exact Track Heuristic Track

Figure 2 Distribution of origin categories of the test instances (both public and private).

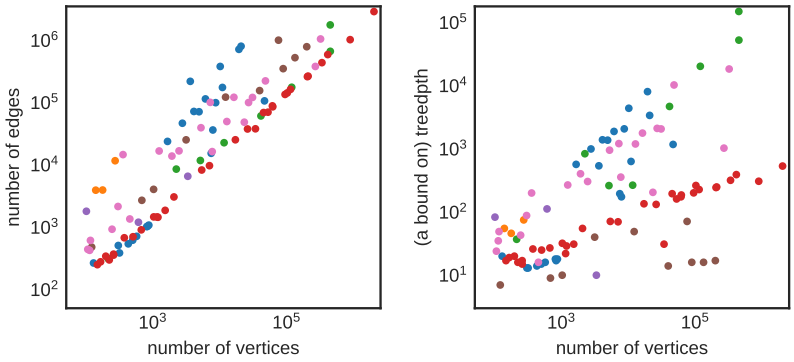


Figure 3 Distribution of origin categories, sizes, densities, and (a bound on) treedepth of the private test instances in the heuristic track. Colors correspond to the origin categories as in Figure 2.

Similarly as for previous editions of PACE, we required that the source code must be published in a public repository and available under an open source license. Following PACE 2019 we also allowed for external dependencies such as ILP, SAT, and treewidth solvers, provided that they were also open source.

4 Selection of the instances

All public and private instances used for PACE 2020 are available at a public repository at the address `https://github.com/lkowalik/Treedepth-PACE-2020-instances`. The set of collected instances contains graphs coming from various applications and graphs generated using a few generators. They can be divided into the following categories (see also Figure 2 for the distribution):

- **biology:** Graphs coming from applications in biology, biochemistry, and medicine. Downloaded from BioGRID [42], SNAP [63], ginsim.org [30], KEGG [24], STRING [48], and network repository [34].
- **computer science:** Control-flow graphs of C functions and graphs originating from register allocation for variables in real codes, created for DIMACS Coloring Challenge 1992–1993.

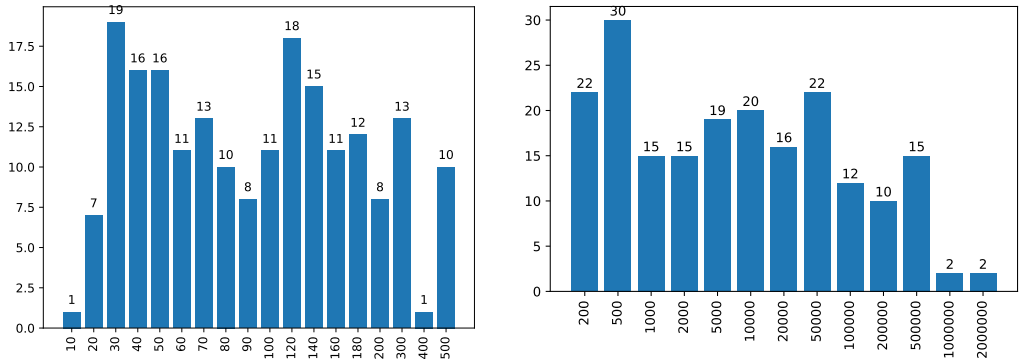


Figure 4 Distribution of sizes in the instance sets.

- **generated:** Graphs obtained from Python's networkx generators: expanders, grids, random cubic graphs, Waxman graphs (random geometric graphs).
- **infrastructure:** Mostly road graphs obtained from open street maps; also power grid networks (from network repository [34]) and public transport graphs contributed by Johannes Fichte for PACE 2016.
- **named:** Small named graphs like the Petersen graph, the flower snark, etc., originating from SageMath.
- **planted:** Random trees and cycles of cliques polluted with random edges that go bottom-up in the optimal treedepth decomposition. These instances were needed for testing correctness of treedepth solvers, because the generator was able to compute the optimum treedepth in polynomial time.
- **social:** Social networks originating from interactions between people, animals, or fictional characters.

Figures 4, 3, and 5 show the distribution of instance sizes and other characteristics depending on track.

5 Participants

There were 15 and 10 teams that officially submitted a solution to the exact and heuristic track, respectively. Five teams participated in both tracks, which gives 20 distinct teams. However, there were 38 more `optil.io` users that submitted a solution to the server during the testing phase (but none of them would be ranked in the top five solves for any track). The 20 teams represented three continents and 12 countries (see Table 1).

6 Exact Track

The results of the Exact Track are as follows.

1. James Trimble (University of Glasgow) solved 78 instances in 6502.97 seconds `github.com/jamestrimble/pace2020-treedepth-solvers` [54, 55]
2. Tuukka Korhonen (University of Helsinki) solved 77 instances in 5599.64 seconds `github.com/Laakeri/pace2020-treedepth-exact` [28, 29]

Table 1 A sorted table of the 20 participating teams' countries.

Country	Number of teams
Germany	4
Netherlands	4
Japan	2
United Kingdom	2
Brazil	1
Finland	1
France	1
India	1
Kosovo	1
Poland	1
Russia	1
Ukraine	1

3. Ruben Brokkelkamp, Mees de Vries, Raymond van Venetië, Jan Westerdiep (Centrum Wiskunde & Informatica (CWI), University of Amsterdam, Korteweg-de Vries Institute for Mathematics, University of Amsterdam) solved 72 instances in 3149.56 seconds
`github.com/mjdv/tdULL` [8, 11]
4. Max Bannach, Sebastian Berndt, Martin Schuster, Marcel Wienöbst (Institute for Theoretical Computer Science at Universität zu Lübeck, Institute for IT Security at Universität zu Lübeck, Institut für Epidemiologie at Universität Kiel) solved 72 instances in 4267.56 seconds
`github.com/maxbannach/PID-Star` [2, 4]
5. Dejun Mao, Vorapong Suppakitpaisarn, Zijian Xu (The University of Tokyo) solved 68 instances in 8794.42 seconds
`github.com/xuzijian629/pace2020` [60, 61]
6. Narek Bojikian, Alexander van der Grinten, Falko Hegerfeld, Laurence Alec Kluge, Stefan Kratsch (Humboldt-Universität zu Berlin) solved 64 instances in 4514.95 seconds
`github.com/PACE-Challenge-Hu-Berlin/PACE-Challenge-2020` [6]
7. Tom van der Zanden (Maastricht University) solved 44 instances in 6304.91 seconds
`github.com/TomvdZanden/BasicTreedepthSolver`
8. Dmitry Sayutin (ITMO University) solved 37 instances in 11465.50 seconds
`github.com/cdkrot/pace2020-sat-dp-solver` [39]
9. Philip de Bruin, Erik Jan van Leeuwen (Utrecht University) solved 27 instances in 4470.34 seconds
`github.com/PhiliPdB/treedepth-exact` [10]
10. Jun Kawahara, Toshiki Saitoh, Akira Suzuki, Toshiyuki Takase, Katsuhisa Yamanaka (Kyoto University, Kyushu Institute of Technology, Tohoku University, Iwate University) solved 6 instances in 198.43 seconds
`github.com/toshimaru0123/pace-2020/` [25]
11. Blend Arifaj, Ardit Baloku, Blend Berisha, Edon Gashi, Endrit Mëziu, Kadri Sylejmani (University of Prishtina) solved 0 instances in 0.00 seconds
`github.com/ksylejmani/treedepth-iterated-local-search`

The following teams submitted a solver, but, as described in the rules, it was disqualified because of at least one suboptimal solution. The number of solved instances reported below refers to the instances for which neither the PC nor any of the submitted solvers were able to produce a better solution.

- Miguel Bosch Calvo, Giorgia Carranza Tejada, Dominik Jeurissen, Steven Kelk, Zhuoer Ma, Alexander Reisach, Borislav Slavchev (Maastricht University) solved 79 instances in 138250.66 seconds
  `github.com/CommanderCero/Treedepth-Pace-2020`
- Sylwester Swat (Poznań University Of Technology) solved 74 instances in 128578.34 seconds
  `github.com/swacisko/pace-2020` [46]
- Marcelo Garlet Milani (Technische Universität Berlin) solved 40 instances in 1469.06 seconds
  `gitlab.tu-berlin.de/mgmillani1/treedepth-pace20` [19]
- Oleg Evseev, Igor Kozin, Alexander Zemlyanskiy (Zaporizhzhya National University) solved 8 instances in 241.10 seconds
  `github.com/oevseev97/pace-2020` [17]

6.1 Details of the solvers

The participants in the exact track used approaches that broadly fell into two categories: bottom-up and top-down.

Bottom-up approaches

In the bottom-up approach, the participants tried to find elimination trees for depths $k = 1, 2, 3, \ldots$ until they succeeded.

The bottom-up approach is to build minimum-depth elimination trees for induced subgraphs of the input graph iteratively from smaller depths to larger depths. Herein, a data structure keeps track of all the vertex sets S for which an elimination tree of $G[S]$ has been computed already. Then, in iterations over the data structure it is tested for which of the subgraphs in the data structure their elimination trees can be combined into an elimination tree of appropriate depth for the union of the subgraphs.

The bottom-up approach is akin to the positive-instance driven approach to dynamic programming [52]. Therein the goal is to avoid unnecessary work that is done in straightforward dynamic programs by carrying the dynamic program out in a forward-looking way. In the usual backward-looking approach we define a signature for subsolutions and we iterate over all signatures that are possible and, for each of them, check whether it is realized by some subsolution, by looking at signatures that are smaller in some well-defined sense and have been computed earlier. In the positive-instance driven way, instead, we directly generate from all the subsolutions that have been generated so far subsolutions which are larger in a well-defined sense. This intuitively avoids checking many signatures if there are only a few possible subsolutions.

The bottom-up approach was used by Trimble (1st place), Bannach et al. (4), Bojikian et al. (6), and van der Zanden (7). All the teams used a number of tricks to speed-up the basic approach and it seems that the winner collected most of them. Perhaps most obviously, many teams used known preprocessing rules [18, 27]. Both Trimble and Bannach et al. observed that the algorithm spends more and more time as k (the upper bound on the depth) grows, simply because then there are more partial solutions to consider. Both

teams came up with the following remedy. First, they run a fast heuristic that computes a treedepth decomposition of some depth t. Then the original exact algorithm follows, with $k = 1, \ldots, t-1$. The hope is that the *optimum* depth is actually t and then we save on the (dominating) time needed for the last iteration. Other speed-ups involved pruning some subsolutions to consider, for example using a domination rule of Ganian et al. [18, Lemma 4.1], or the observation that for a subsolution X all neighbors $N(X)$ must be ancestors in the final elimination tree, so we can discard it when its treedepth plus $|N(X)|$ exceeds the target bound k.

Top-down approaches

In the top-down approach, we try to build an optimal elimination tree from the root to the leaves. In this approach, it is useful to observe that, barring the trivial case of cliques, the top vertices of the elimination tree can be chosen to be an inclusion-wise minimal separator of the input graph [14]. Hence, a natural idea is to enumerate all such separators, branch into all possibilities of taking such a separator for the top vertices of the elimination tree, and recurse into doing the same for each remaining connected component.

The top-down approach was taken by Korhonen (2nd place), Brokkelkamp et al. (3), Mao et al. (5), de Bruin and van Leeuwen (9) and Kawahara et al. (10), though the latter two teams did not use the minimal separators.

Generating minimal separators turned out to be a quite time consuming part of the approach, so different teams used different methods to cope with this issue. Brokkelkamp et al. used the standard minimal separators listing algorithm of Berry et al. [5]. Korhonen used an approach by Tamaki [51]. His algorithm solves the decision problem beginning with *high* upper bound on the treedepth k, e.g., $k = n$, and then improves k until possible. For enumerating minimal separators, first a fast heuristic algorithm [51, Section 4.3] is used, which may not find all the separators (but usually does so). It may happen that this already results in an improved upper bound for the treedepth. Otherwise, the algorithm is run for the second time, this time using a slower, but exact algorithm [51, Section 4.2] for the separator enumeration. Finally, Mao et al. used the space-efficient minimal separator listing algorithm of Takata [49]. Moreover, they skip separators of a size larger than the treewidth of the current graph, since they conjecture that this does not change the resulting treedepth. (Note that the correctness of their algorithm relies on this conjecture.)

Most of the top-down solvers used memoization, i.e., storing upper or lower bounds for subproblems to avoid repeated computation.

Another common technique used by all best solvers using the top-down approach is branch-and-bound, i.e., pruning the branching tree by using lower and upper bounds for the treedepth of subproblems. The combined arsenal of lower bound techniques includes memoized lower bounds for subgraphs, finding a long path or cycle, clique minors, and degeneracy. Korhonen used an interesting upper bound technique. It begins by finding a chordal completion of the graph. A fast heuristic upper bound algorithm is run on the resulting graph, in particular, utilizing the fact that chordal graphs have only a linear number of minimal separators.

Surprisingly, at least according to the submitted descriptions, it seems that only Korhonen applies preprocessing. Namely, he compresses induced subtrees connected to the rest of the graph by a single vertex (using the linear time algorithm for trees of Schäffer [40]) and also generalizes the shared neighborhood rule of Kobayashi and Tamaki [27, Lemma 6].

Table 2 Differences between the top five teams in the exact track (columns contain instances).

Name	068	074	084	088	090	094	108	112	120	148	150	174	180	182	186
Trimble	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓	✓	
Korhonen	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓		✓	
Brokkelkamp et al.		✓		✓						✓			✓	✓	✓
Bannach et al.				✓			✓		✓	✓			✓	✓	
Mao et al.		✓									✓				

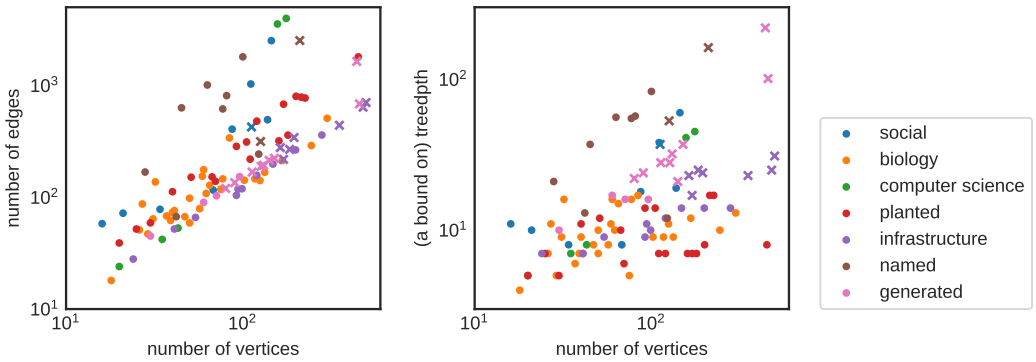


Figure 5 Instances of the exact track: number of edges vs. vertices (left) and (an upper bound on) treedepth vs. vertices (right). Crosses denote instances that were *not* solved by any team.

Other approaches

The solver of Sayutin used the standard $O^*(2^n)$ dynamic programming for small n and the SAT-encoding of Ganian et al. [18]. The solver of Milani implemented the $2^{\mathcal{O}(\mathrm{td}(G)\mathrm{tw}(G))}n^{\mathcal{O}(1)}$ algorithm using dynamic programming over tree decomposition [37] of Reidl et al. The disqualified solvers of Calvo et al. and Swat used heuristic methods (they were written mainly for the heuristic track).

6.1.1 Summary

The top ranked solvers used an impressive number of new and existing ideas. It should be noted that, when properly optimized, both bottom-up and top-down approaches seem to give similar results. Indeed, the solver of Trimble solved just one instance more than the one of Korhonen. This is in strong contrast with exact treewidth computation, where the bottom-up positive-instance driven approach outperforms other known methods (see [13, 52]).

6.2 A closer look at the results of the exact track

Altogether 81 out of the 100 private instances were solved by the participants, which means that the winning team has not solved three instances solved by others. Table 2 shows the differences between the top five teams in the exact track (a common subset of 66 tests was solved by all of them). The participants managed to solve all the tests with less than 80 vertices. The smallest treedepth of an *unsolved* instance was at most 17 (a 170-vertex road network), i.e., we computed an upper bound of 17 by a heuristic solver. The largest treedepth of a *solved* instance was 83 (the 100-vertex Hall-Janko graph). The plots in Figure 5 present solved and unsolved instances divided into categories.

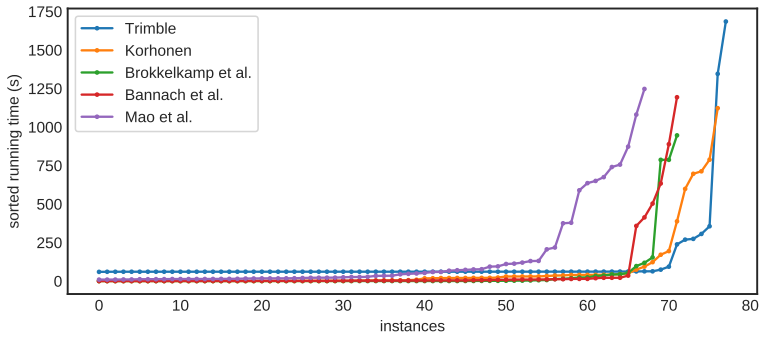


Figure 6 Running times of the five top solvers in the exact track. For each solver, all the instances solved by it were sorted by the running time.

It might be also interesting to look at the *running times* of the solvers. The plot in Figure 6 suggests that the top solvers needed substantial time only for a small fraction of solved instances.

7 Heuristic Track

The results of the Heuristic Track are as follows.

1. Sylwester Swat (Poznań University Of Technology) scored 9710.94 points
   `github.com/swacisko/pace-2020` [46, 47]
2. Ben Strasser scored 9684.10 points
   `github.com/ben-strasser/flow-cutter-pace20` [44, 45]
3. Marcin Wrochna (University of Oxford) scored 9591.21 points
   `github.com/marcinwrochna/sallow` [58, 59]
4. James Trimble (University of Glasgow) scored 9447.96 points
   `github.com/jamestrimble/pace2020-treedepth-solvers` [54, 56]
5. Max Bannach, Sebastian Berndt, Martin Schuster, Marcel Wienöbst (Institute for Theoretical Computer Science at Universität zu Lübeck, Institute for IT Security at Universität zu Lübeck, Institut für Epidemiologie at Universität Kiel) scored 8935.63 points
   `github.com/maxbannach/Fluid` [1, 3]
6. Stéphane Grandcolas (LIS) scored 8880.58 points
   `gitlab.lis-lab.fr/stephane.grandcolas/treedepth-sga/-/tree/master/pace-2020` [22]
7. Miguel Bosch Calvo, Giorgia Carranza Tejada, Dominik Jeurissen, Steven Kelk, Zhuoer Ma, Alexander Reisach, Borislav Slavchev (Maastricht University) scored 6320.19 points
   `github.com/CommanderCero/Treedepth-Pace-2020`
8. Gabriel Duarte, Uéverton Souza, Samuel Silva (Fluminense Federal University) scored 5068.50 points
   `github.com/SamuelEduardoSilva/pace-2020` [15]
9. Aman Singal (Indian Institute of Technology Dharwad) scored 4254.87 points
   `github.com/AmanSingal/pace-2020-submission1` [41]
10. Oleg Evseev, Igor Kozin, Alexander Zemlyanskiy (Zaporizhzhya National University) scored 1071.72 points
    `github.com/oevseev97/pace-2020` [17]

7.1 Details of the solvers

In the heuristic track, similarly as in the exact track, it was not sufficient to come up with a single good idea. All the top solvers used a portfolio of many approaches. This was also forced by the test dataset in which number of vertices varied from 100 to 2 000 000 and treedepth ranged from 7 to up to 150 000. Clearly, in very large graphs the time limit allows only for simple and fast heuristics, while in small and medium instances one can try also more time consuming and possibly more meaningful computation. In contrast to the exact track, here one can try several approaches and output the best result, and this property was frequently used. Again, the most natural classification of methods is bottom-up and top-down processing.

Bottom-up heuristics

The basic scheme of a bottom-up heuristic is to find a so-called *elimination order*, as follows. Pick a vertex v in G (which minimizes some heuristic score), remove v from the graph and connect its remaining neighbors into a clique, obtaining graph G'. Put v in the end of the ordering and find the rest of the ordering recursively. In the resulting treedepth decomposition, the neighbor of v in G' which is the latest in the order becomes v's parent.

In the classic application of the above strategy [21] we always choose a vertex v of minimum degree in the current graph. In the context of treedepth, this is a meaningful measure, because it is a lower bound on the number of ancestors of v. However, one should also take into account the height of v, defined as the maximum of the heights of its eliminated neighbors plus 1, which represents the depth of the subtree rooted at v if it is eliminated at the given moment. Strasser (place 2) and Wrochna (3) used a linear combination of these two values as the vertex score. Wrochna designed also a few increasingly faster and more approximate versions of this approach.

Top-down heuristics

Treedepth can be defined by removing one vertex and recursing to connected components of what remained. However, it is a pretty rare case that by removing just one vertex we get a nice partition into many connected components. If we "unravel" this recursion we may come to a conclusion that it is good to think about removing at once whole *sets* of vertices that are in some way good separators. Ideally, we would like to drive our search of separators by breaking the graph into components with smaller treedepth, but we do not have the desired knowledge about the treedepth of subgraphs, so we need to measure the complexity of subproblems in a different way, for example, by the number of their vertices or edges or some other measure that is easily computed. In theory, approaches driven by an assumption that the treedepth and, say, the number of vertices are closely related can be easily fooled by some hand-crafted instances, but our test dataset was not constructed in such a way, as it contained mostly instances from real-life applications. This motivates an approach where we search for some balanced vertex cuts in our heuristic solver. More precisely, we would like these cuts to be both relatively small and partitioning our graph into components which are significantly smaller than the original graph. This general approach is usually called nested dissection [20].

The participants used various approaches for extracting balanced separators. The most common approach was FlowCutter [23, 43], used in particular by all three top solvers. It uses a maximum flow algorithm to find a minimum cut, and then refines the balance of the

Table 3 Top five solvers. The first column is the number of instances solved *not worse* than any other team. The second column is the number of instances solved *better* than any other team.

Name	The best (with ties)	Single best (no ties)
Swat	51	22
Strasser	47	8
Wrochna	46	7
Trimble	32	1
Bannach et al.	10	0

cut by subsequent flow computations. As a result it identifies a family of cuts with good trade-offs between the size of the cut and its balance. Frequently, the teams used more than one strategy to find separators. For example the winning solver of Swat uses four more ad-hoc separator-finding heuristics, based on articulation points, BFS, and flows. Strasser (2) proposes a local-search heuristic that starts from a cut and improves its size and balance step by step. Bannach et al. (5) use two strategies: an ad-hoc greedy algorithm and a community detection heuristic [36].

The solver of Strasser detects whether the graph passed to the recursive procedure is a clique or a tree and, if so, computes the optimal treedepth (for trees using Schäffer [40]).

Most of the teams just run the whole algorithm from scratch several times, every time using a different balanced separator approach, or different parameters, or a different random seed. However, the winning solver of Swat uses deviating scheme. It always computes many separators, next chooses five of them using a separator scoring function, then attempts to further improve each of the five separators, to finally choose the best of them and recurse.

We also note that Calvo et al. (7), Duarte et al. (8), and Singal (9) used the top-down approach based on removing single vertices (instead of separators), for example, with high centrality measures. However, there was a significant gap between the scores of these solvers and approaches that applied removing separators (1-6), at least as one of the options in their portfolio.

Preprocessing and postprocessing

A bit surprisingly, very few teams used preprocessing. However, Swat (1), Trimble (4), and Grandcolas (6) apply post-processing to see if the resulting tree can be easily improved.

7.2 A closer look at the results of the heuristic track

A closer look at the results obtained by the teams at particular tests shows that there was no single team that dominated the others, in particular each of the top four teams has solved at least one instance *better* than all the others (see Table 3). However, the depths returned by the winning solver of Swat were always within the ratio of 1.13 to the output of any other solver, see Figures 7 and 8. (In Table 3 and Figures 7 and 8 we take into account all the submissions to `optil.io`, including the teams who have not made an official submission to PACE.)

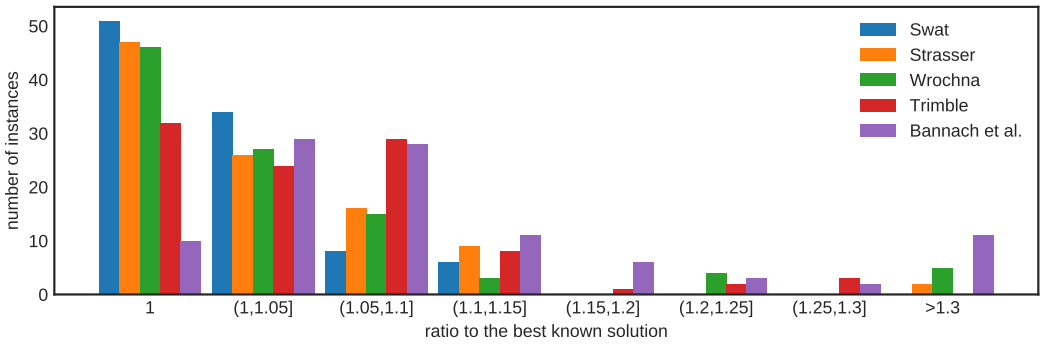


Figure 7 Comparison of the five top solvers in the heuristic track: how many tests they solved within the given ratio to the minimum depth reported by any team.

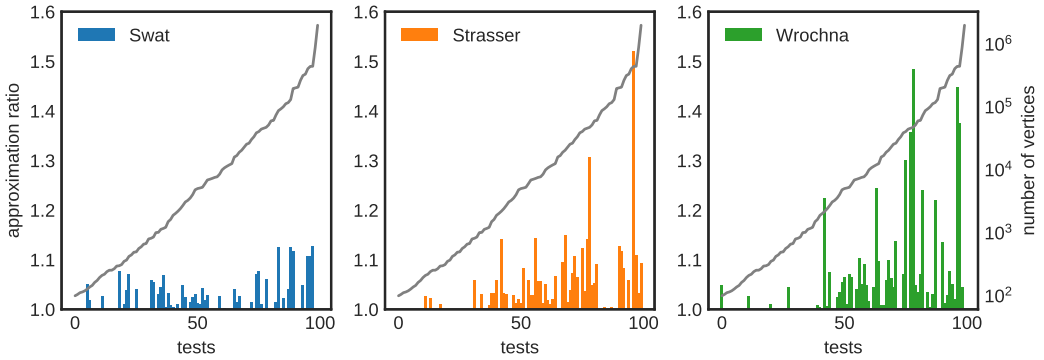


Figure 8 Comparison of the three top solvers in the heuristic track. A bar for team α for test i represents the ratio of the depth of the elimination tree for instance i found by α to the best depth found by of *any other team* for i. In particular no bar means that the team has found the minimum. The gray curve represents size of the tests (in the number of vertices), in the logarithmic scale.

8 PACE organization

The Program Committee of PACE 2020 consisted of Łukasz Kowalik (chair), Marcin Mucha, Wojciech Nadara, Marcin Pilipczuk, Manuel Sorge and Piotr Wygocki, all from University of Warsaw. During the organization of PACE 2020 the Steering Committee was as follows.

Édouard Bonnet	ENS Lyon
Holger Dell	Saarland Informatics Campus
Johannes Fichte	Technische Universität Dresden
Markus Hecher	Technische Universität Wien
Bart M. P. Jansen (chair)	Eindhoven University of Technology
Petteri Kaski	Aalto University
Christian Komusiewicz	Philipps-Universität Marburg
Florian Sikora	Paris-Dauphine University

In October 2020, Łukasz Kowalik, Marcin Pilipczuk, and Manuel Sorge have joined the SC, while Christian Komusiewicz, Florian Sikora, and Petteri Kaski left.

The Program Committee of PACE 2021 will be chaired by André Nichterlein (TU Berlin).

9 Conclusion

We thank all the participants for their impressive work and look forward to the next PACE.

We welcome anyone who is interested to add their name to the mailing list on the website `https://pacechallenge.org/` to receive PACE updates and join the discussion. The updates appear also at Twitter at `https://twitter.com/pace_challenge`. In particular, plans for PACE 2021 will be posted there.

References

1 Max Bannach. maxbannach/fluid: pace-2020, June 2020. doi:10.5281/zenodo.3871709.

2 Max Bannach. maxbannach/pid-star: pace-2020, June 2020. doi:10.5281/zenodo.3871800.

3 Max Bannach, Sebastian Berndt, Martin Schuster, and Marcel Wienöbst. PACE solver description: Fluid. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 27:1–27:3. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.27.

4 Max Bannach, Sebastian Berndt, Martin Schuster, and Marcel Wienöbst. PACE solver description: PID*. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 28:1–28:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.28.

5 Anne Berry, Jean Paul Bordat, and Olivier Cogis. Generating all the minimal separators of a graph. *Int. J. Found. Comput. Sci.*, 11(3):397–403, 2000. doi:10.1142/S0129054100000211.

6 Narek Bojikian, Alexander van der Grinten, Falko Hegerfeld, Laurence Alec Kluge, and Stefan Kratsch. Pace-challenge-2020-hu-berlin-exact-track, June 2020. doi:10.5281/zenodo.3894555.

7 Édouard Bonnet and Florian Sikora. The PACE 2018 Parameterized Algorithms and Computational Experiments Challenge: The Third Iteration. In Christophe Paul and Michal Pilipczuk, editors, *13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.IPEC.2018.26.

8 Ruben Brokkelkamp, Raymond van Venetië, Mees de Vries, and Jan Westerdiep. PACE solver description: tdull. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 29:1–29:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.29.

9 Wojciech Czerwinski, Wojciech Nadara, and Marcin Pilipczuk. Improved bounds for the excluded-minor approximation of treedepth. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 34:1–34:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.34.

10 Philip de Bruin. Philipdb/treedepth-exact: Submission for pace, May 2020. doi:10.5281/zenodo.3866006.

11 Mees de Vries, Ruben Brokkelkamp, Raymond van Venetië, and Jan Westerdiep. tdull, June 2020. doi:10.5281/zenodo.3881472.

12 Holger Dell, Thore Husfeldt, Bart M. P. Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A. Rosamond. The first parameterized algorithms and computational experiments challenge. In Jiong Guo and Danny Hermelin, editors, *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:9, Dagstuhl, Germany, 2017. doi:10.4230/LIPIcs.IPEC.2016.30.

13 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 parameterized algorithms and computational experiments challenge: The second iteration. In *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, pages 30:1–30:12, 2017. doi:10.4230/LIPIcs.IPEC.2017.30.

14 Jitender S. Deogun, Ton Kloks, Dieter Kratsch, and Haiko Müller. On the vertex ranking problem for trapezoid, circular-arc and other graphs. *Discret. Appl. Math.*, 98(1-2):39–63, 1999. doi:10.1016/S0166-218X(99)00179-1.

15 Gabriel Duarte, Samuel Eduardo, and Uéverton Souza. Ufffptteam, June 2020. doi:10.5281/zenodo.3872029.

16 M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher. The PACE 2019 Parameterized Algorithms and Computational Experiments Challenge: The Fourth Iteration (Invited Paper). In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, volume 148 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:23, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.IPEC.2019.25.

17 Oleg Evsees, Alexander Zemlyanskiy, and Igor Kozin. Gensol.exe, June 2020. doi:10.5281/zenodo.3882767.

18 Robert Ganian, Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. Sat-encodings for treecut width and treedepth. In Stephen G. Kobourov and Henning Meyerhenke, editors, *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019, San Diego, CA, USA, January 7-8, 2019*, pages 117–129. SIAM, 2019. doi:10.1137/1.9781611975499.10.

19 Marcelo Garlet Milani. td-milani: A treedepth solver, June 2020. doi:10.5281/zenodo.3885116.

20 Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973. doi:10.1137/0710032.

21 Alan George and Joseph W.H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989. doi:10.1137/1031001.

22 Stéphane Grandcolas. sga, June 2020. doi:10.5281/zenodo.3884054.

23 Michael Hamann and Ben Strasser. Graph bisection with pareto optimization. *ACM J. Exp. Algorithmics*, 23, 2018. doi:10.1145/3173045.

24 Minoru Kanehisa and Susumu Goto. KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic acids research*, 28(1):27–30, 2000.

25 Jun Kawahara, Toshiki Saitoh, Akira Suzuki, Toshiyuki Takase, and Katsuhisa Yamanaka. wankosoba: Exact treedepth decomposition solver, June 2020. doi:10.5281/zenodo.3894127.

26 Ken-ichi Kawarabayashi and Benjamin Rossman. A polynomial excluded-minor approximation of treedepth. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 234–246. SIAM, 2018. doi:10.1137/1.9781611975031.17.

27 Yasuaki Kobayashi and Hisao Tamaki. Treedepth parameterized by vertex cover number. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 18:1–18:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.18.

28 Tuukka Korhonen. Pace 2020 exact treedepth submission: sms, June 2020. doi:10.5281/zenodo.3872898.

29 Tuukka Korhonen. PACE solver description: Sms. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 30:1–30:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.30.

30 A. Naldi, D. Berenguier, A. Fauré, F. Lopez, D. Thieffry, and C. Chaouiya. Logical modelling of regulatory networks with ginsim 2.3. *Biosystems*, 97(2):134–139, 2009. doi:10.1016/j.biosystems.2009.04.008.

31 Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.

32 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

33 Jaroslav Nesetril and Patrice Ossona de Mendez. On low tree-depth decompositions. *Graphs and Combinatorics*, 31(6):1941–1963, 2015. doi:10.1007/s00373-015-1569-7.

34 Network repository. URL: http://networkrepository.com/.

35 Networks project, 2017. URL: http://www.thenetworkcenter.nl.

36 Ferran Parés, Dario Garcia Gasulla, Armand Vilalta, Jonatan Moreno, Eduard Ayguadé, Jesús Labarta, Ulises Cortés, and Toyotaro Suzumura. Fluid communities: A competitive, scalable and diverse community detection algorithm. In Chantal Cherifi, Hocine Cherifi, Márton Karsai, and Mirco Musolesi, editors, *Complex Networks & Their Applications VI*, pages 229–240, Cham, 2018. Springer International Publishing.

37 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014. doi:10.1007/978-3-662-43948-7_77.

38 Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

39 Dmitry Sayutin. PACE 2020 Magnolia Tree Depth solver (DP & Sat based), June 2020. doi:10.5281/zenodo.3888908.

40 Alejandro A. Schäffer. Optimal node ranking of trees in linear time. *Inf. Process. Lett.*, 33(2):91–96, 1989. doi:10.1016/0020-0190(89)90161-0.

41 Aman Singal. Pace2020 - heuristic treedepth, June 2020. doi:10.5281/zenodo.3871918.

42 Chris Stark, Bobby-Joe Breitkreutz, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers. Biogrid: a general repository for interaction datasets. *Nucleic acids research*, 34:D535–D539, 2006.

43 Ben Strasser. Computing tree decompositions with flowcutter: PACE 2017 submission. *CoRR*, abs/1709.08949, 2017. arXiv:1709.08949.

44 Ben Strasser. Flowcutter pace 2020 submission, May 2020. doi:10.5281/zenodo.3870928.

45 Ben Strasser. PACE solver description: Tree depth with flowcutter. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 32:1–32:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.32.

46 Sylwester Swat. swacisko/pace-2020: First release of extreem, June 2020. doi:10.5281/zenodo.3873126.

47 Sylwester Swat. PACE solver description: Finding elimination trees using extreem - a heuristic solver for the treedepth decomposition problem. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 33:1–33:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.33.

48 Damian Szklarczyk, Annika L Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda T Doncheva, John H Morris, Peer Bork, et al. String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic acids research*, 47(D1):D607–D613, 2019.

49 Ken Takata. Space-optimal, backtracking algorithms to list the minimal vertex separators of a graph. *Discret. Appl. Math.*, 158(15):1660–1667, 2010. doi:10.1016/j.dam.2010.05.013.

50 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:13, Dagstuhl, Germany, 2017. doi:10.4230/LIPIcs.ESA.2017.68.

51 Hisao Tamaki. Computing treewidth via exact and heuristic lists of minimal separators. In Ilias S. Kotsireas, Panos M. Pardalos, Konstantinos E. Parsopoulos, Dimitris Souravlias, and Arsenis Tsokas, editors, *Analysis of Experimental Algorithms - Special Event, SEA^2 2019, Kalamata, Greece, June 24-29, 2019, Revised Selected Papers*, volume 11544 of *Lecture Notes in Computer Science*, pages 219–236. Springer, 2019. doi:10.1007/978-3-030-34029-2_15.

52 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *Journal of Combinatorial Optimization*, 37(4):1283–1311, 2019. doi:10.1007/s10878-018-0353-z.

53 James Trimble. An algorithm for the exact treedepth problem. In Simone Faro and Domenico Cantone, editors, *18th International Symposium on Experimental Algorithms, SEA 2020, June 16-18, 2020, Catania, Italy*, volume 160 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SEA.2020.19.

54 James Trimble. jamestrimble/pace2020-treedepth-solvers: v1.0.0, June 2020. doi:10.5281/zenodo.3881441.

55 James Trimble. PACE solver description: Bute-plus: A bottom-up exact solver for treedepth. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 34:1–34:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.34.

56 James Trimble. PACE solver description: Tweed-plus: A subtree-improving heuristic solver for treedepth. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 35:1–35:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.35.

57 Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. Optil.io: Cloud based platform for solving optimization problems using crowdsourcing approach. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*, CSCW '16 Companion, page 433–436, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2818052.2869098.

58 Marcin Wrochna. marcinwrochna/sallow: pace-2020, May 2020. doi:10.5281/zenodo.3870565.

59 Marcin Wrochna. PACE solver description: Sallow: a heuristic algorithm for treedepth decompositions. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 36:1–36:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.36.

60 Zijian Xu, Dejun Mao, and Vorapong Suppakitpaisarn. PACE solver description: Computing exact treedepth via minimal separators. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 31:1–31:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.xx.

61 Zijian Xu, Dejun Mao, and Vorapong Suppakitpaisarn. solver from xuzijian629, May 2020. doi:10.5281/zenodo.3870624.

62 Zijian Xu and Vorapong Suppakitpaisarn. On the size of minimal separators for treedepth decomposition, 2020. arXiv:2008.09822.

63 Marinka Zitnik, Rok Sosič, Sagar Maheshwari, and Jure Leskovec. BioSNAP Datasets: Stanford biomedical network dataset collection. http://snap.stanford.edu/biodata, August 2018.