14th International Symposium on Parameterized and Exact Computation

IPEC 2019, September 11–13, 2019, Munich, Germany

Edited by

Bart M. P. Jansen

Jan Arne Telle

LIPIcs – Vol. 148 – IPEC 2019 www.dagstuhl.de/lipics

Editors

Bart M. P. Jansen
Eindhoven University of Technology, the Netherlands
B.M.P.Jansen@tue.nl

Jan Arne Telle
University of Bergen, Norway
Jan.Arne.Telle@uib.no

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

ISBN 978-3-95977-129-0

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/978-3-95977-129-0.

Publication date
December, 2019

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.2019.0

ISBN 978-3-95977-129-0 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

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

14th International Symposium on Parameterized and Exact Computation (IPEC 2019).
Editors: Bart M. P. Jansen and Jan Arne Telle
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Invited Paper

Preface

This volume contains the papers presented at IPEC 2019: the 14th International Symposium on Parameterized and Exact Computation, which took place September 11–13 in Munich, Germany. IPEC was co-located with five other algorithmic conferences as a part of the annual ALGO congress.

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

In response to the call for papers, 40 abstracts were submitted, which led to 38 submitted papers. Three papers were later withdrawn. Each submission received 3 reviews. The reviews came from the 15 members of the program committee and from 22 external reviewers, together contributing 105 reviews. The program committee held electronic meetings through the EasyChair platform. In the end, the program committee selected 24 of the submissions for presentation at the symposium and inclusion in these proceedings.

The Best Paper Award was given to Giordano Da Lozzo, David Eppstein, Michael Goodrich, and Siddharth Gupta, for their paper *C-Planarity Testing of Embedded Clustered Graphs with Bounded Dual Carving-Width*.

The Best Student Paper Award was given to Gregory Rosenthal for his paper *Beating Treewidth for Average-Case Subgraph Isomorphism*.

IPEC invited one plenary speaker to the ALGO meeting, Raphael Yuster, as part of the award ceremony for the 2019 EATCS-IPEC Nerode Prize for outstanding papers in the area of multivariate algorithmics. The Nerode prize committee consisted of Jianer Chen, Hans L. Bodlaender, and Virginia Vassilevska Williams. They awarded the prize to Noga Alon, Raphael Yuster, and Uri Zwick for their paper *Color-Coding* (Journal of the ACM 42(4): 844–856 (1995)).

IPEC also invited Pasin Manurangsi to present a tutorial on *parameterized inapproximability*. Finally, IPEC hosted the award ceremony and poster session of the fourth *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 M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher on the 2019 PACE challenge.

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. Finally, we are grateful to the local organizers of ALGO, chaired by Susanne Albers, Ernst Bayer, and Gabriele Doblander, for their work on the local arrangements.

Bart M. P. Jansen and Jan Arne Telle
Eindhoven and Bergen, October 2019

14th International Symposium on Parameterized and Exact Computation (IPEC 2019).
Editors: Bart M. P. Jansen and Jan Arne Telle
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Amir Abboud	IBM Almaden Research Center	United States
Édouard Bonnet	ENS Lyon	France
Jianer Chen	Texas A&M University	United States
Petr Golovach	University of Bergen	Norway
Bart M. P. Jansen (co-chair)	Eindhoven University of Technology	The Netherlands
Sudeshna Kolay	Ben-Gurion University	Israel
Lukasz Kowalik	University of Warsaw	Poland
O-joung Kwon	Incheon National University	South Korea
Daniel Marx	MTA SZTAKI	Hungary
Kitty Meeks	University of Glasgow	United Kingdom
Yota Otachi	Kumamoto University	Japan
Felix Reidl	Birkbeck University of London	United Kingdom
Christian Schulz	University of Vienna	Austria
Manuel Sorge	University of Warsaw	Poland
Jan Arne Telle (co-chair)	University of Bergen	Norway

14th International Symposium on Parameterized and Exact Computation (IPEC 2019).
Editors: Bart M. P. Jansen and Jan Arne Telle
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of External Reviewers

Kazuyuki Amano
Rémy Belmonte
Andreas Björklund
Karl Bringmann
Florent Capelli
Radu Curticapean
Andreas Emil Feldmann
Robert Ganian
Tesshu Hanaka
Thore Husfeldt
Tomasz Kociumaka
Christian Komusiewicz
Marvin Künnemann
Martin Lackner
Michael Lampis
Paloma de Lima
Andrea Lincoln
Alexander Noe
Liat Peterfreund
Marcin Pilipczuk
M.S. Ramanujan
Dimitrios Thilikos

14th International Symposium on Parameterized and Exact Computation (IPEC 2019).
Editors: Bart M. P. Jansen and Jan Arne Telle

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

List of Authors

Benjamin Aram Berendsohn (1)
Institut für Informatik, Freie Universität Berlin, Germany

Stéphane Bessy (2)
Université de Montpellier - CNRS, LIRMM, Montpellier, France

Johannes Blum (4)
University of Konstanz, Germany

Markus Bläser (3)
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Édouard Bonnet (5)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Marin Bougeret (2)
Université de Montpellier - CNRS, LIRMM, Montpellier, France

Marco Bressan (6)
Department of Computer Science, Sapienza University of Rome, Italy

Guilherme C. M. Gomes (19)
Universidade Federal de Minas Gerais, Departamento de Ciência da Computação, Belo Horizonte, Brazil; LIRMM, Université de Montpellier, Montpellier, France

Lydia Mirabel Mendoza Cadena (17)
Eötvös Loránd University, Budapest, Hungary

Alan D. A. Carneiro (2)
Universidade Federal Fluminense - Instituto de Computação, Niterói, Brazil

Rajesh Chitnis (7, 8)
School of Computer Science, University of Birmingham, UK

Graham Cormode (7)
University of Warwick, UK

Giordano Da Lozzo (9)
Roma Tre University, Rome, Italy

Jan Dreier (10, 11)
Dept. of Computer Science, RWTH Aachen University, Germany

Gabriel L. Duarte (12)
Fluminense Federal University, Rio de Janeiro, Brazil

M. Ayaz Dzulfikar (25)
University of Indonesia, Kota Depok, Jawa Barat 16424, Indonesia

Christian Engels (3)
IIT Bombay, Mumbai, India

David Eppstein (9)
University of California, Irvine, USA

Hiroshi Eto (13)
Kyushu University, Fukuoka, Japan

Andreas Emil Feldmann (8)
Charles University, Czechia

Johannes K. Fichte (25)
Faculty of Computer Science, TU Dresden, 01062 Dresden, Germany

Till Fluschnik (14)
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany

Florent Foucaud (15)
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans Cedex 2, France; Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France

Janosch Fuchs (10)
Dept. of Computer Science, RWTH Aachen University, Germany

Jiawei Gao (16)
University of California, San Diego, CA, USA

Petr A. Golovach (18)
Department of Informatics, University of Bergen, Norway

Michael T. Goodrich (9)
University of California, Irvine, USA

Siddharth Gupta (9)
Ben-Gurion University of the Negev, Beersheba, Israel

Alexander Göke (17)
Universität Bonn, Bonn, Germany; Technische Universität Hamburg, Hamburg, Germany

Thekla Hamm (20)
Algorithms and Complexity Group, TU Wien, Vienna, Austria

Tesshu Hanaka (13)
Chuo University, Tokyo, Japan

14th International Symposium on Parameterized and Exact Computation (IPEC 2019).
Editors: Bart M. P. Jansen and Jan Arne Telle
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Tim A. Hartmann (10)
Dept. of Computer Science, RWTH Aachen University, Germany

Markus Hecher (25)
Institute of Logic and Computation, TU Wien, Favoritenstraße 9-11, 1040 Wien, Austria; University of Potsdam, Germany

Hervé Hocquard (15)
Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France

Edin Husić (21)
Department of Mathematics, LSE, Houghton Street, London, WC2A 2AE, United Kingdom

Yoichi Iwata (22)
National Institute of Informatics, Tokyo, Japan

Yasuaki Kobayashi (13)
Kyoto University, Kyoto, Japan

Yusuke Kobayashi (13, 22)
Kyoto University, Kyoto, Japan

László Kozma (1)
Institut für Informatik, Freie Universität Berlin, Germany

Philipp Kuinke (10)
Dept. of Computer Science, RWTH Aachen University, Germany

Dimitri Lajou (15)
Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France

Daniel Lokshtanov (12)
University of California Santa Barbara, CA, USA

Dániel Marx (1)
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Valia Mitsou (15)
Université Paris-Diderot, IRIF, CNRS, 75205, Paris, France

Matthias Mnich (17)
Universität Bonn, Bonn, Germany; Technische Universität Hamburg, Hamburg, Germany

Rolf Niedermeier (14)
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany

Jana Novotná (23)
Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Karolina Okrasa (23)
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland

Lehilton L. C. Pedrosa (12)
University of Campinas, São Paulo, Brazil

Théo Pierron (15)
Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France; DIMEA, Masaryk University, 60200 Brno, Czech republic

Michał Pilipczuk (23)
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Fábio Protti (2)
Universidade Federal Fluminense - Instituto de Computação, Niterói, Brazil

Nidhi Purohit (5)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Valentin Rohm (14)
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany

Gregory Rosenthal (24)
University of Toronto, Canada

Peter Rossmanith (10, 11)
Dept. of Computer Science, RWTH Aachen University, Germany

Paweł Rzążewski (23)
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland

Ignasi Sau (19)
CNRS, LIRMM, Université de Montpellier, Montpellier, France

Rafael C. S. Schouery (12)
University of Campinas, São Paulo, Brazil

Uéverton S. Souza (2, 12)
Universidade Federal Fluminense - Instituto de Computação, Niterói, Brazil

Bjoern Tauer (10)
Dept. of Computer Science, RWTH Aachen University, Germany

Dimitrios M. Thilikos (18)
AlGCo project-team, LIRMM, Université de Montpellier, CNRS, Montpellier, France

Stéphan Thomassé (21)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France; Institut Universitaire de France, Paris, France

Nicolas Trotignon (21)
Univ Lyon, ENS de Lyon, Université Claude Bernard Lyon 1, CNRS, LIP, F-69342, Lyon Cedex 07, France

Erik Jan van Leeuwen (23)
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Bartosz Walczak (23)
Department of Theoretical Computer Science, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

Hung-Lung Wang (10)
Computer Science and Information Engineering, National Taiwan Normal University, Taiwan

Philipp Zschoche (14)
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany

Finding and Counting Permutations via CSPs

Benjamin Aram Berendsohn
Institut für Informatik, Freie Universität Berlin, Germany
beab@zedat.fu-berlin.de

László Kozma
Institut für Informatik, Freie Universität Berlin, Germany
laszlo.kozma@fu-berlin.de

Dániel Marx
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
dmarx@mpi-inf.mpg.de

Abstract

Permutation patterns and pattern avoidance have been intensively studied in combinatorics and computer science, going back at least to the seminal work of Knuth on stack-sorting (1968). Perhaps the most natural algorithmic question in this area is deciding whether a given permutation of length n contains a given pattern of length k.

In this work we give two new algorithms for this well-studied problem, one whose running time is $n^{k/4+o(k)}$, and a polynomial-space algorithm whose running time is the better of $O(1.6181^n)$ and $O(n^{k/2+1})$. These results improve the earlier best bounds of $n^{0.47k+o(k)}$ and $O(1.79^n)$ due to Ahal and Rabinovich (2000) resp. Bruner and Lackner (2012) and are the fastest algorithms for the problem when $k \in \Omega(\log n)$. We show that both our new algorithms and the previous exponential-time algorithms in the literature can be viewed through the unifying lens of *constraint-satisfaction*.

Our algorithms can also *count*, within the same running time, the number of occurrences of a pattern. We show that this result is close to optimal: solving the counting problem in time $f(k) \cdot n^{o(k/\log k)}$ would contradict the *exponential-time hypothesis* (ETH). For some special classes of patterns we obtain improved running times. We further prove that 3-*increasing* and 3-*decreasing* permutations can, in some sense, *embed* arbitrary permutations of almost linear length, which indicates that an algorithm with sub-exponential running time is unlikely, even for patterns from these restricted classes.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Pattern matching

Keywords and phrases permutations, pattern matching, constraint satisfaction, exponential time

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.1

Related Version A full version of the paper is available at `https://arxiv.org/abs/1908.04673`.

Funding *Dániel Marx*: Supported by the European Research Council (ERC) Consolidator Grant No. 725978 SYSTEMATICGRAPH.

Acknowledgements An earlier version of the paper contained a mistake in the analysis of the algorithm for Theorem 2. We thank Günter Rote for pointing out the error.
This work was prompted by the Dagstuhl Seminar 18451 "Genomics, Pattern Avoidance, and Statistical Mechanics". The second author thanks the organizers for the invitation and the participants for interesting discussions.

1 Introduction

Let $[n] = \{1, \ldots, n\}$. Given two permutations $\tau : [n] \to [n]$, and $\pi : [k] \to [k]$, we say that τ *contains* π, if there are indices $1 \leq i_1 < \cdots < i_k \leq n$ such that $\tau(i_j) < \tau(i_\ell)$ if and only if $\pi(j) < \pi(\ell)$, for all $1 \leq j, \ell \leq k$. In other words, τ contains π, if the sequence $(\tau(1), \ldots, \tau(n))$

has a (possibly non-contiguous) subsequence with the same ordering as $(\pi(1), \ldots, \pi(k))$, otherwise τ *avoids* π. For example, $\tau = (1, 5, 4, 6, 3, 7, 8, 2)$ contains $(2, 3, 1)$, because its subsequence $(5, 6, 3)$ has the same ordering as $(2, 3, 1)$; on the other hand, τ avoids $(3, 1, 2)$.

Knuth showed in 1968 [40, § 2.2.1], that permutations sortable by a single stack are exactly those that avoid $(2, 3, 1)$. Sorting by restricted devices has remained an active research topic [53, 46, 48, 12, 3, 5], but permutation pattern avoidance has also taken on a life of its own (especially after the influential work of Simion and Schmidt [50]), becoming an important subfield of combinatorics. For more background on permutation patterns and pattern avoidance we refer to the extensive survey [55] and relevant textbooks [13, 14, 39].

Perhaps the most important enumerative result related to permutation patterns is the theorem of Marcus and Tardos [41] from 2004, stating that the number of length-n permutations that avoid a fixed pattern π is bounded by $c(\pi)^n$, where $c(\pi)$ is a quantity independent of n. (This was conjectured by Stanley and Wilf in the late 1980s.)

A fundamental algorithmic problem in this context is *Permutation Pattern Matching* (PPM): Given a length-n permutation τ ("text") and a length-k permutation π ("pattern"), decide whether τ contains π.

Solving PPM is a bottleneck in experimental work on permutation patterns [4]. The problem and its variants also arise in practical applications, e.g. in computational biology [39, § 2.4] and time-series analysis [38, 10, 45]. Unfortunately PPM is, in general, NP-complete, as shown by Bose, Buss, and Lubiw [15] in 1998. For small (e.g. constant-sized) patterns, the problem is solvable in polynomial (in fact, linear) time, as shown by Guillemot and Marx [33] in 2013. Their algorithm has running time $n \cdot 2^{O(k^2 \log k)}$, establishing the *fixed-parameter tractability* of the PPM problem in terms of the pattern length. The algorithm builds upon the Marcus-Tardos proof of the Stanley-Wilf conjecture and introduces a novel decomposition of permutations. Subsequently, Fox [30] refined the Marcus-Tardos result, thereby removing a factor $\log k$ from the exponent of the Guillemot-Marx bound. (Due to the large constants involved, it is however, not clear whether the algorithm can be efficient in practice.)

For longer patterns, e.g. for $k \in \Omega(\log n)$, the complexity of the PPM problem is less understood. An obvious algorithm with running time $O(n^{k+1})$ is to enumerate all $\binom{n}{k}$ length-k subsequences of τ, checking whether any of them has the same ordering as π. The first result to break this "triviality barrier" was the $O(n^{2k/3+1})$-time algorithm of Albert, Aldred, Atkinson, and Holton [4]. Shortly thereafter, Ahal and Rabinovich [1] obtained the running time $n^{0.47k+o(k)}$.

The two algorithms are based on a similar dynamic programming approach: they embed the entries of the pattern π one-by-one into the text τ, while observing the restrictions imposed by the current partial embedding. The order of embedding (implicitly) defines a *path-decomposition* of a certain graph derived from the pattern π, called the *incidence graph*. The running time obtainable in this framework is of the form $O(n^{\mathrm{pw}(\pi)+1})$, where $\mathrm{pw}(\pi)$ is the *pathwidth* of the incidence graph of π.

Ahal and Rabinovich also describe a different, *tree-based* dynamic programming algorithm that solves PPM in time $O(n^{2 \cdot \mathrm{tw}(\pi)+1})$, where $\mathrm{tw}(\pi)$ is the *treewidth* of the incidence graph of π. Using known bounds on the treewidth, however, this running time does not improve the previous one.

Our first result is based on the observation that PPM can be formulated as a *constraint satisfaction problem* (CSP) with binary constraints. In this view, the path-based dynamic programming of previous works has a natural interpretation not observed earlier: it amounts to solving the CSP instance by *Seidel's invasion algorithm*, a popular heuristic [49],[54, § 9.3].

It is well-known that binary CSP instances can be solved in time $O(n^{t+1})$, where n is the *domain size*, and t is the *treewidth* of the *constraint graph* [32, 24]. In our reduction, the domain size is the length n of the text τ, and the constraint graph is the incidence graph of the pattern π; we thus obtain a running time of $O(n^{\mathrm{tw}(\pi)+1})$, improving upon the earlier $O(n^{2\cdot\mathrm{tw}(\pi)+1})$. Second, making use of a bound known for low-degree graphs [28], we prove that the treewidth of the incidence graph of π is at most $k/3 + o(k)$. The final improvement from $k/3$ to $k/4$ is achieved via a technique inspired by recent work of Cygan, Kowalik, and Socała [23] on the k-OPT heuristic for the *traveling salesman problem* (TSP).

In summary, we obtain the following result, proved in § 3.

▶ **Theorem 1.** *Permutation Pattern Matching can be solved in time $n^{k/4+o(k)}$.*

Expressed in terms of n only, none of the mentioned running times improve, in the worst case, upon the trivial 2^n; consider the case of a pattern of length $k \geq n/\log n$. The first improvement in this parameter range was obtained by Bruner and Lackner [18]; their algorithm runs in time $O(1.79^n)$.

The algorithm of Bruner and Lackner works by decomposing both the text and the pattern into *alternating runs* (consecutive sequences of increasing or decreasing elements), and using this decomposition to restrict the space of admissible matchings. The exponent in the running time is, in fact, the *number of runs* of T, which can be as large as n. The approach is compelling and intuitive, the details, however, are intricate (the description of the algorithm and its analysis in [18] take over 24 pages).

Our second result improves this running time to $O(1.618^n)$, with an exceedingly simple approach. A different analysis of our algorithm yields the bound $O(n^{k/2+1})$, i.e. slightly above the Ahal-Rabinovich bound [1], but with polynomial space. The latter bound also matches an earlier result of Guillemot and Marx [33, § 7], obtained via involved techniques.

▶ **Theorem 2.** *Permutation Pattern Matching can be solved using polynomial space, in time $O(1.6181^n)$ or $O(n^{k/2+1})$.*

At the heart of this algorithm is the following observation: if all *even-index* entries of the pattern π are matched to entries of the text τ, then verifying whether the remaining *odd-index* entries of π can be correctly matched takes only a linear-time sweep through both π and τ. This algorithm can be explained very simply in the CSP framework: after substituting a value to every *even-index* variable, the graph of the remaining constraints is a union of paths, and hence can be handled very easily.

Counting patterns. We also consider the closely related problem of *counting* the number of occurrences of π in τ, i.e. finding the number of subsequences of τ that have the same ordering as π. Easy modifications of our algorithms solve this problem within the bounds of Theorems 1 and 2.

▶ **Theorem 3.** *The number of solutions for Permutation Pattern Matching can be computed*
- *in time $n^{k/4+o(k)}$,*
- *in time $O(n^{k/2+2})$ and polynomial space, and*
- *in time $O(1.6181^n)$ and polynomial space.*

Note that the FPT algorithm of Guillemot and Marx [33] cannot be adapted for the counting version. In fact, we argue (§ 5) that a running time of the form $n^{O(k)}$ is almost best possible and a significant improvement in running time for the counting problem is unlikely.

▶ **Theorem 4.** *Assuming the exponential-time hypothesis (ETH), there is no algorithm that counts the number of occurrences of π in τ in time $f(k) \cdot n^{o(k/\log k)}$, for any function f.*

Special patterns. It is possible that PPM is easier if the pattern π comes from some restricted family of permutations, e.g. if it avoids some smaller fixed pattern σ. Several such examples have been studied in the literature, and recently Jelínek and Kynčl [37] obtained the following characterization: PPM is polynomial-time solvable for σ-avoiding patterns π, if σ is one of (1), $(1,2)$, $(1,3,2)$, $(2,1,3)$ or their reverses, and NP-complete for all other σ. (All tractable cases are such that π is a *separable* permutation [15, 36, 56, 4].)

In particular, Jelínek and Kynčl show that PPM is NP-complete even if π avoids $(1,2,3)$ or $(3,2,1)$, but polynomial-time solvable for any proper subclass of these families. For $(1,2,3)$-avoiding and $(3,2,1)$-avoiding patterns, it is known however, that PPM can be solved in time $n^{O(\sqrt{k})}$, i.e. faster than the general case (Guillemot and Vialette [34]).

These results motivate the following general and natural question.

▶ **Question.** *What makes a permutation pattern easier to find than others?*

A permutation is *t-monotone*, if it can be obtained by interleaving t monotone sequences. When all t sequences are increasing (resp. decreasing), we call the resulting permutation t-increasing (resp. t-decreasing). It is well-known that t-increasing (resp. t-decreasing) permutations are exactly those that avoid $(t+1,\dots,1)$, resp. $(1,\dots,t+1)$, see e.g. [7].

We prove that if π is 2-monotone, then the running time of the algorithm of Theorem 1 is $n^{O(\sqrt{k})}$. This result follows from bounding the treewidth of the incidence graph of π, by observing that this graph is *almost planar*. For 2-increasing or 2-decreasing patterns we thus match the bound of Guillemot and Vialette by a significantly simpler argument. (In these special cases the incidence graph is, in fact, planar.)

Jordan-permutations are a natural family of geometrically-defined permutations with applications in computational geometry [47]. They were studied by Hoffmann, Mehlhorn, Rosenstiehl, and Tarjan [35], who showed that they can be sorted with a linear number of comparisons (see also [2] for related enumerative results). A Jordan permutation is generated by the intersection-pattern of two simple curves in the plane: label the intersection points between the curves in increasing order along the first curve, and read out the labels along the second curve; the obtained sequence is a Jordan-permutation (Figure 1). As the incidence graph of the pattern π is planar whenever π is a Jordan-permutation, in this case too an $n^{O(\sqrt{k})}$ bound on the running time follows.

▶ **Theorem 5.** *The treewidth of the incidence graph of π is $O(\sqrt{k})$,*
(i) if π is 2-monotone, or (ii) if π is a Jordan-permutation.

We show that both 2-monotone (and even 2-increasing or 2-decreasing) and Jordan-permutations of length $O(k)$ may contain grids of size $\sqrt{k} \times \sqrt{k}$ in their incidence-graphs, both statements of Theorem 5 are therefore tight, via known lower bounds on the treewidth of grids [11].

In light of these results, one may try to obtain further treewidth-bounds for families of patterns, in order to solve PPM in sub-exponential time. In this direction we show a (somewhat surprising) negative result.

▶ **Theorem 6.** *There are 3-increasing permutations of length k whose incidence graph has treewidth $\Omega(k/\log k)$.*

The same bound applies, by symmetry, to 3-decreasing permutations. The result is obtained by embedding the incidence graph of an *arbitrary* permutation of length $O(k/\log k)$ as a *minor* of the incidence graph of a 3-increasing permutation of length k.

Theorems 5 and 6 (proved in § 4) lead to an almost complete characterization of the treewidth of σ-avoiding patterns. By the Erdős-Szekeres theorem [27] every k-permutation

contains a monotone pattern of length $\lceil\sqrt{k}\rceil$. Thus, for all permutations σ of length at least 10, the class of σ-avoiding permutations contains all 3-increasing or all 3-decreasing permutations, hence by Theorem 6 there exist σ-avoiding patterns π with $\mathrm{tw}(\pi) \in \Omega(k/\log k)$. Addressing a few additional small cases by similar arguments (details given in the thesis of the first author), the threshold 10 can be further reduced. We remark that no algorithm is known to solve PPM in time $n^{o(\mathrm{tw}(\pi))}$; see the discussion in [1, 37].

With a weaker bound we obtain a full characterisation that strengthens the dichotomy-result of Jelínek and Kynčl: in the worst case, the only σ-avoiding patterns π for which $\mathrm{tw}(\pi) \in o(\sqrt{k})$ are those for which PPM is known to be polynomial-time solvable.

Further related work. The complexity of the PPM problem has also been studied under the stronger restriction that the text τ is pattern-avoiding. The problem is polynomial-time solvable if τ is monotone [21] or 2-monotone [19, 34, 6, 4, 43], but NP-hard if τ is 3-monotone [37]. A broader characterization is missing.

Only *classical patterns* are considered in this paper; variants in the literature include *vincular*, *bivincular*, *consecutive*, and *mesh* patterns; we refer to [17] for a survey of related computational questions.

Newman et al. [44] study pattern matching in a *property-testing* framework (aiming to distinguish pattern-avoiding sequences from those that contain *many copies* of the pattern). In this setting, the focus is on the *query complexity* of different approaches, and sampling techniques are often used; see also [9, 31].

A different line of work investigates whether standard algorithmic problems on permutations (e.g. sorting, selection) become easier if the input can be assumed to be pattern-avoiding [8, 20].

2 Preliminaries

A length-n permutation σ is a bijective function $\sigma : [n] \to [n]$, alternatively viewed as the sequence $(\sigma(1), \ldots, \sigma(n))$. Given a length-$n$ permutation σ, we denote as $S_\sigma = \{(i, \sigma(i)) \mid 1 \le i \le n\}$ the *set of points* corresponding to permutation σ.

For a point $p \in S_\sigma$ we denote its first entry as $p.x$, and its second entry as $p.y$, referring to these values as the *index*, respectively, the *value* of p. Observe that for every $i \in [n]$, we have $|\{p \in S_\sigma \mid p.x = i\}| = |\{p \in S_\sigma \mid p.y = i\}| = 1$.

We define four neighbors of a point $(x, y) \in S_\sigma$ as follows.

$$\begin{aligned} N^R((x,y)) &= (x+1,\ \sigma(x+1)),\\ N^L((x,y)) &= (x-1,\ \sigma(x-1)),\\ N^U((x,y)) &= (\sigma^{-1}(y+1),\ y+1),\\ N^D((x,y)) &= (\sigma^{-1}(y-1),\ y-1). \end{aligned}$$

The superscripts R, L, U, D are meant to evoke the directions *right*, *left*, *up*, *down*, when plotting S_σ in the plane. Some neighbors of a point may coincide. When some index is out of bounds, we let the offending neighbor be a "virtual point" as follows: $N^R(n, i) = N^U(i, n) = (\infty, \infty)$, and $N^L(1, i) = N^D(i, 1) = (0, 0)$, for all $i \in [n]$. The virtual points are not contained in S_σ, we only define them to simplify some of the statements.

The *incidence graph* of a permutation σ is $G_\sigma = (S_\sigma, E_\sigma)$, where

$$E_\sigma = \{(p, N^\alpha(p)) \mid \alpha \in \{R, L, U, D\} \text{ and } p, N^\alpha(p) \in S_\sigma\}.$$

In words, each point is connected to its (at most) four neighbors: its successor and predecessor by index, and its successor and predecessor by value. It is easy to see that G_σ is a union of two Hamiltonian paths on the same set of vertices and that this is an exact characterization of permutation incidence-graphs. (See Figure 1 for an illustration.)

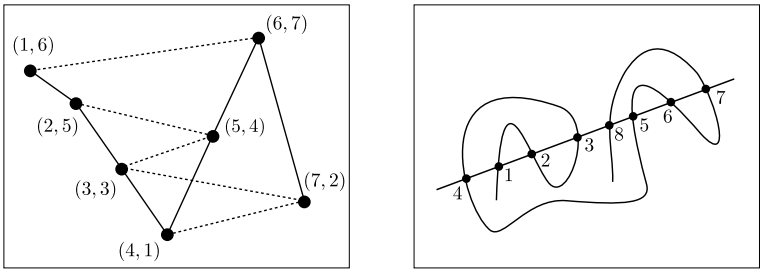


Figure 1 (*left*) Permutation $\pi = (6, 5, 3, 1, 4, 7, 2)$ and its incidence graph G_π. Solid lines indicate neighbors by index, dashed lines indicate neighbors by value (lines may overlap). Indices plotted on x-coordinate, values plotted on y-coordinate. (*right*) Jordan-permutation $(4, 1, 2, 3, 8, 5, 6, 7)$.

Throughout the paper we consider a text permutation $\tau : [n] \to [n]$, and a pattern permutation $\pi : [k] \to [k]$, where $n \geq k$. We give an alternative definition of the Permutation Pattern Matching (PPM) problem in terms of embedding S_π into S_τ.

Consider a function $f : S_\pi \to S_\tau$. We say that f is a *valid embedding* of S_π into S_τ if for all $p \in S_\pi$ the following hold:

$$f(N^L(p)).x \quad < \quad f(p).x \quad < \quad f(N^R(p)).x, \quad \text{and} \tag{1}$$

$$f(N^D(p)).y \quad < \quad f(p).y \quad < \quad f(N^U(p)).y, \tag{2}$$

whenever the corresponding neighbor $N^\alpha(p)$ is also in S_π, i.e. not a virtual point. In words, valid embeddings preserve the relative positions of neighbors in the incidence graph.

▶ **Lemma 7.** *Permutation τ contains permutation π if and only if there exists a valid embedding $f : S_\pi \to S_\tau$.*

For sets $A \subseteq B \subseteq S_\pi$ and functions $g : A \to S_\tau$ and $f : B \to S_\tau$ we say that g is the *restriction* of f to A, denoted $g = f|_A$, if $g(i) = f(i)$ for all $i \in A$. In this case, we also say that f is the *extension* of g to B. Restrictions of valid embeddings will be called *partial embeddings*. We observe that if $f : B \to S_\tau$ is a partial embedding, then it satisfies conditions (1) and (2) with respect to all edges in the induced graph $G_\pi[B]$, i.e. the corresponding inequality holds whenever $p, N^\alpha(p) \in B$.

3 Pattern matching as constraint satisfaction

Readers familiar with the terminology of CSPs should immediately recognize that the definition of valid embedding and Lemma 7 allow us to formulate PPM as a CSP instance with binary constraints. Then known techniques can be applied to solve the problem. A (somewhat different) connection of PPM to CSPs was previously observed by Guillemot and Marx [33]. We first review briefly the CSP problem, referring to [54, 49, 22] for more background.

A *binary CSP* instance is a triplet (V, D, C), where V is a set of variables, D is a set of admissible values (the *domain*), and C is a set of constraints $C = \{c_1, \ldots, c_m\}$, where each constraint c_i is of the form $((x, y), R)$, where $x, y \in V$, and $R \subseteq D^2$ is a binary relation.

A solution of the CSP instance is a function $f : V \to D$ (i.e. an assignment of admissible values to the variables), such that for each constraint $c_i = ((x_i, y_i), R_i)$, the pair of assigned values $(f(x_i), f(y_i))$ is contained in R_i.

The reduction from PPM to CSP is straightforward. Given a PPM instance with text τ and pattern π, of lengths n and k respectively, let $V = \{x_1, \ldots, x_k\}$, and $D = \{1, \ldots, n\}$. The fact that variable x_i takes value j signifies that $\pi(i)$ is matched (embedded) to $\tau(j)$. For the embedding to be valid, by Lemma 7, the relative ordering of entries must be respected, in accordance with conditions (1) and (2). These conditions can readily be described by binary relations for all pairs of variables whose corresponding entries are neighbors in the incidence graph G_π.

More precisely, for $p, N^\alpha(p) \in S_\pi$, for $\alpha \in \{R, L, U, D\}$, we add constraints of the form $((x_i, x_j), R)$, where $i = p.x$, $j = N^\alpha(p).x$ and R contains those pairs $(a, b) \in [n]^2$, for which the relative position of $(a, \tau(a))$ and $(b, \tau(b))$ matches the relative position of p and $N^\alpha(p)$.

The *constraint graph* of the binary CSP instance (also known as *primal graph* or *Gaifman graph*) is a graph whose vertices are the variables V and whose edges connect all pairs of variables that occur together in a constraint. Observe that for instances obtained via our reduction, the constraint graph is exactly the incidence graph G_π. We make use of the following well-known result.

▶ **Lemma 8** ([32, 24]). *A binary CSP instance (V, D, C) can be solved in time $O(|D|^{t+1})$ where t is the treewidth of the constraint graph.*

As discussed in § 2, the incidence graph G_π consists of two Hamiltonian-paths. Accordingly, its vertices have degree at most 4, and the following structural result is applicable.

▶ **Lemma 9** ([28, 29]). *If G is an order-k graph with vertices of degree at most 4, then the pathwidth (and consequently, the treewidth) of G is at most $k/3 + o(k)$. A corresponding tree-(path-)decomposition can be found in polynomial time.*

Algorithms. Our first algorithm amounts to reducing the PPM instance to a binary CSP instance, and using the algorithm of Lemma 8 with a tree-decomposition obtained via Lemma 9. To reach the bound given in Theorem 1, it remains to improve the $k/3$ term in the exponent to $k/4$. We achieve this with a recent technique of Cygan et al. [23], developed in the context of the k-OPT heuristic for TSP.

In our setting, the technique works as follows. We split $[n]$ into $n^{1/4}$ contiguous intervals of equal widths, $n^{3/4}$ each. (For simplicity, we ignore issues of rounding and divisibility.) The intervals induce vertical *strips* in the text τ. For each pattern-index $i \in [k]$ we *guess* the vertical strip of τ into which i is mapped in the sought-for embedding of π into τ. It is sufficient to do this for a subset of the entries in π, namely those that become the *leftmost* in their respective strips in τ. Let $X \subseteq [k]$ be the set of indices of such entries in π.

Guessing X and the strips of τ into which entries of X are mapped increases the running time by a factor of $\sum_{X \subseteq [k]} \binom{n^{1/4}}{|X|} \leq \sum_{X \subseteq [k]} n^{|X|/4}$. Assuming that we guessed correctly, the problem simplifies. First, each pattern-entry can now be embedded into at most $n^{3/4}$ possible locations, hence the domain of each variable will be of size at most $n^{3/4}$. Second, the horizontal constraints that go across strip-boundaries can now be removed as they are implicitly enforced by the distribution of entries into strips (the L-constraint of every X-entry is removed). We have thus reduced the number of edges in the constraint-graph by $|X| - 1$ and can use a stronger upper bound of $(k - |X|)/3 + o(k)$ on the treewidth (see e.g. [28, 23]).

The overall running time becomes

$$\sum_{X \subseteq [k]} n^{\frac{|X|}{4}} \cdot n^{\frac{3}{4} \cdot (\frac{k}{3} - \frac{|X|}{3}) + o(k)} = 2^k \cdot n^{k/4 + o(k)} = n^{k/4 + o(k)}.$$

We remark that our use of this technique is essentially the same as in Cygan et al. [23], but the CSP-formalism makes its application more transparent. We suspect that further classes of CSPs could be handled with a similar approach.

The even-odd method. The algorithm for Theorem 2 can be obtained as follows. Let (Q^E, Q^O) be the partition of S_π into points with even and odd indices. Formally, $Q^E = \{(2k, \pi(2k)) \mid 1 \leq k \leq \lfloor k/2 \rfloor\}$, and $Q^O = \{(2k-1, \pi(2k-1)) \mid 1 \leq k \leq \lceil k/2 \rceil\}$. Construct the CSP instance corresponding to the problem as above. A solution is now found by trying first every possible combination of values for the variables representing Q^E. Clearly, there are $n^{|Q^E|} = n^{\lfloor k/2 \rfloor}$ possible combinations. If the value of a variable x_i is fixed to $a \in [n]$, then x_i is removed from the problem and every neighbor of x_i is restricted by a new unary constraint in an appropriate way, i.e. if there is a constraint $((x_i, x_j), R)$, then x_j should be restricted to values b for which $(a, b) \in R$.

How does the constraint graph look like if we remove every variable (and its incident edges) corresponding to Q^E? It is easy to see that this destroys every constraint corresponding to L-R neighbors and all the remaining binary constraints represent U-D neighbors. As these constraints form a Hamiltonian path, the remaining constraint graph consists of a union of disjoint paths. Such graphs have treewidth 1, hence the resulting CSP instance can be solved efficiently using Lemma 8, resulting in the running time $O(n^{k/2+2})$. A more careful argument improves this bound to $O(n^{k/2+1})$; we defer the details to the full version of the paper.

We can refine the analysis, noting that when we are assigning values $a_2 < a_4 < a_6 < \ldots$ to the variables x_2, x_4, x_6, ... representing Q^E, then we need to consider only increasing sequences where there is a gap of at least one between each successive entry (e.g. $a_4 > a_2 + 1$) to allow a value for the odd-indexed variables. The number of such subsequences is $\binom{n - \lceil k/2 \rceil}{\lfloor k/2 \rfloor}$: consider a sequence with a minimum required gap of one between consecutive entries, then distribute the remaining total gap of $n - k$ among the $\lfloor k/2 \rfloor + 1$ slots. As $\max_k \binom{n-k}{k} = O(1.6181^n)$, see e.g. [52, 51], we obtain an upper bound of this form (independent of k) on the running time of the algorithm.

Counting solutions. The algorithms described above can be made to work for the counting version of the problem. This has to be contrasted with the FPT algorithm of Guillemot and Marx [33], which cannot be adapted for the counting version: a crucial step in that algorithm is to say that if the text is sufficiently complicated, then it contains every pattern of length k, hence we can stop. Indeed, as we show in § 5, we cannot expect an FPT algorithm for the counting problem.

To solve the counting problem, we modify the dynamic programming algorithm behind Lemma 8 in a straightforward way. Even if not stated in exactly the following form, results of this type are implicitly used in the counting literature.

▶ **Lemma 10.** *The number of solutions of a binary CSP instance (V, D, C) can be computed in time $O(|D|^{t+1})$ where t is the treewidth of the constraint graph.*

It is not difficult to see that by replacing the use of Lemma 8 with Lemma 10 in the algorithms of Theorems 1 and 2, the counting algorithms stated in Theorem 3 follow.

4 Special patterns

In this section we prove Theorems 5 and 6. We define a *k-track graph* $G = (V, E)$ to be the union of two Hamiltonian paths H_1 and H_2, where V can be partitioned into *sequences* $S_1, S_2, \ldots, S_k$, the *tracks* of G, such that both H_1 and H_2 visit the vertices of S_i in the given order, for all $i \in [k]$. Observe that k-track graphs are exactly the incidence graphs of permutations that are either k-increasing or k-decreasing.

2-monotone patterns. We prove Theorem 5(i). As a special case, we first look at patterns that are 2-increasing. Let G be a 2-track graph. Arrange the vertices of the two tracks on a line ℓ, the first track in reverse order, followed by the second track in sorted order. Any Hamiltonian path that respects the order of the two tracks can be drawn (without crossings) on one side of ℓ. This means that the two Hamiltonian paths of G can be drawn on different sides of ℓ, and therefore G is planar. See Figure 2 (left) for an example.

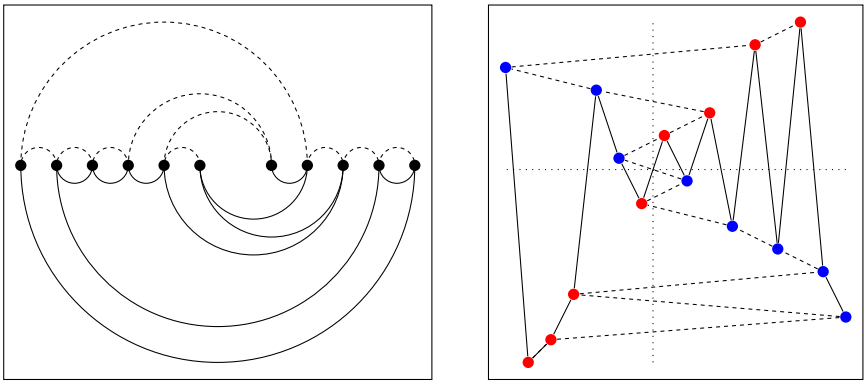


Figure 2 (*left*) A planar drawing of a 2-track graph, with one of the two Hamiltonian paths drawn with dashed arcs. Note that edges contained in both Hamiltonian paths are drawn twice for clarity. (*right*) A drawing of the incidence graph of a 2-monotone permutation. Red and blue dots indicate an increasing (resp. decreasing) subsequence.

The treewidth of a k-vertex planar graph is known to be $O(\sqrt{k})$ [11, 25]. A corresponding path-decomposition can be obtained by a recursive use of planar separators. For the case of a pattern π that consists of an increasing and a decreasing subsequence (i.e. 2-monotone patterns), we show that the straight-line drawing of G_π (with points S_π as vertices) has at most one intersection. An $O(\sqrt{k})$ bound on the treewidth follows via known results [26].

Divide S_π by one horizontal and one vertical line, such that each of the resulting four sectors contains a monotone sequence. More precisely, the top left and bottom right sectors contain decreasing subsequences, and the other two sectors contain increasing subsequences. Let $e = \{u, v\}$ be an edge of the horizontal Hamiltonian path such that $u.x = v.x - 1$, and let $f = \{s, t\}$ be an edge that intersects e, such that $s.x < t.x$. Edge f must come from the vertical Hamiltonian path, i.e. $|s.y - t.y| = 1$. As u and v are horizontal neighbors, $s.x < u.x < v.x < t.x$ holds. Assume that $u.y < v.y$ (otherwise flip G_π vertically before the argument, without affecting the graph structure). We claim that $u.y < t.y < s.y < v.y$ must hold, as otherwise π contains the pattern $(2, 1, 4, 3)$ and cannot decompose into an increasing and a decreasing subsequence.

Thus (s, u, v, t) must form the pattern $(3, 1, 4, 2)$, and therefore s and t belong to the decreasing and u and v to the increasing subsequence. It is easy to see now that s, u, v, t must be in pairwise distinct sectors, and u (s, t, v) is the unique rightmost (bottommost,

topmost, leftmost) point of the bottom left (top left, top right, bottom right) sector, and due to the monotonicity of all four sectors no more intersections can happen; see Figure 2 (right). This concludes the proof of Theorem 5(i).

We show that the previous result is tight, by constructing a 2-track graph $G = (V, E)$ with $n = 2k^2$ vertices, for some even k, that contains a $k \times 2k$ grid graph.

Let $x_1, x_2, \ldots, x_{k^2}$ and $y_1, y_2, \ldots, y_{k^2}$ be the two tracks of G. We obtain G as the union of the following two Hamiltonian paths:

$$x_1, y_1, y_2, x_2, x_3, y_3, y_4, x_4, \ldots, x_{k^2-1}, y_{k^2-1}, y_{k^2}, x_{k^2}; \quad \text{and}$$
$$x_1, x_2, \ldots, x_k,$$
$$y_1, x_{k+1}, x_{k+2}, y_2, y_3, x_{k+3}, x_{k+4}, y_4, \ldots y_{k^2-k-1} x_{k^2-1}, x_{k^2}, y_{k^2-k},$$
$$y_{k^2-k+1}, y_{k^2-k+2}, \ldots, y_{k^2}.$$

The two Hamiltonian paths respect the order of the two tracks.

We now relabel the vertices to show the contained grid. For $i \in [k]$, $j \in [2k]$, let $z_{i,j} = x_{\lfloor j/2 \rfloor k+i}$ if j is odd, and $z_{i,j} = y_{(j/2-1)k+i}$ if j is even. It is easy to see that $z_{i,j}$ is adjacent to $z_{i+1,j}$ and $z_{i,j+1}$ for $i \in [k-1]$ and $j \in [2k-1]$. For an illustration of the obtained permutation and the contained grid graph, see Figure 3.

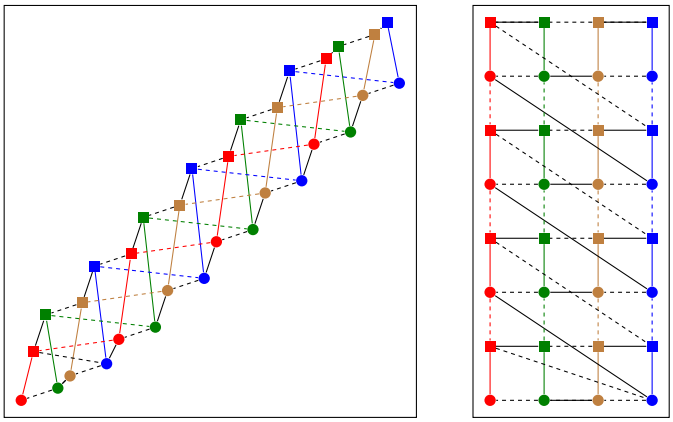


Figure 3 A 2-increasing permutation of length 32 whose incidence graph contains a 4×8 grid. Vertex shape indicates the track, colors are for emphasis of the grid structure.

The case of Jordan patterns (Theorem 5(ii)) is immediate, as the incidence graph of Jordan permutations is by definition planar. We defer the details to the full version of the paper.

3-monotone patterns. We now prove Theorem 6. Due to space constraints we omit some figures that illustrate the proof; these can be found in the full version of the paper. We start with some definitions and observations. For a set Π of length-n permutations, define the graph $G_H(\Pi) = ([n], E)$, where E is the union of the Hamiltonian paths corresponding to all $\pi \in \Pi$. For an arbitrary length-n permutation π, the graph G_π is isomorphic to $G_H(\{\mathrm{id}_n, \pi\})$, where id_n is the length-n identity permutation.

Permutation π' is a *split* of a permutation π if π' arises from π by moving a subsequence of π to the front. For example, $(1, 3, 5, 2, 4)$ is a split of id_5, obtained by moving $(1, 3, 5)$ to the front. We call a permutation *split permutation* if it is a split of the identity permutation. Observe that for a length-n split permutation $\sigma \neq \mathrm{id}_n$, there is a unique integer $p(\sigma) \in [n]$

such that both $\sigma(1), \sigma(2), \ldots, \sigma(p(\sigma))$ and $\sigma(p(\sigma)+1), \sigma(p(\sigma)+2), \ldots, \sigma(n)$ are increasing. Furthermore, σ^{-1} is a merge of the two subsequences $1, 2, \ldots, p(\sigma)$ and $p(\sigma)+1, p(\sigma)+2, \ldots, n$.

If π' is a split of π, then $\pi' = \pi \circ \sigma$ for some split permutation σ. Ahal and Rabinovich [1] mention that every n-permutation can be obtained from id_n by at most $\lceil \log n \rceil$ splits. Let π be an *arbitrary* n-permutation and consider the sequence $\mathrm{id}_n = \pi_1, \ldots, \pi_m = \pi$, where for each $i \in [m-1]$ we have $\pi_{i+1} = \pi_i \circ \sigma_i$ for some split permutation $\sigma_i \neq \mathrm{id}_n$, and $m \leq \lceil \log n \rceil$. Let $\Pi = \{\pi_1, \ldots, \pi_m\}$.

To prove Theorem 6, we show that the graph $G_H(\Pi)$ can be embedded (as a minor) in the incidence graph G of some permutation of length at most $2mn$. We further show that G is a 3-track graph (and thus, its underlying permutation can be assumed 3-increasing). The lower bound on the treewidth of G then follows by (i) choosing π to be a permutation whose incidence graph has treewidth $\Omega(n)$, (ii) the fact that G_π is a subgraph of $G_H(\Pi)$, and thus, a minor of G, and (iii) the observation that the treewidth of a graph is not less than the treewidth of its minor.

We first define the vertex sets corresponding to the three tracks of G. Let

$$V_x = \{x_{i,j} \mid i \in [m], j \in [n]\},$$
$$V_y = \{y_{i,j} \mid i \in [m-1], j \in [p(\sigma_i)]\}, \quad \text{and}$$
$$V_z = \{z_{i,j} \mid i \in [m-1], j \in [n] \setminus [p(\sigma_i)]\}.$$

Let $V = V_x \cup V_y \cup V_z$ be the vertex set of G, and observe that $|V| = mn + (m-1)n \leq 2mn$.

To later show that G is a 3-track graph, we fix a total order $\prec$ on each track, namely, the lexicographic order of the vertex-indices, i.e. $x_{i,j} \prec x_{i',j'}$ if and only if $i < i'$ or $(i = i') \wedge (j < j')$, and analogously for V_y and V_z. Before proceeding, we define the following functions:

$$s_x \colon V_x \setminus \{x_{m,n}\} \to V_x \setminus \{x_{1,1}\},$$
$$s_x(x_{i,j}) = \begin{cases} x_{i,j+1}, & \text{if } j < n, \\ x_{i+1,1}, & \text{if } j = n. \end{cases}$$
$$s_c \colon V \setminus \{x_{m,j} \mid j \in [n]\} \to V \setminus \{x_{1,j} \mid j \in [n]\},$$
$$s_c(x_{i,j}) = \begin{cases} y_{i,\sigma_i^{-1}(j)}, & \text{if } \sigma_i^{-1}(j) \leq p(\sigma_i), \\ z_{i,\sigma_i^{-1}(j)}, & \text{if } \sigma_i^{-1}(j) > p(\sigma_i). \end{cases}$$
$$s_c(y_{i,j}) = x_{i+1,j},$$
$$s_c(z_{i,j}) = x_{i+1,j}.$$

Note that s_x is just the successor with respect to the total order $\prec$ on V_x, and that s_c is a bijection.

Now we define the two Hamiltonian paths whose union is G. The first path P_1 goes as follows: start at $x_{1,1}$, then, from every $x_{i,j}$ with $i < m$, go to $s_c(x_{i,j})$, and then to $s_x(x_{i,j})$. For $x_{m,j}$ with $j < n$, go directly to $s_x(x_{m,j})$. Path P_1 contains all vertices of V_x in the correct order. The same holds for V_y and V_z, by the definition of s_c.

The second path P_2 also starts at $x_{1,1}$, but first goes along V_x until it reaches $x_{2,1}$, i.e. the first part of P_2 is $x_{1,1}, x_{1,2}, \ldots, x_{1,n}, x_{2,1}$. Then, from every $x_{i,j}$ with $i \geq 2$, it first moves to $s_c^{-1}(x_{i,j})$ and then to $s_x(x_{i,j})$. Again, P_2 contains all vertices of V_x in the correct order. As $s_c^{-1}(x_{i,j})$ is either $y_{i-1,j}$ or $z_{i-1,j}$, this is also true for V_y and V_z.

To obtain $G_H(\Pi) = ([n], E)$, color the vertices of the graph with n colors, where color k induces a path C_k of length m in G. We then prove that for each $\{k_1, k_2\} \in E$, the graph G

contains adjacent vertices of the colors k_1 and k_2. Then, by contracting C_k for $k \in [n]$, we obtain a supergraph of $G_H(\Pi)$.

For $k \in [n]$, define the path $C_k = (x_{1,k}, s_c(x_{1,k}), s_c^2(x_{1,k}), \ldots, s_c^{2m-2}(x_{1,k}))$. As s_c is a bijection, these paths are disjoint. Note that for each $x_{i,j} \in V_x \setminus \{x_{m,n}\}$,

$$s_c^2(x_{i,j}) = x_{i+1,\sigma_i^{-1}(j)}.$$

We claim that the color of $x_{i,j}$ is $\pi_i(j)$. This is because:

$$\begin{aligned} s_c^{2i-2}(x_{1,\pi_i(j)}) &= s_c^{2i-2}(x_{1,\sigma_1\sigma_2\ldots\sigma_{i-1}(j)}) \\ &= s_c^{2i-4}(x_{2,\sigma_1^{-1}\sigma_1\sigma_2\ldots\sigma_i(j)}) = s_c^{2i-4}(x_{2,\sigma_2\sigma_3\ldots\sigma_{i-1}(j)}) \\ &= \ldots = s_c^{2i-2\ell}(x_{\ell,\sigma_\ell\sigma_{\ell+1}\ldots\sigma_{i-1}(j)}) \\ &= \ldots = x_{i,j}. \end{aligned}$$

Now let k_1 and k_2 be adjacent in $G_H(\Pi)$. Then, there exist i, j such that $\pi_i(j) = k_1$ and $\pi_i(j+1) = k_2$ and, as discussed above, $x_{i,j} \in C_{k_1}$ and $x_{i,j+1} \in C_{k_2}$. By definition $x_{i,j} \in C_{k_1}$ implies $s_c^{-1}(x_{i,j}) \in C_{k_1}$. Finally, P_2 has an edge from $s_c^{-1}(x_{i,j})$ to $s_x(x_{i,j}) = x_{i,j+1}$. This concludes the proof.

The construction can be extended to embed the union of k arbitrary Hamiltonian paths on n vertices as a minor of a 3-track graph with $O(kn \log n)$ vertices. As every order-n graph of maximum degree d is edge-colorable with $d+1$ colors (by Vizing's theorem), such graphs are in the union of at most $d+1$ Hamiltonian paths, can thus be embedded in 3-track graphs of order $O(dn \log n)$.

5 Hardness result

In this section we prove Theorem 4. The hardness proof proceeds in two steps. First, we reduce the *partitioned subgraph isomorphism* (PSI) problem to the *partitioned permutation pattern matching* (PPPM) problem. Then, we reduce from the more difficult, counting variant of PPPM to the regular counting PPM (the subject of Theorem 4), using a (by now standard) technique based on inclusion-exclusion.

PSI to PPPM. The input to the PSI problem (introduced in [42]) consists of a graph G, a graph H, and a coloring ϕ of $V(G)$ with colors $V(H)$. The task is to decide whether there is a mapping $g : V(H) \to V(G)$ such that $\{u, v\} \in E(H)$ if and only if $\{g(u), g(v)\} \in E(G)$, and $\phi(g(u)) = u$ for all $u \in V(H)$. In words, we look for a subgraph of G that is isomorphic to H, with the restriction that each vertex of H can only correspond to a vertex of G from a prescribed set, moreover, these sets are disjoint.

Let n denote the number of vertices of G, and let k denote the number of *edges* of H. It is known [42, Corr. 6.3], that PSI cannot be solved in time $f(k) \cdot n^{o(k/\log k)}$, unless ETH fails, moreover, this holds even if $|E(H)| = |V(H)|$ (see e.g. [16]).

The input to the PPPM problem (introduced in [33]) consists of permutations τ and π of lengths n and k respectively, and a coloring $\phi : [n] \to [k]$ of the entries of τ. The task is to decide whether there is an embedding $g : [k] \to [n]$ of π into τ in the sense of the standard PPM problem, with the additional restriction that $\phi(g(i)) = i$, for all $i \in [k]$.

Guillemot and Marx show [33, Thm. 6.1], through a reduction from *partitioned clique*, that PPPM is $W[1]$-hard. Due to the density of a clique, the same reduction would, at best, yield a lower bound with exponent $\sqrt{k}$. We strengthen (and somewhat simplify) this reduction, to show that PPPM is at least hard as PSI, obtaining the following.

▶ **Lemma 11.** *PPPM cannot be solved in time $f(k) \cdot n^{o(k/\log k)}$, unless ETH fails.*

#PPPM to #PPM. The counting variant of PPPM (denoted #PPPM) is clearly at least as hard as PPPM. We now show that the counting variant of PPM (denoted #PPM) is at least as hard as #PPPM, thereby proving Theorem 4.

We use oracle-calls to #PPM for all subsets $X \subseteq [k]$, to count the number of embeddings of π into τ using entries of τ with colors from the set X, but ignoring colors for the purpose of the embedding. (We can achieve this by deleting the entries of τ with color in $[k] \setminus X$ before each oracle-call.) Then, using the inclusion-exclusion formula, we obtain the number C of embeddings that use *all* colors in $[k]$ as follows:

$$C = \sum_{X \subseteq [k]} (-1)^{k-|X|} C_X,$$

where C_X denotes the number of embeddings that use colors from the set X (obtained by oracle calls). Since π is of length k, the quantity C counts exactly the number of embeddings that use each color once.

It remains to show that embeddings that use every color in $[k]$ are such that π_i is matched to an entry of τ of color i, for all $i \in [k]$, i.e. the colors are not permuted. This is indeed the case for the hard instance constructed in the proof of Lemma 11. Towards this claim (referring to the details of the reduction in the full version of the paper) observe that all points that are unique in their respective pattern-cell (i, j) can only be matched to a point in the corresponding text-cell (i, j), which is of the correct color. In each diagonal cell (i, i), for $i > 0$, there is a matched point, and the pattern has two bracketing points in decreasing order in pattern-cell $(i, 0)$, and two bracketing points in increasing order in pattern-cell $(0, i)$. By construction, the only two points in the correct order are the nearest bracketing points in text-cell $(i, 0)$, resp. $(0, i)$, which are indeed of the correct color.

The number of oracle calls and additional overhead amounts to a factor 2^k in the running time, absorbed in the quantity $f(k) \cdot n^{o(k/\log k)}$. This concludes the proof.

References

1. Shlomo Ahal and Yuri Rabinovich. On Complexity of the Subpattern Problem. *SIAM J. Discrete Math.*, 22(2):629–649, 2008. doi:10.1137/S0895480104444776.
2. M.H. Albert and M.S. Paterson. Bounds for the growth rate of meander numbers. *Journal of Combinatorial Theory, Series A*, 112(2):250–262, 2005. doi:10.1016/j.jcta.2005.02.006.
3. Michael Albert and Mireille Bousquet-Mélou. Permutations sortable by two stacks in parallel and quarter plane walks. *Eur. J. Comb.*, 43:131–164, 2015. doi:10.1016/j.ejc.2014.08.024.
4. Michael H. Albert, Robert E. L. Aldred, Mike D. Atkinson, and Derek A. Holton. Algorithms for Pattern Involvement in Permutations. In *Proceedings of the 12th International Symposium on Algorithms and Computation*, ISAAC '01, pages 355–366, London, UK, 2001. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=646344.689586.
5. Michael H. Albert, Cheyne Homberger, Jay Pantone, Nathaniel Shar, and Vincent Vatter. Generating permutations with restricted containers. *J. Comb. Theory, Ser. A*, 157:205–232, 2018. doi:10.1016/j.jcta.2018.02.006.
6. Michael H. Albert, Marie-Louise Lackner, Martin Lackner, and Vincent Vatter. The Complexity of Pattern Matching for 321-Avoiding and Skew-Merged Permutations. *Discrete Mathematics & Theoretical Computer Science*, 18(2), 2016. URL: http://dmtcs.episciences.org/2607.
7. David Aldous and Persi Diaconis. Longest Increasing Subsequences: From Patience Sorting to the Baik-Deift-Johansson Theorem. *Bull. Amer. Math. Soc*, 36:413–432, 1999.

8 David Arthur. Fast Sorting and Pattern-avoiding Permutations. In *Proceedings of the Fourth Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2007*, pages 169–174, 2007. doi:10.1137/1.9781611972979.1.

9 Omri Ben-Eliezer and Clément L. Canonne. Improved Bounds for Testing Forbidden Order Patterns. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 2093–2112, 2018. doi:10.1137/1.9781611975031.137.

10 Donald J. Berndt and James Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, AAAIWS'94, pages 359–370. AAAI Press, 1994. URL: http://dl.acm.org/citation.cfm?id=3000850.3000887.

11 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

12 Miklós Bóna. A survey of stack-sorting disciplines. *The Electronic Journal of Combinatorics*, 9(2):1, 2003.

13 Miklós Bóna. *Combinatorics of Permutations*. CRC Press, Inc., Boca Raton, FL, USA, 2004.

14 Miklós Bóna. *A Walk Through Combinatorics: An Introduction to Enumeration and Graph Theory*. World Scientific, 2011. URL: https://books.google.de/books?id=TzJ2L9ZmlQUC.

15 Prosenjit Bose, Jonathan F. Buss, and Anna Lubiw. Pattern Matching for Permutations. *Inf. Process. Lett.*, 65(5):277–283, 1998. doi:10.1016/S0020-0190(97)00209-3.

16 Karl Bringmann, László Kozma, Shay Moran, and N. S. Narayanaswamy. Hitting Set for Hypergraphs of Low VC-dimension. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 23:1–23:18, 2016. doi:10.4230/LIPIcs.ESA.2016.23.

17 Marie-Louise Bruner and Martin Lackner. The computational landscape of permutation patterns. *Pure Mathematics and Applications*, 24(2):83–101, 2013.

18 Marie-Louise Bruner and Martin Lackner. A Fast Algorithm for Permutation Pattern Matching Based on Alternating Runs. *Algorithmica*, 75(1):84–117, 2016. doi:10.1007/s00453-015-0013-y.

19 Laurent Bulteau, Romeo Rizzi, and Stéphane Vialette. Pattern Matching for k-Track Permutations. In Costas Iliopoulos, Hon Wai Leong, and Wing-Kin Sung, editors, *Combinatorial Algorithms*, pages 102–114, Cham, 2018. Springer International Publishing.

20 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Pattern-Avoiding Access in Binary Search Trees. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 410–423, 2015. doi:10.1109/FOCS.2015.32.

21 Maw-Shang Chang and Fu-Hsing Wang. Efficient algorithms for the maximum weight clique and maximum weight independent set problems on permutation graphs. *Information Processing Letters*, 43(6):293–295, 1992. doi:10.1016/0020-0190(92)90114-B.

22 Hubie Chen. A rendezvous of logic, complexity, and algebra. *ACM Comput. Surv.*, 42(1):2:1–2:32, 2009. doi:10.1145/1592451.1592453.

23 Marek Cygan, Lukasz Kowalik, and Arkadiusz Socala. Improving TSP Tours Using Dynamic Programming over Tree Decompositions. In *25th Annual European Symposium on Algorithms, ESA 2017*, pages 30:1–30:14, 2017. doi:10.4230/LIPIcs.ESA.2017.30.

24 Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353–366, 1989. doi:10.1016/0004-3702(89)90037-4.

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

26 Vida Dujmovic, David Eppstein, and David R. Wood. Structure of Graphs with Locally Restricted Crossings. *SIAM J. Discrete Math.*, 31(2):805–824, 2017. doi:10.1137/16M1062879.

27 Pál Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935. URL: http://www.numdam.org/item/CM_1935__2__463_0.

28 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.

29 Fedor V. Fomin and Kjartan Høie. Pathwidth of cubic graphs and exact algorithms. *Information Processing Letters*, 97(5):191–196, 2006. doi:10.1016/j.ipl.2005.10.012.

30 Jacob Fox. Stanley-Wilf limits are typically exponential. *CoRR*, abs/1310.8378, 2013. arXiv:1310.8378.

31 Jacob Fox and Fan Wei. Fast property testing and metrics for permutations. *Combinatorics, Probability and Computing*, pages 1–41, 2018.

32 Eugene C. Freuder. Complexity of K-tree Structured Constraint Satisfaction Problems. In *Proceedings of the Eighth National Conference on Artificial Intelligence - Volume 1*, AAAI'90, pages 4–9. AAAI Press, 1990. URL: http://dl.acm.org/citation.cfm?id=1865499.1865500.

33 Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 82–101, 2014. doi:10.1137/1.9781611973402.7.

34 Sylvain Guillemot and Stéphane Vialette. Pattern Matching for 321-Avoiding Permutations. In Yingfei Dong, Ding-Zhu Du, and Oscar Ibarra, editors, *Algorithms and Computation*, pages 1064–1073. Springer Berlin Heidelberg, 2009.

35 Kurt Hoffmann, Kurt Mehlhorn, Pierre Rosenstiehl, and Robert E. Tarjan. Sorting Jordan sequences in linear time using level-linked search trees. *Information and Control*, 68(1-3):170–184, 1986.

36 Louis Ibarra. Finding pattern matchings for permutations. *Information Processing Letters*, 61(6):293–295, 1997. doi:10.1016/S0020-0190(97)00029-X.

37 Vít Jelínek and Jan Kyncl. Hardness of Permutation Pattern Matching. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 378–396, 2017. doi:10.1137/1.9781611974782.24.

38 Eamonn J. Keogh, Stefano Lonardi, and Bill Yuan-chi Chiu. Finding surprising patterns in a time series database in linear time and space. In *Proceedings of the Eighth ACM SIGKDD 2002 International Conference on Knowledge Discovery and Data Mining*, pages 550–556, 2002. doi:10.1145/775047.775128.

39 Sergey Kitaev. *Patterns in Permutations and Words*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2011. doi:10.1007/978-3-642-17333-2.

40 Donald E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms*. Addison-Wesley, 1968.

41 Adam Marcus and Gábor Tardos. Excluded permutation matrices and the Stanley-Wilf conjecture. *J. Comb. Theory, Ser. A*, 107(1):153–160, 2004. doi:10.1016/j.jcta.2004.04.002.

42 Dániel Marx. Can You Beat Treewidth? *Theory of Computing*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.

43 Both Neou, Romeo Rizzi, and Stéphane Vialette. Permutation Pattern matching in (213, 231)-avoiding permutations. *Discrete Mathematics & Theoretical Computer Science*, Vol. 18 no. 2, Permutation Patterns 2015, 2017. URL: https://dmtcs.episciences.org/3199.

44 Ilan Newman, Yuri Rabinovich, Deepak Rajendraprasad, and Christian Sohler. Testing for Forbidden Order Patterns in an Array. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 1582–1597, 2017. doi:10.1137/1.9781611974782.104.

45 Pranav Patel, Eamonn Keogh, Jessica Lin, and Stefano Lonardi. Mining Motifs in Massive Time Series Databases. In *In Proceedings of IEEE International Conference on Data Mining ICDM'02*, pages 370–377, 2002.

46 Vaughan R. Pratt. Computing Permutations with Double-ended Queues, Parallel Stacks and Parallel Queues. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 268–277. ACM, 1973. doi:10.1145/800125.804058.

47 Pierre Rosenstiehl. Planar permutations defined by two intersecting Jordan curves. In *Graph Theory and Combinatorics*, pages 259–271. London, Academic Press, 1984. URL: https://hal.archives-ouvertes.fr/hal-00259765.

48 Pierre Rosenstiehl and Robert E. Tarjan. Gauss codes, planar hamiltonian graphs, and stack-sortable permutations. *Journal of Algorithms*, 5(3):375–390, 1984. doi:10.1016/0196-6774(84)90018-X.

49 Raimund Seidel. A New Method for Solving Constraint Satisfaction Problems. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence, IJCAI 1981*, pages 338–342, 1981. URL: http://ijcai.org/Proceedings/81-1/Papers/062.pdf.

50 Rodica Simion and Frank W. Schmidt. Restricted Permutations. *European Journal of Combinatorics*, 6(4):383–406, 1985. doi:10.1016/S0195-6698(85)80052-4.

51 N. J. A. Sloane. The Encyclopedia of Integer Sequences, http://oeis.org. Sequence A073028.

52 S. M. Tanny and M. Zuker. On a unimodal sequence of binomial coefficients. *Discrete Mathematics*, 9(1):79–89, 1974. doi:10.1016/0012-365X(74)90073-9.

53 Robert E. Tarjan. Sorting Using Networks of Queues and Stacks. *J. ACM*, 19(2):341–346, April 1972. doi:10.1145/321694.321704.

54 Edward P. K. Tsang. *Foundations of constraint satisfaction*. Computation in cognitive science. Academic Press, 1993.

55 Vincent Vatter. Permutation Classes. In Miklós Bóna, editor, *Handbook of Enumerative Combinatorics*, chapter 12. Chapman and Hall/CRC, New York, 2015. Preprint at https://arxiv.org/abs/1409.5159.

56 V. Yugandhar and Sanjeev Saxena. Parallel algorithms for separable permutations. *Discrete Applied Mathematics*, 146(3):343–364, 2005. doi:10.1016/j.dam.2004.10.004.

Width Parameterizations for Knot-Free Vertex Deletion on Digraphs

Stéphane Bessy
Université de Montpellier - CNRS, LIRMM, Montpellier, France
stephane.bessy@lirmm.fr

Marin Bougeret
Université de Montpellier - CNRS, LIRMM, Montpellier, France
marin.bougeret@lirmm.fr

Alan D. A. Carneiro
Universidade Federal Fluminense - Instituto de Computação, Niterói, Brazil
aaurelio@ic.uff.br

Fábio Protti
Universidade Federal Fluminense - Instituto de Computação, Niterói, Brazil
fabio@ic.uff.br

Uéverton S. Souza[1]
Universidade Federal Fluminense - Instituto de Computação, Niterói, Brazil
ueverton@ic.uff.br

Abstract

A knot in a directed graph G is a strongly connected subgraph Q of G with at least two vertices, such that no vertex in $V(Q)$ is an in-neighbor of a vertex in $V(G) \setminus V(Q)$. Knots are important graph structures, because they characterize the existence of deadlocks in a classical distributed computation model, the so-called OR-model. Deadlock detection is correlated with the recognition of knot-free graphs as well as deadlock resolution is closely related to the Knot-Free Vertex Deletion (KFVD) problem, which consists of determining whether an input graph G has a subset $S \subseteq V(G)$ of size at most k such that $G[V \setminus S]$ contains no knot. Because of natural applications in deadlock resolution, KFVD is closely related to Directed Feedback Vertex Set. In this paper we focus on graph width measure parameterizations for KFVD. First, we show that: (i) KFVD parameterized by the size of the solution k is W[1]-hard even when p, the length of a longest directed path of the input graph, as well as κ, its Kenny-width, are bounded by constants, and we remark that KFVD is para-NP-hard even considering many directed width measures as parameters, but in FPT when parameterized by clique-width; (ii) KFVD can be solved in time $2^{O(tw)} \times n$, but assuming ETH it cannot be solved in $2^{o(tw)} \times n^{O(1)}$, where tw is the treewidth of the underlying undirected graph. Finally, since the size of a minimum directed feedback vertex set (dfv) is an upper bound for the size of a minimum knot-free vertex deletion set, we investigate parameterization by dfv and we show that (iii) KFVD can be solved in FPT-time parameterized by either $dfv + \kappa$ or $dfv + p$. Results of (iii) cannot be improved when replacing dfv by k due to (i).

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Knot, deadlock, width measure, FPT, W[1]-hard, directed feedback vertex set

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.2

Related Version A full version of the paper is available at `http://arxiv.org/abs/1910.01783`.

[1] corresponding author

14th International Symposium on Parameterized and Exact Computation (IPEC 2019).
Editors: Bart M. P. Jansen and Jan Arne Telle; Article No. 2; pp. 2:1–2:16

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

Funding Supported by Grant E-26/203.272/2017, Rio de Janeiro Research Foundation (FAPERJ) and by Grant 303726/2017-2, National Council for Scientific and Technological Development (CNPq).

Acknowledgements We thank Ignasi Sau for introducing Alan Carneiro to Stéphane Bessy and Marin Bougeret.

1 Introduction

The study of the Knot-Free Vertex Deletion problem emerges from its application in resolution of deadlocks, where a deadlock is detected in a distributed system and then a minimum cost deadlock-breaking set must be found and removed from the system. More precisely, distributed computations are usually represented by directed graphs called *wait-for graphs*. In a wait-for graph $G = (V, E)$, the vertex set V represents processes, and the set E of directed arcs represents wait conditions [4]. An arc exists in E directed away from $v_i \in V$ towards $v_j \in V$ if v_i is blocked waiting for a signal from v_j. The graph G changes dynamically according to a set of prescribed rules (the *deadlock model*), as the computation progresses. In essence, the deadlock model governs how processes should behave throughout computation, i.e., the deadlock model specifies rules for vertices that are not *sinks* (vertices with at least one out-neighbor) in G to become sinks [3] (vertices without out-neighbors). The two main classic deadlock models are the AND model, in which a process v_i can only become a sink when it receives a signal from *all* the processes in $N^+(v_i)$, where $N^+(v_i)$ stands for the set of out-neighbors of v_i (a conjunction of resources is needed); and the OR model, in which it suffices for a process v_i to become a sink to receive a signal from *at least one* of the processes in $N^+(v_i)$ (a disjunction of resources is sufficient). Distributed computations are dynamic, however deadlock is a stable property, in the sense that once it occurs in a consistent global state of a distributed computation, it still holds for all the subsequent states. Therefore, as it is typical in deadlock studies, G represents a static wait-for graph that corresponds to a *snapshot* of the distributed computation in the usual sense of a consistent global state [13]. Thus, the motivation of our work comes from *deadlock resolution*, where deadlocks are detected into a consistent global state G, and must be solved through some external intervention such as aborting one or more processes to break the circular wait condition causing the deadlock.

Deadlock resolution problems differ according to the considered deadlock model, i.e., according to the graph structure that characterizes the deadlock situation. In the AND-model, the occurrence of deadlocks is characterized by the existence of cycles [3, 5]. Therefore, deadlock resolution by vertex deletion in the AND-model corresponds precisely to the well-known Directed Feedback Vertex Set (DFVS) problem, proved to be NP-hard in the seminal paper of Karp [24], and proved to be FPT in [14]. On the other hand, the occurrence of deadlocks in wait-for graphs G working according to the OR-model are characterized by the existence of *knots* in G [5, 21]. A knot in a directed graph G is a strongly connected subgraph Q of G (with at least two vertices) such that there is no arc uv of G with $u \in V(Q)$ and $v \notin V(Q)$. Thus, deadlock resolution by vertex deletion in the OR-model can be viewed as the following problem.

Knot-Free Vertex Deletion (KFVD)
Instance: A directed graph $G = (V, E)$; a positive integer k.
Question: Determine if G has a set $S \subset V(G)$ such that $\|S\| \leq k$ and $G[V \setminus S]$ is knot-free.

Notice that a digraph G is knot-free if and only if for any vertex v of G, v has a path to a sink.

In [12], Carneiro, Souza, and Protti proved that KFVD is NP-complete; and, in [11], it was shown that KFVD is W[1]-hard when parameterized by k.

KFVD is closely related to DFVS not only because of their relation to deadlocks, but also some structural similarities between them: the goal of DFVS is to obtain a direct acyclic graph (DAG) via vertex deletion (in such graphs *all* maximal directed paths end at a sink); the goal of KFVD is to obtain a knot-free graph, and in such graphs for every vertex v there *exists* at least one maximal path containing v that ends at a sink. Finally, every directed feedback vertex set is a knot-free vertex deletion set; thus an optimum for DFVS provides an upper bound for KFVD. Although Directed Feedback Vertex Set is a well-known problem, this is not the case of Knot-Free Vertex Deletion, which we propose to analyze more deeply in this work.

Let S be a solution for KFVD, and let Z be the set of sinks in $G[V \setminus S]$. One can see that any $v \in V \setminus S$ has a path (that does not use any vertex in S) to a vertex in Z. Thus, KFVD can be seen as the problem of creating a set Z of sinks (doing at most k vertex removals) such that every remaining vertex has a path (in $G[V \setminus S]$) to a vertex in Z. In this paper, we denote the set of deleted vertices by S, and the set of sinks in $G[V \setminus S]$ by Z.

To get intuition on KFVD, note that the choice of the vertices to be removed must be carefully done, since the removal of a subset of vertices can turn some strongly connected components into new knots that will need to be broken by the removal of some internal vertices. Ideally, it is desirable to solve the current knots by removing as few vertices as possible for each knot, without creating new ones. Unfortunately, the generation of other knots can not always be avoided.

In [10, 12], Carneiro, Souza, and Protti present a polynomial-time algorithm for KFVD in graphs with maximum degree three. They also show that the problem is NP-complete even restricted to planar bipartite graphs G with maximum degree four. Later, in [11], a parameterized analysis of KFVD is presented, where it was shown that: KFVD is W[1]-hard when parameterized by the size of the solution; and it can be solved in $2^{k \log \varphi} n^{O(1)}$ time, but assuming SETH it cannot be solved in $(2-\epsilon)^{k \log \varphi} n^{O(1)}$ time, where φ is the size of the largest strongly connected subgraph.

Since the introduction of directed treewidth, much effort has been devoted to identify algorithmically useful digraph width measures [26]. Useful width measures imply polynomial time tractability for many combinatorial problems on digraphs of constant width. Since KFVD is W[1]-hard when parameterized by k, in this paper we investigate the ecology of width measures in order to find useful parameters to solve KFVD in FPT time. First, taking k as parameter, we show that KFVD remains W[1]-hard even on instances with both longest directed path and K-width bounded by constants. From the same reduction, it follows that KFVD is para-NP-hard even considering many width measures as parameters, such as directed treewidth and DAG-width. Contrasting with the hardness of KFVD on several directed width measure parameterizations, we show that KFVD is FPT when parameterized by the clique-width of the underlying undirected graph; and it can be solved in $2^{O(tw)} \times n$ time, but assuming ETH it cannot be solved in $2^{o(tw)} \times n^{O(1)}$ time, where tw is the treewidth of the underlying undirected graph. After that, we consider the most natural width parameter related to KFVD, the size of a minimum directed feedback vertex set (dfv). Such a parameter is at the same time a measure of the distance from the input graph to a DAG as well as an upper bound for the size of a minimum knot-free vertex deletion set. Finally, we show that KFVD can be solved in FPT time either parameterized by dfv and K-width, or dfv and the length of a longest directed path. The complexity of KFVD parameterized only by dfv remains open.

In the rest of this section we give necessary definitions and concepts used in this work. In Section 2 we present some useful observations and preliminary results. In Section 3 we discuss digraph width measures and show the W[1]-hardness. In Section 4 we discuss the consequences of treewidth parameterization. Finally, Section 5 we explore the directed feedback vertex set number as a parameter.

Due to space constraints, some proofs are omitted.

Additional notation. We use standard graph-theoretic and parameterized complexity notations and concepts, and any undefined notation can be found in [9, 17]. We consider here directed graphs. Given a vertex v and a subset of vertices Z, we say that there is a path from v to Z iff there exists $z \in Z$ such that there is a vz-(directed) path. For $v \in V(G)$, let $D(v)$ denote the set of descendants of v in G , i.e. nodes that are reachable from v by a non-empty directed path. Given a set of vertices $C = \{v_1, v_2, \ldots, v_p\}$ of G, we define $D(C) = \bigcup_{i=1}^{p} D(v_i)$. Let $A(v_i)$ denote the set of ancestors of v_i in G, i.e., nodes that reach v_i through a non-empty directed path. We also define $A[v_i] = A(v_i) \cup \{v_i\}$, and given a set of vertices $C = \{v_1, v_2, \ldots, v_p\}$ of G, we define $A(C) = \bigcup_{i=1}^{p} A(v_i)$. For a vertex v of G, the out-neighborhood of v is denoted by $N^+(v) = \{u | vu \in E\}$, and given a set of vertices $C = \{v_1, v_2, \ldots, v_p\}$, we define $N^+(C) = \bigcup_{i=1}^{p} N^+(v_i) \setminus C$. We refer to a Strongly Connected Component as an SCC. A knot in a directed graph G is an SCC Q of G with at least two vertices such that there is no arc uv of G with $u \in V(Q)$ and $v \notin V(Q)$. Finally, a sink (resp. a source) of G is a vertex with out-degree 0 (resp. in-degree 0). Given a subset of vertices S, we denote $G_S = G[S]$ and $\bar{S} = V \setminus S$. Thus, $G_{\bar{S}}$ denote the graph obtained by removing S.

We denote by $dfv(G)$ the size of a minimum directed feedback vertex set of G. We generally use F to denote a directed feedback vertex set and by R the remaining subset, i.e., $R = V \setminus F$. The length of a longest directed path of G is denoted by $p(G)$. The Kenny-width [18] or K-width of G is denoted by $\kappa(G)$ and is the maximum number of distinct directed st-paths in G over all pairs of distinct vertices $s, t \in V(G)$, where two st-paths are distinct iff they do not use the exact same set of arcs. For any function g (like dfv, κ, p), $g(G)$ will be denoted simply by g when the considered graph G can be deduced from the context. In what follows we denote by g-KFVD the KFVD problem parameterized by g ($g = k$ denotes the parameterization by the solution size).

2 Preliminaries

In this section we present some useful remarks and reduction rules. Remind that in the decision version of the problem we are given G and a positive integer k.

The first observation is immediate, as if we can make the graph acyclic, then it will be knot-free.

▶ **Observation 1.** *If $k \geq dfv(G)$ then G is a* yes*-instance.*

The two others observations are less obvious but rather natural.

▶ **Observation 2.** *Let S be a solution with set of sinks Z in $G_{\bar{S}}$, and $s \in S$. Let $S' = S \setminus \{s\}$ and Z' be the set of sinks of $G_{\bar{S'}}$. If there is a path from s to Z' in $G_{\bar{S'}}$ then S' is also a solution.*

Informally, after deleting a vertex s, we can add s back to the graph when it is certain that s has a path to a sink in the current graph. This is detailed by the following lemma and its corollary.

▶ **Lemma 1.** *Let S be a solution with set of sinks Z in $G_{\bar{S}}$. If there exists $s \in S$ with $s \notin N^+(Z)$, then $S' = S \setminus \{s\}$ is also a solution.*

▶ **Corollary 2.** *In any optimal solution S with set of sinks Z in $G_{\bar{S}}$, we have $N^+(Z) = S$.*

▶ **Observation 3.** *Let S be a knot-free vertex deletion with set of sinks Z in $G_{\bar{S}}$. If $|S| \leq k$ then for any vertex v with $d^+(v) > k$ it holds that $v \notin Z$.*

To complete the previous observations, we can design two general reduction rules.

▶ **Reduction Rule 1.** *If $v \in V(G)$ is an SCC of size one then remove $A[v]$.*

Proof. Let G' be the graph obtained by removing $A[v]$. Let of first show that (G, k) is a *yes*-instance implies that (G', k) is also a *yes*-instance. Let S be a solution of G of size at most k with set of sinks Z in $G_{\bar{S}}$. Let $S' = S \setminus A[v]$, and Z' the set of sinks in $G'_{\bar{S}'}$. Let us prove that every $u \in V(G'_{\bar{S}'})$ has a path ot Z' in $G'_{\bar{S}'}$. Let $u \in V(G'_{\bar{S}'})$. As u is also in $V(G_{\bar{S}})$, there is a uz-path P in $G_{\bar{S}}$ where $z \in Z$. As $u \notin A[v]$, $V(P) \cap A[v] = \emptyset$ and thus, the path P still exists in $G'_{\bar{S}'}$. Moreover, $u \notin A[v]$ implies that $N^+(z) \cap A[v] = \emptyset$, and thus that $N^+(v) \subseteq S'$, implying that $z \in Z'$.

Let us now consider the reverse implication, and let S' be a solution of G' of size at most k with set of sinks Z' in $G'_{\bar{S}'}$ and prove that S' is a solution of G. Let us start with $u \in V(G_{\bar{S}'}) \setminus A[v]$. As S' is a solution of G' and $u \in V(G'_{\bar{S}'})$, there is uz'-path P' in $G'_{\bar{S}'}$ where $z' \in Z'$, and this path still exists in $G_{\bar{S}'}$. As $N^+(z') \cap A[v] = \emptyset$, z' is still a sink in $G_{\bar{S}'}$ and we are done. Consider now a vertex $u \in V(G_{\bar{S}'}) \cap A[v]$. As $S' \cap A[v] = \emptyset$, there is uv-path P in $G_{\bar{S}'}$. If $N^+(v) \subseteq S'$ then v is a sink in $G_{\bar{S}'}$ and we are done. Otherwise, let $w \in N^+(v) \setminus S'$. As v is a SCC of size 1, $N^+(v) \cap A[v] = \emptyset$, implying that $w \in V(G_{\bar{S}'}) \setminus A[v]$, and thus according to the previous case w has a path to a sink in $G_{\bar{S}'}$. ◀

The previous reduction rule removes in particular sources and sinks, as they are SCC's of size one.

▶ **Reduction Rule 2.** *Let U_i be a strongly connected component of G with strictly more than k out-neighbors in $G[V \setminus V(U_i)]$. Then we can safely remove $A[U_i]$.*

Proof. Let G' be the graph obtained by removing $A[U_i]$. Let us first show that (G, k) is a *yes*-instance implies that (G', k) is also a *yes*-instance. Let S be a solution of G of size at most k and Z the set of sinks in $G_{\bar{S}}$. Let $S' = S \setminus A[U_i]$, and Z' the set of sinks in $G'_{\bar{S}'}$. Using the same argument (replacing $A[v]$ by $A[U_i]$) as in the first part of proof of Reduction 1, we get that every $u \in V(G'_{\bar{S}'})$ has a path ot Z' in $G'_{\bar{S}'}$.

Let us now consider the reverse implication, and let S' be a solution of G' of size at most k with set of sinks Z' in $G'_{\bar{S}'}$ and prove that S' is a solution of G. Let us start with $u \in V(G_{\bar{S}'}) \setminus A[v]$. As S' is a solution of G' there is uz'-path P' in $G'_{\bar{S}'}$ where $z' \in Z'$, and this path still exists in $G_{\bar{S}'}$. As $N^+(z') \cap A[U_i] = \emptyset$, z' is still a sink in $G_{\bar{S}'}$ and we are done. Consider now a vertex $u \in V(G_{\bar{S}'}) \cap A[U_i]$. As $S' \cap A[U_i] = \emptyset$, there is uU_i-path P in $G_{\bar{S}'}$. As U_i has strictly more than k out-neighbors in $G[V \setminus V(U_i)]$, there is arc from U_i to $w \in V(G_{\bar{S}'})$ and thus according to the previous case w has a path to a sink in $G_{\bar{S}'}$. ◀

3 W[1]-hardness and directed width measures

k-KFVD was shown to be W[1]-hard using a reduction from k-Multicolored Independent Set (k-MIS) [11]. However, the gadget used in this reduction to encode each color class has a longest directed path of unbounded length. First, we remark that it is possible to modify the reduction in order to prove that k-KFVD is W[1]-hard even if the input graph G has longest path length and K-width bounded by constants.

▶ **Theorem 3.** *There is a polynomial-time reduction, preserving the size of the parameter, from* k-MIS *to* k-KFVD *such that the resulting graph has longest directed path of length at most* 5 *and K-width equal to* 2.

Proof. Let (G', k) be an instance of Multicolored Independent Set, and let $V^1, V^2, \ldots, V^k$ be the color classes of G'. We construct an instance (G, k) of Knot-Free Vertex Deletion with bounded longest path length and K-width as follows.

1. for each $v_i \in V(G')$, create a directed cycle of size two with the vertices w_i and z_i in G;
2. for a color class V^j in G', create one vertex u_j;
3. for each vertex z_i in G corresponding to a vertex v_i of the color class V^j in G', create an arc from z_i to u_j and from u_j to z_i.
4. for each vertex w_i in G corresponding to a vertex v_i of the color class V^j in G', create an arc from u_j to w_i
5. for each edge $e_p = (v_i, v_l)$ in G' create a set X_p with two artificial vertices x_p^i and x_p^l and the arcs $x_p^i x_p^l$ and $x_p^l x_p^i$;
6. for each artificial vertex x_p^i, create an edge from x_p^i towards z_i in G.

Finally, set $Y_j = \{w_i, z_i \ : \ v_i \in V^j\} \cup \{u_j\}$, Y_j is the set of vertices of G corresponding to the vertices of G' in the same color class V^j. Notice that, the longest path of G has at most 5 vertices, and for any pair s, t in $V(G)$ there are at most 2 distinct directed st-paths in G.

Now, suppose that now S' is a k-independent set with exactly one vertex of each set V^j of G'. By construction, G has k knots which are $G[Y_1], \ldots, G[Y_k]$. Thus, at least k vertex removals are necessary to make G free of knots. We set $S = \{z_i \mid v_i \in S'\}$ and show that $G[V \setminus S]$ is knot-free. For $j = 1, \ldots, k$ the vertex w_j is a sink in $G \setminus S$, and every vertex of $Y_j \setminus S$ still reaches w_j. Now, as S' is a k-independent set of G' each set X_p in G is adjacent to at least one vertex that is not in S. Hence, each X_p will still have at least one arc pointing outside X_p, i.e., no new knots are created, and $G \setminus S$ is knot-free.

Conversely, suppose that G has a set of vertices S of size k such that $G[V \setminus S]$ is knot-free. In particular S has to contain exactly one vertex of each of the knot Y_j, for $j = 1, \ldots, k$. Since at least one sink has to be created in order to untie the knot Y_j, and since the only vertices of Y_j with only one out-neighbor are the w's ones, S has to contain a vertex z_i of each set $Y_1, \ldots, Y_k$. Moreover by deleting one vertex z_i in a knot Y_j, the vertex w_j is turned into sink and every other vertex of the same knot still has a path to w_j. Since $G[V \setminus S]$ is knot-free, no new knots are created by the deletion of S; thus, every SCC X_p will still have at least one arc pointing outside it. So, we set $S' = \{v_i \mid z_i \in S\}$. Since each SCC X_p corresponds to an edge of G', and at least one vertex of each edge of G' is not in S', the set S' contains no pair of adjacent vertices. Moreover, S' is composed by one vertex of each knot, which corresponds to a color of G'. Therefore, S' is a multicolored independent set of G'. ◀

▶ **Corollary 4.** k-KFVD *is W[1]-hard even if the input graph has longest directed path of length at most* 5 *and K-width equal to* 2.

After the introduction of the notion of directed treewidth (dtw) [23], a large number of width measures in digraphs were developed, such as: cycle rank [20] (cr); directed pathwidth [2] (dpw); zig-zag number [25] (zn); Tree-Zig-Zag number [26] (Tzn); Kelly-width [22] (Kelw); DAG-width [6] (dagw); D-width [29] (Dw); weak separator number [26] (s); entanglement [7] (ent); DAG-depth [18] (ddp). However, if a graph problem is hard when both the longest directed path length and the K-width are bounded, then it is hard for all these measures (see Figure 1).

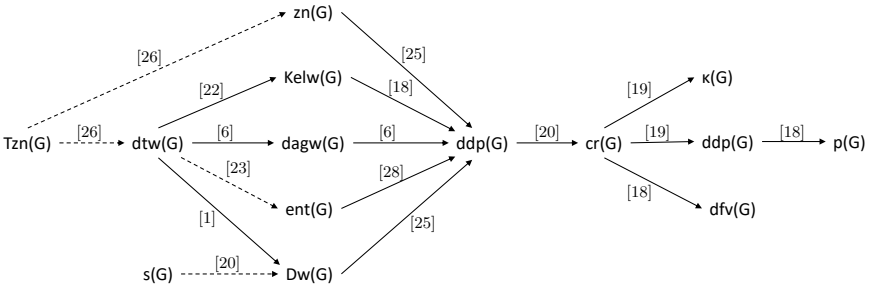


Figure 1 A hierarchy of digraph width measure parameters. $\alpha \to \beta$ indicates that $\alpha(G) \leq f(\beta(G))$ for any digraph G and some function f. More details about the relationships between these parameters can be found in the references corresponding to each arrow.

Therefore, from the reduction presented in Theorem 3 we can observe that KFVD is para-NP-hard with respect to all these width measures, and k-KFVD is W[1]-hard even on inputs where such width measures are bounded. Thus, it seems to be extremely hard to identify nice width parameters for which KFVD can be solved in FPT-time or even in XP-time. Fortunately, there remain some parameters for which, at least, XP-time solvability is achieved. One of them is the *directed feedback vertex set number* (dfv). This invariant is an upper bound on the size of a minimum knot-free vertex deletion set, so XP-time algorithms are trivial. This parameter is discussed in more detail in Section 5.

Another interesting width parameter for directed graphs G that is not bounded by a function of the K-width and the length of a longest directed path is the clique-width of G. Courcelle et al. [16] showed that every graph problem definable in LinEMSOL can be solved in time $f(w) \times n^{O(1)}$ on graphs with clique-width at most w, when a w-expression is given as input. Using a result of Oum [27], the same follows even if no w-expression is given.

▶ **Proposition 5.** *[15] There is a monadic second-order formula expressing the following property of vertices x, y and of a set of vertices X of a directed graph G: "$x, y \in X$ and there is a directed path from x to y in the subgraph induced by X".*

From Proposition 5 one can show that KFVD is LinEMSOL-definable. Thus Theorem 6 holds.

▶ **Theorem 6.** KFVD *is FPT when parameterized by clique-width of the underlying undirected graph.*

The fixed-parameter tractability for clique-width parameterization implies fixed-parameter tractability of KFVD for many other popular parameters. For example, it is well-known that the clique-width of a directed graph G is at most $2^{2tw(G)+2} + 1$, where $tw(G)$ is the treewidth of the underlying undirected graph (see [15, Proposition 2.114]). However, although Theorem 6 implies the FPT-membership of the problem parameterized by the treewidth of the underlying undirected graph, the dependence on $tw(G)$ provided by the model checking framework is huge. So, it is still a pertinent question whether such a parameterized problem admits a single exponential algorithm, which is discussed in Section 4.

4 The treewidth of the underlying undirected graph as parameter

Given a tree decomposition $\mathcal{T}$, we denote by t one node of $\mathcal{T}$ and by X_t the vertices contained in the *bag* of t. We assume w.l.o.g that $\mathcal{T}$ is a *nice* tree decomposition (see [17]), that is, we assume that there is a special root node r such that $X_t = \emptyset$ and all edges of the tree are directed towards r and each node t has one of the following four types: *Leaf*, *Introduce vertex*, *Forget vertex*, and *Join*.

Based on the following results we can assume that we are given a nice tree decomposition of G.

▶ **Theorem 7.** *[8] There exists an algorithm that, given an n-vertex graph G and an integer k, runs in time $2^{O(k)} \times n$ and either outputs that the treewidth of G is larger than k, or constructs a tree decomposition of G of width at most $5k + 4$.*

▶ **Lemma 8.** *[17] Given a tree decomposition $(T, \{X_t\}_{t \in V(T)})$ of G of width at most k, one can in time $O(k^2 \cdot \max(|V(T)|, |V(G)|))$ compute a nice tree decomposition of G of width at most k that has at most $O(k|V(G)|)$ nodes.*

Now we are ready to use a nice tree decomposition in order to obtain an FPT-time algorithm with single exponential dependency on $tw(G)$ and linear with respect to n.

▶ **Theorem 9.** Knot-Free Vertex Deletion *can be solved in $2^{O(tw)} \times n$ time, but assuming ETH there is no $2^{o(tw)} n^{O(1)}$ time algorithm for* KFVD*, where tw is the treewith of the underlying undirected graph of the input G.*

Proof. Let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of the input digraph G, with width equal to tw. First, we consider the following additional notation and definitions: t is the index of a bag of T; G_t is the graph induced by all vertices $v \in X_{t'}$ such that either $t' = t$ or $X_{t'}$ is a descendant of X_t in T; Given a knot-free vertex deletion set S, for any bag X_t there is a partition of X_t into S_t, Z_t, F_t, B_t where

- S_t (removed) is the set of vertices of X_t that are going to be removed ($S_t = S \cap X_t$);
- Z_t (sinks) is the set of vertices of X_t that are going to be turned into sinks after the removal of S;
- F_t (free/released) is the set of vertices of X_t that, after the removal of S, are going to reach a sink that belongs to $V(G_t)$;
- B_t (blocked) is the set of vertices of X_t that, after the removal of S, are going to reach *no* sink that belongs to $V(G_t)$;

Let $Y \subseteq X_t$. We denote by $A_t(Y)$ the set of vertices in F_t that reach some vertex of Y in the graph induced by $V(G_t) \setminus S_t$.

The recurrence relation of our dynamic programming has the signature $C[t, S_t, Z_t, F_t, B_t]$, representing the minimum number of vertices in G_t that must be removed in order to produce a graph such that for every remaining vertex v either v reaches a vertex in B_t (meaning that it may still be released in the future) or v reaches a vertex that became a sink (possibly the vertex itself), where every vertex in S_t is removed, every vertex in Z_t becomes a sink, every vertex in F_t will have a path to a sink in G_t, and S_t, Z_t, F_t, B_t form a partition of X_t. Notice that the generated table has size $4^{tw} \times tw \times n$, and when $t = r$, $X_t = \emptyset$ and therefore $C[r, \emptyset, \emptyset, \emptyset, \emptyset]$ contains the size of a minimum knot-free vertex deletion set of $G_r = G$.

The recurrence relation for each type of node is described as follows.

First, notice that if $v \in Z_t$ and there is an out-neighbor w of v that is not in S_t, there is an inconsistency, i.e. w must be deleted (must belong to S_t). In addition, if $v \in B_t$ but has an out-neighbor in $Z_t \cup F_t$, there is another inconsistency (v is not blocked), and if $v \in F_t$ but the removal of $S_t \cup B_t$ turns v into an isolated vertex, v is not released, and it must belong to B_t. For the inconsistent cases, $C[t, S_t, Z_t, F_t, B_t] = +\infty$. Such cases can be recognized and treated by simple preprocessing in linear time on the size of the table. Therefore, we consider next only consistent cases.

Leaf Node: If X_t is a leaf node then $X_t = \emptyset$. Therefore

$$C[t, \emptyset, \emptyset, \emptyset, \emptyset] = 0.$$

Insertion Node: Let X_t be a node of T with a child $X_{t'}$ such that $X_t = X_{t'} \cup \{v\}$ for some $v \notin X_{t'}$. We have the following:

$$C[t, S_t, Z_t, F_t, B_t] = \begin{cases} 1)\ case\ v \in S_t: \\ - \quad C[t', S_t \setminus \{v\}, Z_t, F_t, B_t] + 1, \\ 2)\ case\ v \in Z_t: \\ - \quad \min_{A' \subseteq A_t(v)} \{C[t', S_t, Z_t \setminus \{v\}, F_t \setminus A', B_t \cup A']\}, \\ 3)\ case\ v \in F_t: \\ - \quad \min_{A' \subseteq A_t(v)} \{C[t, S_t, Z_t, F_t \setminus \{A' \cup \{v\}\}, B_t \cup A']\}, \\ 4)\ case\ v \in B_t: \\ - \quad C[t', S_t, Z_t, F_t, B_t \setminus \{v\}] \end{cases}.$$

Recall that $A_t(v)$ is the set of vertices in F_t that reach v in the graph induced by $V(G_t) \setminus S_t$, i.e., the set of vertices that can be released by v if it was blocked in $G_{t'}$. Also note that, for simplicity, we consider only consistent cases, thus in case 2 it holds that $N^+(v) \cap X_t \subseteq S_t$, in case 3 it holds that $N^+(v) \cap (Z_t \cup F_t) \neq \emptyset$, and in case 4 it holds that $N^+(v) \cap \{Z_t \cup F_t\} = \emptyset$.

Forget Node: Let X_t be a forget node with a child $X_{t'}$ such that $X_t = X_{t'} \setminus \{v\}$, for some $v \in X_{t'}$. The forget node selects the best scenario considering all the possibilities for the forgotten vertex, discarding cases that lead to non-feasible solutions. In this problem, unfeasible cases are identified when the forgotten vertex v of $X_{t'}$ was blocked and reached no other node in B_t. Hence:

- If $N^+(v) \cap B_{t'} \neq \emptyset$ then

$$C[t, S_t, Z_t, F_t, B_t] = \min \begin{cases} C[t', S_t \cup \{v\}, Z_t, F_t, B_t], \\ C[t', S_t, Z_t \cup \{v\}, F_t, B_t], \\ C[t', S_t, Z_t, F_t \cup \{v\}, B_t], \\ C[t', S_t, Z_t, F_t, B_t \cup \{v\}] \end{cases}.$$

- If $N^+(v) \cap B_{t'} = \emptyset$ then

$$C[t, S_t, Z_t, F_t, B_t] = \min \begin{cases} C[t', S_t \cup \{v\}, Z_t, F_t, B_t], \\ C[t', S_t, Z_t \cup \{v\}, F_t, B_t], \\ C[t', S_t, Z_t, F_t \cup \{v\}, B_t], \end{cases}.$$

Join Node: Let X_t be a join node with children X_{t_1} and X_{t_2}, such that $X_t = X_{t_1} = X_{t_2}$. For any optimal knot-free vertex deletion set S of G it holds that $V(G_t) \cap S = \{V(G_{t_1}) \cap S\} \cup \{V(G_{t_2}) \cap S\}$. Clearly, if $S_t \subseteq S$ then we can assume that $S_t = S_{t_1} = S_{t_2}$. In addition, $Z_t = Z_{t_1} = Z_{t_2}$ otherwise we will have an inconsistency. Also note that a vertex is released in G_t if it reaches a vertex (possibly the vertex itself) that is released either in G_{t_1} or G_{t_2}. Thus:

$$C[t, S_t, Z_t, F_t, B_t] = \min_{\forall F', F''} \{C[t_1, S_t, Z_t, F', B'] + C[t_2, S_t, Z_t, F'', B'']\} - |S_t|,$$
$$\text{where } A_t(F' \cup F'') = F_t.$$

Note that $A_t(F' \cup F'')$ is the set of vertices that either are released in G_{t_i} $(i \in \{1, 2\})$ or can be released in G_t by vertices of $F' \cup F''$, even if they are blocked in both G_{t_1} and

G_{t_2}; this can occur, for example, if a blocked vertex v reaches another blocked node w in G_{t_1}, and in G_{t_2} vertex w is released.

Now, in order to run the algorithm, one can visit the bags of $\mathcal{T}$ in a bottom-up fashion, performing the queries described for each type of node. Since the reachability between the vertices of a bag can be stored in a bottom-up manner on $\mathcal{T}$, one can fill each entry of the table in $2^{O(tw)}$ time, and as the table has size $2^{O(tw)} \times n$, the dynamic programming can be performed in time $2^{O(tw)} \times n$.

Regarding correctness, let S^* be a minimum knot-free vertex deletion set of a digraph G with a tree decomposition $\mathcal{T}$. Let $S_t^*, Z_t^*, F_t^*, B_t^*$ be a partition of the vertices of X_t into removed, sinks, released and blocked, with respect to G_t after the removal of S^*. Note that $S_t^* = X_t \cap S^*$.

Fact 1. *There is no vertex $w \in V(G_t) \setminus X_t$ such that w reaches a vertex $v \in B_t^*$ in $G[V(G_t) \setminus S_t]$ and $w \in S^*$. Otherwise, since every vertex in B_t^* will reach a sink that is not in G_t, by Observation 2 one can remove from S^* every vertex that reaches B_t^* in $G[V(G_t) \setminus S_t]$, obtaining a subset of S^* which is also a knot-free vertex deletion set, contradicting the fact that S is minimum.*

This fact implies that the paths considered to compute $A_t(v)/A_t(F' \cup F')$' can in fact be used to release blocked vertices. Similarly, Fact 2 also holds.

Fact 2. *Let $\widehat{S}$ be a set for which the minimum is attained in the definition of $C[t, S_t^*, Z_t^*, F_t^*, B_t^*]$. Then $\widehat{S} \cup (S^* \setminus V(G_t))$ is also a solution (which is minimum) for* KFVD. *Otherwise, from $\widehat{S} \cup (S^* \setminus V(G_t))$ we can also obtain a knot-free vertex deletion set smaller than S^*, which is a contradiction.*

Fact 2 implies that we have stored enough information. At this point, the correctness of the recursive formulas is straightforward.

Finally, to show a lower bound based on ETH, we can transform an instance F of 3-SAT into an instance G_F of KFVD using the polynomial reduction presented in [11, Theorem 4], obtaining in polynomial time a graph with $|V| = 2n + 2m$, and so $tw = O(n + m)$. Therefore, if KFVD can be solved in $2^{o(tw)}|V|^{O(1)}$ time, then we can solve 3-SAT in $2^{o(n+m)}(n + m)^{O(1)}$ time, i.e., ETH fails. ◀

5 The size of a minimum directed feedback vertex set as parameter

Recall that k-KFVD is $W[1]$-hard (for fixed K-width and longest directed path) and that, as noticed in Observation 1, we can assume $k < dfv(G)$. This motivates us to determining the status of dfv-KFVD. In this section, we present two FPT-algorithms. Both with the size of a minimum directed feedback vertex set as parameter but with an aggregate parameter, the K-width, $\kappa(G)$, for the first one and the length of a longest directed path, $p(G)$, for the second one. Since finding a minimum directed feedback vertex set F in G can be solved in FPT-time (with respect to dfv) [14], we consider that F, a minimum DFVS, is given. Namely, we show that both (dfv, κ)-KFVD and (dfv, p)-KFVD are FPT.

At this point, we need to define the following variant of KFVD.

Disjoint Knot-Free Vertex Deletion (Disjoint-KFVD)
Instance: A directed graph $G = (V, E)$; a subset $X \subseteq V$; and a positive integer k.
Question: Determine if G has a set $S \subset V(G)$ such that $\|S\| \leq k$, $S \cap X = \emptyset$ and $G[V \setminus S]$ is knot-free.

We call *forbidden vertices* the vertices of the set X. It is clear that Disjoint-KFVD generalizes KFVD by taking $X = \emptyset$.

Let us now define two more steps that are FPT parameterized by dfv and that will be used for both (dfv, κ)-KFVD and (dfv, p)-KFVD. The next step will allow us to consider that the vertices of F are forbidden. We need the following straightforward observation.

▶ **Observation 4.** *Let (G, k) be an instance of* KFVD *and $v \in V(G)$.*

- *if (G, k) is a* yes*-instance and there exists a solution S with $v \in S$, then $(G \setminus \{v\}, k-1)$ is a* yes*-instance*
- *if $(G \setminus \{v\}, k-1)$ is a* yes*-instance then (G, k) is a* yes*-instance*

▶ **Branching 1** (On the directed feedback vertex set F)**.** *Let (G, F, k) be an instance of dfv-*KFVD*. In time $3^{dfv} \times O(n)$ we can build 3^{dfv} instances (G^i, F_1^i, X^i, k^i) of dfv-*Disjoint-KFVD *as follows. We consider all possible partitions of F into three parts: F_1, the set of vertices of F that will not be removed (i.e., they become forbidden); F_2, the set of vertices in F that will be removed; and F_3, the set of vertices in F that will be turned into sinks. For each such a partition (indicated by the index i), we remove the set $Y^i = F_2^i \cup N^+(F_3^i)$ of vertices and we apply exhaustively Reduction Rules 1 and 2 (see Section 2). We denote by G^i the obtained graph, $X^i = F_1^i$, and $k^i = k - |Y|$.*

According to Observation 4, it is clear that (G, F, k) is a *yes*-instance of dfv-KFVD if and only if one of the instances (G^i, F_1^i, X^i, k^i), $1 \leq i \leq 3^{dfv}$, of dfv-Disjoint-KFVD is a *yes* instance. Since there are at most 3^{dfv} partitions of F, the branching reduction can be performed in FPT time. Although at this point $X^i = F_1^i$, in the next steps some vertices of $V(G) \setminus F_1$ may become forbidden and therefore should be added to X^i. Also, from this point forward, we assume that we are given an instance (G, F_1, X, k) of dfv-Disjoint-KFVD.

Notice that after applying Reduction Rule 1 (Section 2), each strongly connected component of G is at least of size two. Thus, each of them must contain at least one cycle; therefore, the number of strongly connected components of G is bounded by dfv. Moreover, for any strongly connected component U of G, Reduction Rule 2 gives an upper bound for the number of vertices in $N^+(V(U))$ (i.e., vertices that are not in U but it is out-neighbour of some vertex in U). This implies that G has at most $dfv \times k \leq dfv^2$ such vertices between its strongly connected components. This observation leads to a branching rule.

▶ **Branching 2** (On strongly connected components)**.** *Let S_H be the set of vertices that are extremities of arcs between the strongly connected components of G. We have $|S_H| \leq 2 \times dfv \times k \leq 2 \times dfv^2$ and we can branch in FPT-time trying all possible partitions of S_H into two sets: S_1, the set of vertices to be deleted in G such that $|S_1| \leq k$; and $S_2 = S_H \setminus S_1$, the set of vertices marked as forbidden, and then added into X.*

Notice that this step involves a $2^{|S_H|}$ branching. At this point, we may consider that we have an instance (G, F, X, k) where $F \subseteq X$ and such that for any arc uv between two SCC's U_i and U_j, $\{u, v\} \subseteq X$. We call such an instance as a *nice* instance.

▶ **Lemma 10** (After cleaning of Branching 2)**.** *If there is an algorithm running in time $g(dfv) \times poly(n)$ for dfv-*Disjoint-KFVD *restricted to nice instances that are strongly connected, then there is an FPT algorithm running in time $g(dfv) \times poly(n) \times c.n.log(dfv)$ (where c is a constant) to solve dfv-*Disjoint-KFVD *for any nice instance.*

Proof. Let (G, F, X, k) be a nice instance and S be a solution. Let $\mathcal{U} = \{U_1, \ldots, U_s\}$ be the partition of $V(G)$ where each U_i is an SCC, and let $\mathcal{K} = \{U_i : U_i \text{ is a knot}\}$. Without loss of generality we can assume that $\mathcal{K} = \{U_1, \ldots, U_t\}$ for some $t \leq s$. Let $S_i = S \cap U_i$. Notice that if S is a solution then for any $i \in [t]$, S_i is a solution of $(G[U_i], F \cap U_i, X \cap U_i, |S_i|)$. Moreover, for any solutions S_i' to $(G[U_i], F \cap U_i, X \cap U_i, |S_i'|)$ where $\sum_{i=1}^{t} |S_i'| \leq k$, $S' = \bigcup_{i=1}^{t} S_i'$ will be

a solution to (G, F, X, k) because vertices of some $U_j \notin \mathcal{K}$ will still have a path to a set $U_i \in \mathcal{K}$ in $G_{\bar{S}'}$ since any arc between two SCC's has forbidden endpoints. Thus, given a nice instance (G, F, X, k) and an algorithm A for a nice instance restricted to one SCC, for any $U_i \in \mathcal{K}$ we perform a binary search to find the smallest k_i such that $A(G[U_i], F \cap U_i, X \cap U_i, k_i)$ answers *yes*, and we answer *yes* iff $\sum_{i=1}^{t} k_i \leq k$. ◀

From Lemma 10, we may assume that we have an instance (G, F, X, k) such that $F \subseteq X$ and G is strongly connected (there is only one SCC). We call such an instance as a *super nice* instance.

5.1 Combining DFVS-number and K-width

In this section, we prove that (dfv, κ)-Disjoint-KFVD restricted to super nice instances is FPT.

Let $F = \{v_1, \ldots, v_{dfv}\}$. For every pair of integers i, j with $1 \leq i, j \leq dfv$ we define $H_{i,j}$ as the (i,j)-*connectivity set*, that is, the set of vertices which are contained in a directed path from v_i to v_j in the induced subgraph $G[V \setminus (F \setminus \{v_i, v_j\})]$ (if $i = j$ then $H_{i,i}$ is the set of vertices contained in a cycle in $G[V \setminus (F \setminus \{v_i\})]$). Let us define a set B on which we will later branch in a way to ensure connectivity among different connectivity sets. We start with $B = \{\emptyset\}$, and then, for each possible pair of connectivity sets $H_{i,j}$, $H_{i',j'}$ we increase B as follows:

- **(i)** add $N^+(H_{i,j} \setminus H_{i',j'}) \cap H_{i',j'}$ to B.
- **(ii)** add $N^+(H_{i,j} \cap H_{i',j'}) \cap (H_{i',j'} \setminus H_{i,j})$ to B.
- **(iii)** add $N^+(H_{i',j'} \setminus H_{i,j}) \cap H_{i,j}$ to B.
- **(iv)** add $N^+(H_{i',j'} \cap H_{i,j}) \cap (H_{i,j} \setminus H_{i',j'})$ to B.

For a given pair of connectivity sets, in each of the items *i*), *ii*), *iii*) and *iv*) the number of added vertices to B is at most κ. For instance,let $y_1, \ldots, y_l$ be the vertices added by item *i*), where each $y_s \in N^+(H_{i,j} \setminus H_{i',j'}) \cap H_{i',j'}$. By definition, there exist vertices $x_1, \ldots, x_l$ of $H_{i,j} \setminus H_{i',j'}$ such that $x_s y_s$ are arcs of G for $s = 1, \ldots, l$. Notice that while the y_s's are distinct, the x_s's are not forced to be so. For any $s \in \{1, \ldots, l\}$, there exists a path P_s in $H_{i',j'}$ from y_s to $v_{j'}$, and such a path does not intersect $H_{i,j} \setminus H_{i',j'}$. In the same way, by finding a path Q_s from v_i to x_s for every $s \in \{1, \ldots, l\}$, we form l distinct paths $Q_s P_s$ from v_i to $v_{j'}$, implying $l \leq \kappa$, the K-width of G. So, as there are dfv^2 different connectivity sets, at the end of the process B contains at most $\kappa \times dfv^4$ vertices. Figure 2 shows examples of vertices to be added in B regarding the interaction of two different connectivity sets.

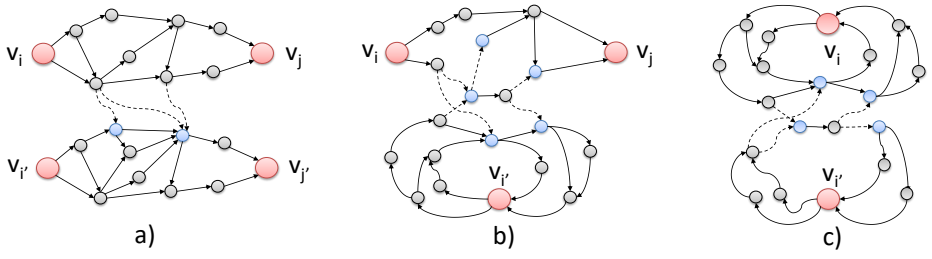


Figure 2 a) two connectivity sets with no intersection. b) an intersection with two vertices belonging to both connectivity sets. c) two connectivity sets $H_{i,j}$ with $i = j$. Vertices to be added in B are marked in blue.

Next we establish our last branching rule.

▶ **Branching 3** (On the connectivity sets). *We branch by partitioning B into three parts: B_1, the set of vertices that will not be removed (ie. they become forbidden); B_2, the set of vertices that will be removed; and B_3, the set of vertices that will become sinks. Since $|B| \leq \kappa \times dfv^4$, we branch at most $3^{\kappa.dfv^4}$ times.*

At this point, without loss of generality, one can assume that none of the above branching and reductions rules are applicable. Hence, the analysis boils down to the case where $F \cup B \subseteq X$, meaning that all the vertices of $F \cup B$ are forbidden to be deleted or become sinks, and G is strongly connected.

▶ **Observation 5** (The consequences of Branching 3). *Let G be a graph for which no Reduction Rules 1 and 2 or Branching Rules 1 to 3 can be applied. Let $H_{i,j}$ and $H_{i',j'}$ be two different connectivity arc sets in G. If there is an arc from $H_{i,j} \setminus H_{i',j'}$ to $H_{i',j'} \setminus H_{i,j}$ or $H_{i,j} \cap H_{i',j'}$ to $H_{i',j'} \setminus H_{i,j}$ in $G[H_{i,j} \cup H_{i',j'}]$, then the head vertex of such an arc is a forbidden vertex.*

We now aim to show that, for any vertex v^* such that v^* can be turned into a sink, that is, $N^+(v^*) \cap X = \emptyset$ and $d^+(v^*) \leq k$, the deletion of $N^+(v^*)$ is sufficient for G to become knot-free.

▶ **Lemma 11.** *Let (G, F, X, k) be an instance of (dfv, κ)-Disjoint-KFVD such that G is strongly connected and none of the branching and reduction rules can be applied. If there is a vertex v^* with no forbidden out-neighbors, then $G[V \setminus N^+(v^*)]$ is knot-free.*

Proof. Let (G, F, k, X) and v^* as stated. Denote by G' the resulting graph, i.e, $G' = G[V \setminus N^+(v^*)]$. For contradiction, assume that G' contains a knot K. As G is strongly connected, K was not a knot in G, implying that there exists an arc xy of G such that $x \in V(K)$ and $y \in N^+(v^*)$. Notice that $v^* \notin F$ since vertices from F cannot become sinks and $y \notin X$, since y has to be deleted in order to v^* to become a sink. Let us now define a connectivity set containing both y and v^*. Let s be any source of the DAG $G[V \setminus F]$ such that there is a sv^* path in $G[V \setminus F]$, and let z be any sink of $G[V \setminus F]$ such that there is a yz path in $G[V \setminus F]$. As G is strongly connected, there exist arcs $v_i s$ and $z v_j$ where $\{v_i, v_j\} \subseteq F$ and we get that $\{v^*, y\} \subseteq H_{i,j}$. Notice that $i = j$ is possible. Similarly, as $G[V(K)]$ is strongly connected, it contains a cycle C' containing x and thus there exists a connectivity set $H_{k,l}$ containing a path P from v_k to v_l which is a subpath of $G[V(K)]$ containing x, and with $\{v_k, v_l\} \subseteq V(K)$. Notice first that $v^*, y \notin F$. In addition, v^* is not a vertex of $H_{k,l}$, otherwise there would exist a path P' from v_k to v^* containing no vertex of $F \setminus \{v_k\}$, which is not possible. Indeed, either $V(P') \cap N^+(v^*) = \emptyset$ and we would get that K is not a knot, or $V(P') \cap N^+(v^*) \neq \emptyset$, implying that there is a cycle with no vertex of F. Thus, as y was not a forbidden vertex, it means that $y \notin H_{k,l}$ otherwise the arc v^*y would go from $H_{i,j} \setminus H_{k,l}$ to $H_{i,j} \cap H_{k,l}$ and y should be forbidden by Branching 3 item i). Then we have $y \in H_{i,j} \setminus H_{k,l}$. Similarly, we have $x \notin H_{i,j} \cap H_{k,l}$, otherwise by item ii) of Branching 3, vertex y would be forbidden. Finally $x \in H_{k,l} \setminus H_{i,j}$ and $y \in H_{i,j} \setminus H_{k,l}$, since $(H_{i,j} \setminus H_{k,l}) \subseteq H_{i,j}$, and by item iii) of Branching 3, vertex y would again be a forbidden vertex, a contradiction. ◀

In conclusion, by Lemma 11, we can find in polynomial time the optimum solution for G: we choose a vertex v^* with minimum out-degree.

▶ **Theorem 12.** Knot-free Vertex Deletion *can be solved in $2^{O(\kappa dfv^5)} \times n^{O(1)}$.*

Proof. Let us now compute the running time of the overall algorithm. First notice that applying Branchings 1 and 2 results in $3^{dfv} \times 2^{2dfv^2}$ branches. Branching 3 can be done

in time $3^{\kappa.dfv^4}$, but may re-create several SCC's, forcing us to apply again Branching 2 and reduction rules again, but decreasing k. This implies that the total running time is $3^{dfv} \times (2^{2dfv^2}3^{\kappa.dfv^4})^k \times n^{O(1)}$, thus the result holds. ◀

5.2 Combining DFVS-number and length of a longest directed path

In this subsection we investigate the length of a longest path and $dfv(G)$ as aggregate parameters.

▶ **Lemma 13.** *(dfv, p)-Disjoint-KFVD on super nice instances can be solved in $2^{O(dfv^3)}p^{O(dfv)} \times n^{O(1)}$.*

Proof. Let (G, F, X, k) be a super nice instance. Recall that the directed feedback vertex set F is a set of forbidden vertices ($F \subseteq X$) and G is strongly connected. The proof is by induction on $|F|$. If $|F| = 1$, then, for any vertex v of $V(G) \setminus F$ that can be turned into a sink, $N^+(v)$ will be a solution set for G. Therefore, the optimum solution can be found in polynomial time. Assume now that $|F| \geq 2$ and denote F by $\{v_1, \ldots, v_{dfv}\}$. As G is strongly connected, there exists a path P_1 of length at most p from v_1 to v_2 and a path P_2 of length at most p from v_2 to v_1. Denote by C the digraph $G[V(P_1) \cup V(P_2)]$; it is strongly connected, contains v_1 and v_2 and at most $2p$ vertices. Since the number of vertices in C is bounded, we may branch $2p + 1$ times by trying to guess a vertex that will be deleted in C. Each time a vertex of C will be guessed as deleted, the parameter k will decrease by one. So, k will decrease in all branches, except in the one where we guess that no vertex is deleted, and then where all the vertices of C are forbidden. In this case, C is a strongly connected component whose vertices are all forbidden and containing at least two vertices of F. So, we contract C to obtain a new instance G'. Formally, we remove $V(C)$ from G, add a new vertex v_C, and for all vertices of $G \setminus C$ having at least one in-neighbor (resp. out-neighbor) in C, we add an arc from v_C (resp. to v_C) to this vertex. Let F' be the set $\{v_C\} \cup F \setminus V(C)$ and notice that F' is a directed feedback vertex set of G' and that $|F'| < |F|$. Similarly, let X' be the set $(X \setminus V(C)) \cup \{v_C\}$. We claim that both instances (G, F, k, X) and (G', F', k, X') are equivalent. Indeed, it suffices to notice that as $V(C)$ contains only forbidden vertices in G and that v_C is forbidden in G', then any solution to the KFVD problem for G is a solution of G', and conversely. Then, we apply Branchings 1 and 2 to obtain a super nice instance equivalent to (G', F', k, X'), and we can apply the induction hypothesis.
So at each branching, either the parameter k decreases by at least one or the size of F decreases by at least one. As both values are bounded above by dfv, we branch consecutively at most $2dfv$ times. And since Branching rules 1 and 2 create at most $3^{dfv} \times 2^{2dfv^2}$ branches, and branching on cycle C creates $2p + 1$ branches, the total number of branches is $(3^{dfv} \times 2^{2dfv^2} \times (2p + 1))^{2dfv} = 2^{O(dfv^3)}p^{O(dfv)}$, and we get the desired running time. ◀

Given that we can obtain a super nice instance in $2^{O(dfv^3)} \times n^{O(1)}$, it holds that Knot-free Vertex Deletion can be solved in time $2^{O(dfv^3)}p^{O(dfv)} \times n^{O(1)}$.

References

1 Saeed Akhoondian Amiri, Lukasz Kaiser, Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Graph searching games and width measures for directed graphs. In *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

2 János Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2):161–172, 2006.

3 Valmir C. Barbosa. The Combinatorics of Resource Sharing. In *Models for Parallel and Distributed Computation*, pages 27–52. Springer, 2002.

4 Valmir C. Barbosa and Mario R. F. Benevides. A graph-theoretic characterization of AND-OR deadlocks. Technical Report COPPE-ES-472/98, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, 1998.

5 Valmir C. Barbosa, Alan D. A. Carneiro, Fábio Protti, and Uéverton S. Souza. Deadlock Models in Distributed Computation: Foundations, Design, and Computational Complexity. In *Proceedings of the 31st ACM/SIGAPP Symposium on Applied Computing*, pages 538–541, 2016.

6 Dietmar Berwanger, Anuj Dawar, Paul Hunter, Stephan Kreutzer, and Jan Obdržálek. The dag-width of directed graphs. *Journal of Combinatorial Theory, Series B*, 102(4):900–923, 2012.

7 Dietmar Berwanger and Erich Grädel. Entanglement – A Measure for the Complexity of Directed Graphs with Applications to Logic and Games. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 209–223, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

8 Hans L Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A $c^k n$ 5-Approximation Algorithm for Treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016.

9 John A. Bondy and Uppaluri S. R. Murty. *Graph theory with applications*, volume 290. Macmilan, 1976.

10 Alan D. A. Carneiro, Fábio Protti, and Uéverton S. Souza. Deletion Graph Problems Based on Deadlock Resolution. In *The 23rd International Computing and Combinatorics Conference, COCOON 2017, Hong Kong, China, August 3-5, 2017. Lecture Notes in Computer Science*, volume 10392, pages 75–86. Springer, 2017.

11 Alan D. A. Carneiro, Fábio Protti, and Uéverton S. Souza. Fine-Grained Parameterized Complexity Analysis of Knot-Free Vertex Deletion – A Deadlock Resolution Graph Problem. In *The 24th International Computing and Combinatorics Conference, COCOON 2018, Qingdao, China , July 2-4, 2018. Lecture Notes in Computer Science*, volume 10976, pages 84–95. Springer, 2018.

12 Alan D. A. Carneiro, Fábio Protti, and Uéverton S Souza. Deadlock resolution in wait-for graphs by vertex/arc deletion. *Journal of Combinatorial Optimization*, 37(2):546–562, 2019.

13 K Mani Chandy and Leslie Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3:63–75, 1985.

14 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.

15 Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic: a language-theoretic approach*, volume 138. Cambridge University Press, 2012.

16 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.

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

18 Robert Ganian, Petr Hliněný, Joachim Kneis, Alexander Langer, Jan Obdržálek, and Peter Rossmanith. Digraph width measures in parameterized algorithmics. *Discrete applied mathematics*, 168:88–107, 2014.

19 Robert Ganian, Petr Hliněný, Joachim Kneis, Daniel Meister, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Are there any good digraph width measures? In *International Symposium on Parameterized and Exact Computation*, pages 135–146. Springer, 2010.

20 Hermann Gruber. Digraph Complexity Measures and Applications in Formal Language Theory. *Discrete Mathematics & Theoretical Computer Science*, 14(2):189–204, 2012.

21 Richard C Holt. Some deadlock properties of computer systems. *ACM Computing Surveys (CSUR)*, 4(3):179–196, 1972.

22 Paul Hunter and Stephan Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theoretical Computer Science*, 399(3):206–219, 2008. Graph Searching.

23 Thor Johnson, Neil Robertson, P.D. Seymour, and Robin Thomas. Directed Tree-Width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.

24 RichardM. Karp. Reducibility among Combinatorial Problems. In RaymondE. Miller, JamesW. Thatcher, and JeanD. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

25 Mateus de O. Oliveira. Subgraphs satisfying MSO properties on z-topologically orderable digraphs. In *International Symposium on Parameterized and Exact Computation*, pages 123–136. Springer, 2013.

26 Mateus de O. Oliveira. An algorithmic metatheorem for directed treewidth. *Discrete Applied Mathematics*, 204:49–76, 2016.

27 Sang-Il Oum. Approximating Rank-width and Clique-width Quickly. *ACM Transactions on Algorithms*, 5(1):10:1–10:20, 2008.

28 Roman Rabinovich and Lehr-und Forschungsgebiet. *Complexity measures of directed graphs*. PhD thesis, RWTH Aachen University, 2008.

29 Mohammad Ali Safari. D-Width: A More Natural Measure for Directed Tree Width. In Joanna Jędrzejowicz and Andrzej Szepietowski, editors, *Mathematical Foundations of Computer Science 2005*, pages 745–756, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

Parameterized Valiant's Classes

Markus Bläser
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
mblaeser@cs.uni-saarland.de

Christian Engels
IIT Bombay, Mumbai, India
christian@cse.ittb.ac.in

Abstract

We define a theory of parameterized algebraic complexity classes in analogy to parameterized Boolean counting classes. We define the classes VFPT and VW[t], which mirror the Boolean counting classes #FPT and #W[t], and define appropriate reductions and completeness notions. Our main contribution is the VW[1]-completeness proof of the parameterized clique family. This proof is far more complicated than in the Boolean world. It requires some new concepts like composition theorems for bounded exponential sums and Boolean-arithmetic formulas. In addition, we also look at two polynomials linked to the permanent with vastly different parameterized complexity.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases Algebraic complexity theory, parameterized complexity theory, Valiant's classes

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.3

Related Version https://arxiv.org/abs/1907.12287

Acknowledgements We thank Holger Dell for valuable comments on a first draft of this paper and Radu Curticapean and Marc Roth for helpful discussions.

1 Introduction

When Valiant invented the theory of computational counting and #P-completeness, he also defined an algebraic model for computing families of polynomials [24]. This was very natural, since many (Boolean) counting problems are evaluations of polynomials: Counting perfect matchings in bipartite graphs is the same as evaluating the permanent at the adjacency matrix, counting Hamiltonian tours in directed graphs is the same as evaluating the Hamiltonian cycle polynomial at the adjacency matrix, etc. There is a fruitful interplay between the Boolean and the algebraic world: algebraic methods like interpolation can be used to design counting algorithms as well as for proving hardness results. Proving lower bounds might be easier in the algebraic world and then we can use transfer theorems from the algebraic world to the Boolean world [3].

Parameterized counting complexity has been very successful in the recent years, see for instance [1, 8, 6, 7, 16, 23]. Parameterized complexity provides a more fine-grained view on #P-complete problems. There are problems like counting vertex covers of size k, which are fixed-parameter tractable, and others, which are presumably harder, like the problem of counting cliques of size k. Beside the classes VP and VNP (and subclasses of them), which correspond to time bounded computation in the Boolean world, there have also been definitions of algebraic classes that correspond to space bounded Boolean computation, see [18, 17, 20]. However, we are not aware of any parameterized classes in the algebraic world despite some algorithmic upper bounds being known, see for instance [4, 10].

1.1 Our Contribution

In this paper, we define a theory of parameterized algebraic complexity classes. While some of the definitions are rather obvious modifications of the Boolean ones and some of the basic theorems easily transfer from the Boolean world to the algebraic world, some concepts have to be modified. For instance, we cannot use projections to define hardness in general in the parameterized world, since they can only decrease the degree. On the one hand, one could choose the degree as a parameter, for instance, when computing the vertex cover polynomial. On the other hand, one could have parameterized families where the degree is always n, like the permanent on bounded (orientable[1]) genus graphs. One cannot compare these families with projections, although both of them turn out to be fixed parameter tractable. We could use c-reductions instead (which are the analogue of Turing reductions). However, these seem too powerful. We propose some intermediate concept, namely fpt-substitutions: We may replace the variables of a polynomial by other polynomials that are computed by small circuits (and not simply constants and variables like in the case of projections). This mirrors what is done in parsimonious reductions: the input is transformed by a polynomial time computable function but no post-processing is allowed.

Our main technical contribution is the VW[1]-completeness proof of the parameterized clique polynomial. This proof turns out to be far more complicated than in the Boolean world, since we are not counting satisfying assignments to Boolean circuits but we are computing sums over algebraic circuits. First we prove that one can combine two exponential sums into one sum. While this is very easy in the case of VNP, it turns out to be quite complicated in the case of VW[1]. Then we prove a normal form for so-called weft 1 circuits, the defining circuits for VW[1]. We go on with proving that the components consisting of all monomials that depend on a given number of variables of a polynomial computed by a weft 1 formula can be written as a bounded exponential sum over so-called Boolean-arithmetic expressions. We then show how to reduce such a sum to the clique problem.

In our final section, we study two polynomials based on cycle covers. In the first one, the covers consist of one cycle of length k and all other cycles being self loops. The second polynomial is similar, but allows all other cycles to be of constant length. We prove that the first problem is VW[1] complete while the second problem is hard for VW[t] for all t.

2 Valiant's Classes

We give a brief introduction to Valiant's classes, for further information we refer the reader to [2, 19]. Throughout the whole paper, K will denote the underlying ground field. An arithmetic circuit C is an acyclic directed graph such that every node has either indegree 0 or indegree 2. Nodes with indegree 0 are called input nodes and they are either labeled with a constant from K or with some variable. The other nodes are called computation gates, they are either labeled with $+$ (addition gate) or $*$ (multiplication gate). Every gate computes a polynomial in the obvious way. There is exactly one gate of outdegree 0, the polynomial computed there is the output of C. The size of a circuit is the number of edges in it. The depth of a circuit is length of a longest path from an input node to the output node. Later, we will also look at circuits in which the computation gates can have arbitrary fan-in.

The objects that we study will be polynomials over K in variables $X_1, X_2, \ldots$ (Occasionally, we will rename these variables to make the presentation more readable.) We will denote

[1] We restrict ourselves to study orientable genus in this paper.

$\{X_1, X_2, \dots\}$ by X. The circuit complexity $C(f)$ of a polynomial f is the size of a smallest circuit computing f. We call a function $r\colon \mathbb{N} \to \mathbb{N}$ *p-bounded* if there is a polynomial p such that $r(n) \leq p(n)$ for all n.

▶ **Definition 2.1.** *A sequence of polynomials $(f_n) \in K[X]$ is called a* p-family *if for all n,*
1. *$f_n \in K[X_1, \dots, X_{p(n)}]$ for some p-bounded function p and*
2. *$\deg f_n$ is p-bounded.*

▶ **Definition 2.2.** *The class* VP *consists of all p-families (f_n) such that $C(f_n)$ is p-bounded.*

Let $f \in K[X]$ be a polynomial and $s\colon X \to K[X]$ be a mapping that maps indeterminates to polynomials. Now, s can be extended in a unique way to an algebra endomorphism $K[X] \to K[X]$. We call s a *substitution*. (Think of the variables being replaced by polynomials.)

▶ **Definition 2.3.** 1. *Let $f, g \in K[X]$. f is called a* projection *of g if there is a substitution $r\colon X \to X \cup K$ such that $f = r(g)$. We write $f \leq_p g$ in this case. (Since g is a polynomial, it only depends on a finite number of indeterminates. Therefore, we only need to specify a finite part of r.)*
2. *Let (f_n) and (g_n) be p-families. (f_n) is a* p-projection *of (g_n) if there is a p-bounded function $q\colon \mathbb{N} \to \mathbb{N}$ such that $f_n \leq_p g_{q(n)}$. We write $(f_n) \leq_p (g_n)$.*

Projections are very simple reductions. Therefore, we can also use them to define hardness for "small" complexity classes like VP. More powerful are so-called *c-reductions*, which are an analogue of Turing reductions. c-reductions are strictly more powerful than p-projections [15]. Let g be a polynomial in s variables. A g-oracle gate is a gate of arity s that on input $t_1, \dots, t_s$ outputs $g(t_1, \dots, t_s)$. The size of such a gate is s. $C^g(f)$ denotes the minimum size of a circuit with g-oracle gates that computes f. If G is a set of polynomials, then $C^G(f)$ is the minimum size of an arithmetic circuit that can use any g-oracle gates for any $g \in G$.

▶ **Definition 2.4.** *Let (f_n) and (g_n) be p-families. (f_n) is a* c-reduction *of (g_n) if there is a p-bounded function $q\colon \mathbb{N} \to \mathbb{N}$ such that $C^{G_{q(n)}}(f_n)$ is p-bounded, where $G_{q(n)} = \{g_i \mid i \leq q(n)\}$. We write $(f_n) \leq_c (g_n)$.*

▶ **Definition 2.5.** *A p-family (f_n) is in* VNP*, if there are p-bounded functions p and q and a sequence $(g_n) \in$ VP *of polynomials $g_n \in K[X_1, \dots, X_{p(n)}, Y_1, \dots, Y_{q(n)}]$ such that*

$$f_n = \sum_{e \in \{0,1\}^{q(n)}} g_n(X_1, \dots, X_{p(n)}, e_1, \dots, e_{q(n)}).$$

VP and VNP are algebraic analogues of the classes P and #P in the Boolean world. The permanent family (per_n) is complete for VNP under p-projections (over fields of characteristic distinct from two) and the problem of computing the permanent of a given $\{0,1\}$-matrix is complete for #P under parsimonious reductions.

3 Parameterized (Counting) Complexity

Parameterized counting complexity was introduced by Flum and Grohe [11]. We give a short introduction to fixed parameterized counting complexity. For more information on parameterized complexity, we refer the reader to [12, 9].

▶ **Definition 3.1.** *A* parameterized counting problem *is a function $F\colon \Sigma^* \times \mathbb{N} \to \mathbb{N}$.*

The idea is that an input has two components (x, k), $x \in \Sigma^*$ is the instance and the parameter k measures the "complexity" of the input.

▶ **Definition 3.2.** *A parameterized counting problem is* fixed parameter tractable *if there is an algorithm computing $F(x,k)$ in time $f(k)|x|^c$ for some computable function $f\colon \mathbb{N} \to \mathbb{N}$ and some constant c. The class of all fixed parameter tractable counting problems is denoted by* #FPT.

A parameterized counting problem is fixed-parameter tractable if the running time is polynomial in the instance size. The "combinatorial explosion" is only in the parameter k. In particular, the exponent of n does not depend on k. The classical example for a parameterized counting problem in #FPT is the vertex cover problem: Given a graph G and a natural number k, count all vertex covers of G of size k.

Fixed parameter tractable problems represent the "easy" problems in parameterized complexity. An indication that a problem is not fixed parameter tractable is that it is hard for the class #W[1]. Reductions that are used to define hardness are *parsimonious fpt-reductions*: Such a reduction maps an instance (x,k) to an instance (x',k') such that the value of the two instances is the same, the running time of the reduction is $f(k)|x|^c$ for some computable function f and a constant c, and there is a computable function g such that $k' \leq g(k)$. It is quite easy to see that the composition of two parsimonious fpt-reductions is again a parsimonious fpt-reduction and that #FPT is closed under parsimonious fpt-reductions.

We now define weft t formulas inductively.[2]

▶ **Definition 3.3.** *A* weft 0 *formula is a layered Boolean formula and the gates have fan-in two (over the basis $\wedge$, $\vee$, and $\neg$). A weft t formula is a layered Boolean formula where the gates have fan-in two, except one layer of gates that has unbounded fan-in. This formula has as inputs weft $t-1$ formulas.*

Weft t formulas have t layers of unbounded fan-in gates, and all other gate have bounded fan-in. Weft t formulas are the defining machine model of the #W[t] classes:

▶ **Definition 3.4.** *The class #W[t] are all parameterized counting problems that are reducible by parsimonious fpt-reductions to the following problem: Given a weft t formula C of constant depth and a parameter k, count all satisfying assignments of C that have exactly k 1s.*

A classical example of a counting problem, that is #W[1]-complete, is counting cliques of size k in a graph. Clique is used as a major complete problem for #W[1] by Flum and Grohe [11]. It is known that P = #P implies #FPT = #W[1]. Curticapean [5] proves that counting k-matchings, the parameterized analogue to the permanent, is #W[1]-hard (under Turing fptreductions).

4 Parameterized Valiant's Classes

We now define fixed-parameter variants of Valiant's classes. Our families of polynomials will now have two indices. They will be of the form $(p_{n,k})$. Here, n is the number of indeterminates and k is the parameter.

▶ **Definition 4.1.** *A parameterized p-family is a family $(p_{n,k})$ of polynomials such that*

1. *$p_{n,k} \in K[X_1, \ldots, X_{q(n)}]$ for some p-bounded function q, and*
2. *the degree of $p_{n,k}$ is p-bounded (as a function of $n+k$).*

[2] The term "weft" originates from textile fabrication and has been used in Boolean parameterized complexity from its very beginning.

The most natural parameterization is by the degree: Let (p_n) be any p-family then we get a parameterized family $(p_{n,k})$ by setting $p_{n,k} = H_k(p_n)$. Here $H_k(f)$ denotes the homogeneous part of degree k of some polynomial f.[3] Since $\deg(p_n)$ is polynomially bounded, $p_{n,k}$ is zero when k is large enough. (This will usually be the case for any parameterization.) More generally, we will also allow that $p_{n,k} = H_{t(k)}(p_n)$ for some function t that solely depends on k.

Recall that a vertex cover C of a graph $G = (V, E)$ is a subset of V such that for every edge $e \in E$ at least one endpoint is in C.

▶ **Example 4.2.** Let $\mathcal{G} = (G_n)$ be a family of graphs such that G_n has n nodes. We will assume that the nodes of G_n are $\{1, \dots, n\}$.

1. The vertex cover family $(\mathrm{VC}_n^{\mathcal{G}})$ with respect to $\mathcal{G}$ is defined as

$$\mathrm{VC}_n^{\mathcal{G}} = \sum_{C \subseteq \{1,\dots,n\}} \prod_{i \in C} X_i$$

   where the sum is taken over all vertex covers C of G_n.
2. The parameterized vertex cover family $(\mathrm{VC}_{n,k}^{\mathcal{G}})$, with respect to $\mathcal{G}$, is defined as

$$\mathrm{VC}_{n,k}^{\mathcal{G}} = \sum_{\substack{C \subseteq \{1,\dots,n\} \\ |C|=k}} \prod_{i \in C} X_i$$

where we now sum over all vertex covers of size k of G_n. This is a homogeneous polynomial of degree k. (We will call both families $\mathrm{VC}^{\mathcal{G}}$. There is no danger of confusion, since we mainly deal with the parameterized family.)

Every node has a label X_i and for every vertex cover we enumerate (or more precisely, sum up) its weight, which is the product of the labels of the nodes in it. Above, every graph family defines a particular vertex cover family. We can also define a unifying vertex cover family.

▶ **Example 4.3.** Let $E_{i,j}, X_i$, $1 \leq i < j \leq n$, be variables over some field K. The *parameterized vertex cover polynomial* of size n is defined by

$$\mathrm{VC}_{n,k} = \sum_{\substack{C \subseteq \{1,\dots,n\} \\ |C|=k}} \prod_{\substack{i,j \notin C \\ i<j}} (1 - E_{i,j}) \prod_{i \in C} X_i.$$

The parameterized vertex cover family is defined as $(\mathrm{VC}_{n,k})$.

If we set the variables $E_{i,j}$ to values $e_{i,j} \in \{0, 1\}$ we get the vertex cover polynomial of the graph given by the adjacency matrix $(e_{i,j})$. The first product is 0 if there is an uncovered edge. More generally, if we take a family of graphs $\mathcal{G} = (G_n)$ such that G_n has n nodes and if we plug in the adjacency matrix of G_n into in each $\mathrm{VC}_{n,k}$ then we get the family $(\mathrm{VC}_{n,k}^{\mathcal{G}})$. $(\mathrm{VC}_{n,k}^{\mathcal{G}})$ is parameterized by the degree since we have $\mathrm{VC}_{n,k}^{\mathcal{G}} = H_k(\mathrm{VC}_n^{\mathcal{G}})$. $(\mathrm{VC}_{n,k})$, however, is not parameterized by the degree as $\mathrm{VC}_{n,k}$ contains monomials of degree polynomial in n (independent of k).

Recall that a clique C of a graph is a subset of the vertices such that for every pair of nodes in C there is an edge between them.

[3] I.e., the sum of all monomials of degree k with their coefficients.

▶ **Example 4.4.** **1.** Let $E_{i,j}, X_i$, $1 \leq i, j \leq n$, $i < j$, be variables over some field K. The *clique polynomial* of size n is defined by

$$\text{Clique}_n = \sum_{C \subseteq \{1,\dots,n\}} \prod_{\substack{i,j \in C \\ i<j}} E_{i,j} \prod_{i \in C} X_i.$$

The clique family is defined as (Clique_n).

2. The parameterized clique family $(\text{Clique}_{n,k})$ is defined by

$$\text{Clique}_{n,k} = \sum_{\substack{C \subseteq \{1,\dots,n\} \\ |C|=k}} \prod_{\substack{i,j \in C \\ i<j}} E_{i,j} \prod_{i \in C} X_i.$$

(Again, we will call both families Clique.)

If we set the variables $E_{i,j}$ to values $e_{i,j} \in \{0,1\}$, we get the clique polynomial of the graph given by the adjacency matrix $(e_{i,j})$, since the first product checks whether C is a clique. For each clique, we enumerate a monomial $\prod_{i \in C} X_i$. X_i is the label of the node i. Clique is a polynomial defined on edges and nodes. This seems to be necessary, since the polynomial $\sum_{C \subseteq \{1,\dots,n\}} \prod_{i \in C} X_i = (1 + X_1) \cdots (1 + X_n)$, which is the "node-only" version of clique polynomial of the complete graph, is easy to compute. Therefore, we cannot expect that the "node-only" version of the clique family is hard for some class.

Notice, that the parameterized clique family $(\text{Clique}_{n,k})$ has variables standing in for vertices. These vertices seem to be necessary, as in the counting world, counting the number of k cliques and counting the number of k-independent sets are tightly related. Namely, the number of cliques is the number of independent sets on the complement graph. We want to keep this relationship as the problem is an important member of #W[1] and hence we incorporate the vertices.

$(\text{Clique}_{n,k})$ is parameterized by the degree, since $\text{Clique}_{n,k} = H_{\binom{k}{2}+k}(\text{Clique}_n)$. Here is another example, beside the general vertex cover family, of a family that is parameterized by a different parameter:

▶ **Example 4.5.** Let $\mathcal{G} = (G_{n,k})$ be a family of bipartite graphs such that $G_{n,k}$ has n nodes on both sides and genus k, $k \leq \lceil (n-2)^2/4 \rceil$.[4] Let $A_{n,k}$ be the $n \times n$-matrix that has a variable $X_{i,j}$ in position (i,j) if there is an edge between i and j in $G_{n,k}$ and a 0 otherwise. The *$\mathcal{G}$-parameterized permanent family* $\text{per}^{\mathcal{G}} = (\text{per}^{\mathcal{G}}_{n,k})$ is defined as $\text{per}^{\mathcal{G}}_{n,k} = \text{per}(A_{i,j})$.

There is another natural way to parameterize the permanent:

▶ **Example 4.6.** Given a $k \times n$-matrix $X = (X_{i,j})$ with variables as entries, the rectangular permanent is defined as

$$\text{rper}_{n,k}(X) = \sum_{\substack{f\colon \{1,\dots,k\} \to \{1,\dots,n\} \\ f \text{ is injective}}} \prod_{i=1}^{k} X_{i,f(i)}.$$

When $k = n$ then this is the usual permanent. The *rectangular permanent family* is defined as $\text{rper} = (\text{rper}_{n,k})$.

[4] This is the genus of the $K_{n,n}$ [22].

We will give some more parameterizations of the permanent in Section 8 where we also prove some hardness results.

We now define fixed parameter variants of Valiant's classes.

▶ **Definition 4.7.** 1. *A parameterized p-family* $(p_{n,k})$ *is in the class* VFPT *if* $C(p_{n,k})$ *is bounded by* $f(k)p(n)$ *for some p-bounded function* p *and some arbitrary function* $f\colon \mathbb{N} \to \mathbb{N}$.[5]

2. *The subclass of* VFPT *of all parameterized p-families that are parameterized by the degree is denoted by* VFPT_{deg}.

We will also say above that $C(p_{n,k})$ is *fpt-bounded*. We will see in one of the next sections that the vertex cover family and the bounded genus permanent are in VFPT. We will say that a family of circuits (C_n, k) has *fpt size* if the size is bounded by $f(k)p(n)$ for some function $f\colon \mathbb{N} \to \mathbb{N}$ and p-bounded function p.

▶ **Definition 4.8.** *A parameterized p-family* $f = (f_{n,k})$ *is an* fpt-projection *of another parameterized p-family* $g = (g_{n,k})$ *if there are functions* $r, s, t\colon \mathbb{N} \to \mathbb{N}$ *such that* r *is p-bounded and* $f_{n,k}$ *is a projection of* $g_{r(n)s(k),k'}$ *for some* $k' \leq t(k)$.[6] *We write* $f \leq_p^{fpt} g$.

▶ **Lemma 4.9.** *If* $f \in$ VFPT *(or* VFPT_{deg}*) and* $g \leq_p^{fpt} f$*, then* $g \in$ VFPT *(or* VFPT_{deg}*, respectively).*

▶ **Lemma 4.10.** $\leq_p^{fpt}$ *is transitive.*

One can define a notion of completeness. In the case of fpt-projections, the degree of the polynomial is the only meaningful parameter to consider: The permanent family on bounded genus graphs $\mathrm{per}^{\mathcal{G}}$ is in VFPT and so is (a variant of) the vertex cover family VC. However, every polynomial in the permanent family has degree equal to the number of nodes in the graph (independent of the genus) whereas the degree of the vertex cover polynomial depends on the degree. If a polynomial p is a projection of q, then $\deg p \leq \deg q$. Therefore, $\mathrm{per}^{\mathcal{G}}$ cannot be an fpt-projection of VC. Now we can call a parameterized family f VFPT_{deg}*-complete* (under fpt-projections), if it is in VFPT_{deg} and for all $g \in \mathsf{VFPT}_{deg}$, $g \leq_p^{fpt} f$.

For other parameters, we need a stronger notion of reduction. There are the so-called c-reductions, see [2], which are the analogue of Turing reductions in Valiant's world. This is the strongest kind of reduction one could use. However, the p-projections in Valiant's world seem to be weaker than parsimonious polynomial-time reductions in the Boolean world. Therefore, we propose an intermediate concept, which models parsimonious reductions in the algebraic world. In parsimonious reductions, the input instance is transformed by a polynomial time or fpt computable reduction, then the function we reduce to is evaluated, and the result that we get shall be the value of our given function evaluated at the original instance.

In the algebraic world, this is modeled as follows: We call a p-family $f = (f_n)$ with $f_n \in K[X_1, \ldots, X_{p(n)}]$ a *p-substitution* of a p-family $g = (g_n)$ with $g_n \in K[X_1, \ldots, X_{q(n)}]$ if there is a p-bounded function r, and for all n, there are $h_1, \ldots, h_{q(r(n))}$ such that $f_n = g_{r(n)}(h_1, \ldots, h_{q(r(n))})$ and $\deg(h_i)$[7] as well as $C(h_i)$ is p-bounded for all i. We write $f \leq_s g$.

[5] f need not be computable, since Valiant's model is nonuniform.

[6] k' might depend on n, but its size is bounded by a function in k. There are examples in the Boolean world, where this dependence on n is used.

[7] Note that a polynomial size circuit can construct superpolynomial degree polynomials by repeated squaring

Compared to a projection, we are now allowed to substitute polynomials of p-bounded complexity. We have that $\leq_s$ is transitive and that $p \leq_s q$ and $q \in \mathsf{VP}$ implies $p \in \mathsf{VP}$.

The parameterized analogue is defined as follows.

▶ **Definition 4.11.** *A parameterized p-family $f = (f_{n,k})$ with $f_{n,k} \in K[X_1, \ldots, X_{p(n)}]$ is an* fpt-substitution *of another parameterized p-family $g = (g_{n,k})$ with $g_{n,k} \in K[X_1, \ldots, X_{q(n)}]$ if there are functions $r, s, t \colon \mathbb{N} \to \mathbb{N}$ such that for all n, k, r is p-bounded and there exist polynomials $h_1, \ldots, h_{q(r(n)s(k))} \in K[X_1, \ldots, X_{p(n)}]$ such that*

$$f_{n,k} = g_{r(n)s(k),k'}(h_1, \ldots, h_{q(r(n)s(k))})$$

for some $k' \leq t(k)$ and $\deg(h_i)$ as well as $C(h_i)$ are fpt-bounded (with respect to n and k) for all i. We write $f \leq_s^{fpt} g$.

The proof of the following two lemmas is almost identical to the proofs of Lemmas 4.9 and 4.10. The only difference is that fpt-substitutions do not preserve the degree.

▶ **Lemma 4.12.** *If $f \in \mathsf{VFPT}$ and $g \leq_s^{fpt} f$, then $g \in \mathsf{VFPT}$.*

▶ **Lemma 4.13.** *$\leq_s^{fpt}$ is transitive.*

To define an algebraic analogue of $\#\mathsf{W}[t]$, we study unbounded fan-in arithmetic circuits. These circuits have multiplication and addition gates of arbitrary fan-in. A gate with fan-in 2 will be called a gate of bounded fan-in, any other gate is a gate of unbounded fan-in. (Instead of 2, we can fix any other bound b.)

▶ **Definition 4.14.** *Let C be an arithmetic circuit. The* weft *of C is the maximum number of unbounded fan-in gates on any path from a leaf to the root.*

For $s, k \in \mathbb{N}$, $\binom{s}{k}$ denotes the set of all $\{0,1\}$-vectors of length s having exactly k 1s.

▶ **Definition 4.15.** **1.** *A parameterized p-family $(f_{n,k})$ is in $\mathsf{VW}[t]$, if there is a p-family (g_n) of polynomials $g_n \in K[X_1, \ldots, X_{p(n)}, Y_1, \ldots, Y_{q(n)}]$ with p-bounded p and q such that g_n is computed by a constant depth unbounded fan-in circuit of weft $\leq t$ and polynomial size*[8] *and*

$$(f_{n,k}) \leq_s^{fpt} \Big(\sum_{e \in \binom{q(n)}{k}} g_n(X_1, \ldots, X_{p(n)}, e_1, \ldots, e_{q(n)}) \Big). \tag{1}$$

2. *$\mathsf{VW}_{deg}[t]$ is the subset of all families in $\mathsf{VW}[t]$, that have the degree as the parameter.*

In essence, we emulate the Boolean $\#\mathsf{W}[t]$ definition. Instead of Boolean circuits of weft t we take an arithmetic circuit and instead of counting the number of assignments, we sum over all assignments. In addition, we only count the assignments that have weight k by adjusting the vectors we sum over, namely to $\{0,1\}$-vectors with exactly k ones. While in the Boolean setting the closure is taken with respect to parsimonious fpt-reductions, in the arithmetic setting, we take fpt-substitutions. Hence, our definition seems to be the most appropriate analogue.

The clique family is in $\mathsf{VW}[1]$, since we can write it as

$$\mathrm{Clique}_{n,k} = \sum_{v \in \binom{n}{k}} \prod_{\substack{i,j=1 \\ i<j}}^{n} (E_{i,j} v_i v_j + 1 - v_i v_j) \prod_{i=1}^{n} (X_i v_i + 1 - v_i).$$

[8] Note, that we do not need to require fpt-size, as we use an fpt sized reduction.

This formula has weft 1, since there are two unbounded product gates and none is a predecessor of the other. We replace the product over all C by a product over all vertices and use the v-vectors to switch variables on and off.

Like in the Boolean case, we will show that the parameterized clique family is complete for the class VW[1] (albeit for a stronger notion of reductions, namely fpt-c-reductions). It turns out that this proof is far more complicated than in the Boolean setting, since our circuits can compute arbitrary polynomials and not only Boolean values. Furthermore, multiplication and addition cannot be reduced to each other since there is no analog of de Morgan's law.

▶ **Definition 4.16.** *Let $f = (f_{n,k})$ and $g = (g_{n,k})$ be parameterized p-families. f* fpt-c-reduces *to g if there is a p-bounded function $q\colon \mathbb{N} \to \mathbb{N}$ and functions $s, t\colon \mathbb{N} \to \mathbb{N}$ such that $C^{G_{q(n)s(k),t(k)}}(f_{n,k})$ is fpt-bounded, where $G_{q(n)s(k),t(k)} = \{g_{i,j} \mid i \leq q(n)s(k),\ \ j \leq t(k)\}$. We write $f \leq_c^{fpt} g$.*

The following two lemmas are proved like for $\leq_c$ and VP. We replace oracle gates by circuits and use the fact that fpt-bounded functions are closed under composition.

▶ **Lemma 4.17.** *If $f \in$ VFPT and $g \leq_c^{fpt} f$, then $g \in$ VFPT.*

▶ **Lemma 4.18.** *$\leq_c^{fpt}$ is transitive.*

So we have two different notions to define #W[t]-hardness. Presumably, they are different, see [15].

5 VFPT

▶ **Theorem 5.1.** *For every family of graphs $\mathcal{G} = (G_n)$, where G_n has n nodes, $\mathrm{VC}_{n,k}^{\mathcal{G}}$ is in VFPT_{deg}.*

▶ Remark 5.2. It is unlikely that the general family $\mathrm{VC}_{n,k}$ is in VFPT. Take any graph $G = (V, E)$ on n nodes and m edges and compute $\mathrm{VC}_{n,k}$ on this graph, i.e., set all edge variables to zero that do not occur in E.. Now, for $i < j$, we set

$$E_{i,j} = \begin{cases} 1 - S & \text{if } \{i,j\} \in E, \\ 0 & \text{otherwise,} \end{cases}$$

and $X_i = T$ for all i. Then we get a bivariate polynomial. This polynomial contains a monomial S^iT^j iff there is a vertex cover of size j in G not covering i edges, or, equivalently, covering $m - i$ edges. Note that since the polynomial is now bivariate, we can easily compute its coefficients using interpolation. While the (Boolean decision version of) vertex cover is in FPT, it turns out [14] that the more general question whether there is a set of nodes of size k covering at least t edges is W[1]-hard (with parameter k). Therefore, it seems to be unlikely that $\mathrm{VC}_{n,k}$ has circuits of fpt size.

Mahajan and Saurabh [21] define another variant of the vertex cover polynomial. We multiply each cover by a product over the uncovered edges. They multiply by a product over the covered edges. Both polynomials are essentially equivalently, one can turn one into the other by dividing through the product over all edges, doing a variables transform, and removing divisions.

The *sun graph* $S_{n,k} = (V, E)$ on n nodes is defined as follows: The first k nodes form a clique. And every other node is connected to the nodes $1, \ldots, k$, but to no other nodes, that is, the nodes $k + 1, \ldots, n$ form an independent set. (Note that there are other definitions

of sun graphs in the literature, but all of them look like a sun when drawn appropriately.) Every graph G with n nodes that contains a vertex cover of size k is a subgraph of $S_{n,k}$. To see this, we map the nodes of the vertex cover of G to the nodes of the clique and the remaining nodes of G to the other $n-k$ nodes. Note that there are cannot be any edges in G between the nodes outside of the vertex cover.

We define $\mathrm{VC}^s_{n,k}$ like $\mathrm{VC}_{n,k}$ but on the graph $S_{n,k}$ instead of K_n, i.e., all edge variables not in $S_{n,k}$ are set to zero. The difference to VC is, that we now have some idea where the vertex cover is located (like it is in the Boolean case where we can find a potential set for instance by computing a maximum matching). Therefore, we can obtain:

▶ **Theorem 5.3.** $\mathrm{VC}^s \in \mathsf{VFPT}$.

Both parameterized permanent families turn out to be fixed parameter tractable.

▶ **Theorem 5.4** ([13])**.** *For every family of bipartite graphs $\mathcal{G} = (G_{n,k})$ such that $G_{n,k}$ has n nodes and genus k,* $\mathrm{per}^{\mathcal{G}}$ *is in* VFPT.

▶ **Theorem 5.5** ([25])**.** $\mathrm{rper} \in \mathsf{VFPT}$.

Kernelization is an important concept in parameterized complexity. In the algebraic setting, VFPT can also be characterized by kernels with size only bounded by k.

We also develop a notion of kernelization. We refer the reader to the full version.

6 The VW-hierarchy

We start with proving some basic facts about the $\mathsf{VW}[t]$ classes, in analogy to the Boolean world.

▶ **Lemma 6.1.** $\mathsf{VFPT} = \mathsf{VW}[0]$ *and* $\mathsf{VFPT}_{deg} = \mathsf{VW}_{deg}[0]$.

Proof. The proof is obvious, since $\mathsf{VW}[0]$ and $\mathsf{VW}_{deg}[0]$ are defined as the closure under fpt-substitutions, so we can compute problems in VFPT simply by using the reduction. ◀

The following lemma is obvious.

▶ **Lemma 6.2.** *For every t,* $\mathsf{VW}[t] \subseteq \mathsf{VW}[t+1]$ *and* $\mathsf{VW}_{deg}[t] \subseteq \mathsf{VW}_{deg}[t+1]$.

We call a parameterized p-family f $\mathsf{VW}[t]$*-hard* (under fpt-substitutions), if for all $g \in \mathsf{VW}[t]$, $g \leq^{fpt}_s f$. f is called $\mathsf{VW}[t]$*-complete* (under fpt-substitutions) if in addition, $f \in \mathsf{VW}[t]$. If the same way, we can also define hardness and completeness under fpt-c-reductions.

For the classes $\mathsf{VW}_{deg}[t]$, it is reasonable to study hardness and completeness under fpt-projections. We call a parameterized p-family f $\mathsf{VW}_{deg}[t]$*-hard* (under fpt-projections), if for all $g \in \mathsf{VW}[t]$, $g \leq^{fpt}_p f$. f is called $\mathsf{VW}_{deg}[t]$*-complete* (under fpt-projections) if in addition, $f \in \mathsf{VW}[t]$.

▶ **Lemma 6.3.** *If f is $\mathsf{VW}[t+1]$-complete under fpt-substitutions and $f \in \mathsf{VW}[t]$, then $\mathsf{VW}[t] = \mathsf{VW}[t+1]$. In the same way, if f is $\mathsf{VW}_{deg}[t+1]$-complete under fpt-substitutions or fpt-projections and $f \in \mathsf{VW}_{deg}[t]$, then $\mathsf{VW}_{deg}[t] = \mathsf{VW}_{deg}[t+1]$.*

It is open in the Boolean case whether $\mathsf{W}[t] = \mathsf{W}[t+1]$ or $\#\mathsf{W}[t] = \#\mathsf{W}[t+1]$ implies a collapse of the corresponding hierarchy. Maybe the algebraic setting can provide more insights.

▶ **Theorem 6.4.** *If* VFPT $\neq$ VW[1] *then* VP $\neq$ VNP.

If one takes the defining problems for VW[t] (sums over $\{0,1\}$ vectors with k 1s of weft t circuits) instead of clique, one can prove the same theorem for arbitrary classes VW[t] in place of VW[1]. The proof only gets technically a little more complicated.

7 Hardness of Clique

Our main technical result is the VW[1]-hardness of Clique. The proof is technically much more intricate than in the Boolean setting. We will first give a short outline.

- First, we prove as a technical tool that two bounded exponential sums over a weft t circuit can be expressed by one exponential sum over a (different) weft t circuit. In the case of VNP, a similar proof is easy: Instead of summing over bit vectors of length p and then of length q, we can sum over bit vectors of length $p+q$ instead. If the number of ones is however bounded, this does not work easily anymore. It turns out that for the most interesting class VW[1] of the VW-hierarchy, the construction is astonishingly complicated.
- Next, we prove a normal form for weft 1 circuits. Every weft 1 circuit can be replaced by an equivalent weft 1 circuit that has five layers: The first layer is a bounded summation gate, the second layer consist of bounded multiplication gates, the third layer is the only layer of unbounded gates, the fourth layer again consists of bounded addition gates and the fifth layer of bounded multiplication gates.
- Then we introduce *Boolean-arithmetic formulas*: A Boolean-arithmetic formula is a formula of the form

$$B(X_1, \dots, X_n) \cdot \prod_{i=1}^{n} (R_i X_i + 1 - X_i)$$

  where B is an arithmetization of some Boolean formula and the R_i some polynomial or even rational function (over a different set of variables). For each satisfying $\{0,1\}$-assignment e to B, that is, $B(e) = 1$, the right hand side produces one product and the e_i (assigned to the X_i variables) switch the factors R_i on or off. For a polynomial f, the monomials of support size k are all monomials that depend on exactly k variables. The sum of all these monomials is denoted by $S_k(f)$. A central result for the hardness proof is that when f is computed by a circuit of weft 1, then we can write $S_k(f)$ as a bounded sum over a weft 1 Boolean arithmetic expression, that is, $S_k(f) = \sum_{e \in \binom{p(n,k)}{q(k)}} \mathcal{B}(e) \cdot \prod_{i=1}^{p(n,k)} (R_i e_i + 1 - e_i)$.
- Finally, we prove that Clique is VW[1]-complete under fpt-c-reductions (or under fpt-substitutions that allow rational expressions). Given some bounded sum over a polynomial $g_n(X_1, \dots, X_p, Y_1 \dots, Y_q)$ computed by a weft 1 circuit, we view g_n as a polynomial over the Y-variables, the coefficients of which are polynomials in the X-variables. Then $S_0(g), \dots, S_k(g)$ are the parts of g_n that contribute to the sum when summing over all bit vectors with k ones. We can write this as a bounded sum over a Boolean-arithmetic formula. The concept of Boolean-arithmetic formulas allows us to reuse parts of the Boolean hardness proof.

Note that once we have the VW[1]-hardness of Clique, we get further hardness results for free.

8 Hardness of the Permanent and Cycle Covers

In this section we will highlight the vast difference between the provable complexity of the following two problems. Having cycle covers where one cycle is of length k and all other cycles are self loops and the complexity of all cycle covers where one cycle is length k and all other cycle covers are of length some fixed constant c. As always in this paper, we will look at corresponding polynomials to this problem.

8.1 Hardness of the k-permanent

We are adapting the proof from Curticapean and Marx [8] to show the hardness of the following parameterization of the permanent. Notice that the theorem from [8] is not enough for us. It is in general unclear how and in which way the cycles transfer in the theorem while we need an explicit fpt-projection.

We define the k-permanent polynomial as follows. Let S'_n be the set of all permutations on n elements that map $n-k$ elements to itself. Then

$$\mathrm{per}_k = \sum_{\sigma \in S'_n} \prod_{i \in [n]} x_{i,\sigma(i)}.$$

Notice, we do not include the selected vertices, as all vertices are in the cycle cover and hence the two problems are equivalent.

▶ **Corollary 8.1.** (per_k) *is* VW[1] *hard under fpt-c-reductions.*

8.2 Bounded length Cycle Covers

▶ **Definition 8.2.** *We define* $\mathrm{per}_{\leq c,k}$*, the bounded length* k*-permanent, to be the following polynomial over all cycle covers where one cycle has length* k *and all other cycles have length bounded by some constant* $c \geq 3$*.*

$$\mathrm{per}_{\leq c,k} = \sum_{\sigma \in S''_n} \prod_{i \in [n]} x_{i,\sigma(i)}$$

where S''_n *is the set of permutations* σ *on* n *elements such that* σ *has one cycle of length* k *and all other cycles have length* $\leq c$*.*

With this, we can prove the following theorem.

▶ **Theorem 8.3.** *For all* t*, there exists a constant* c *such that* $\mathrm{per} \leq c, k$ *is hard for* VW[t] *under fpt-c-reductions.*

References

1 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *J. Comput. Syst. Sci.*, 87:119–139, 2017. doi:10.1016/j.jcss.2017.03.003.

2 Peter Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory.* Springer, 2000.

3 Peter Bürgisser. Cook's versus Valiant's hypothesis. *Theor. Comput. Sci.*, 235(1):71–88, 2000. doi:10.1016/S0304-3975(99)00183-8.

4 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics*, 108(1-2):23–52, 2001. doi:10.1016/S0166-218X(00)00221-3.

5 Radu Curticapean. Counting Matchings of Size k Is W[1]-Hard. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 352–363. Springer, 2013. doi:10.1007/978-3-642-39206-1_30.

6 Radu Curticapean, Holger Dell, Fedor V. Fomin, Leslie Ann Goldberg, and John Lapinskas. A Fixed-Parameter Perspective on #BIS. *CoRR*, abs/1702.05543, 2017. arXiv:1702.05543.

7 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 210–223. ACM, 2017. doi:10.1145/3055399.3055502.

8 Radu Curticapean and Dániel Marx. Complexity of Counting Subgraphs: Only the Boundedness of the Vertex-Cover Number Counts. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 130–139. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.22.

9 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

10 Uffe Flarup, Pascal Koiran, and Laurent Lyaudet. On the Expressive Power of Planar Perfect Matching and Permanents of Bounded Treewidth Matrices. In Takeshi Tokuyama, editor, *Algorithms and Computation, 18th International Symposium, ISAAC 2007, Sendai, Japan, December 17-19, 2007, Proceedings*, volume 4835 of *Lecture Notes in Computer Science*, pages 124–136. Springer, 2007. doi:10.1007/978-3-540-77120-3_13.

11 Jörg Flum and Martin Grohe. The Parameterized Complexity of Counting Problems. *SIAM J. Comput.*, 33(4):892–922, 2004. doi:10.1137/S0097539703427203.

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 Anna Galluccio and Martin Loebl. On the Theory of Pfaffian Orientations. I. Perfect Matchings and Permanents. *Electr. J. Comb.*, 6, 1999. URL: http://www.combinatorics.org/Volume_6/Abstracts/v6i1r6.html.

14 Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized Complexity of Vertex Cover Variants. *Theory Comput. Syst.*, 41(3):501–520, 2007. doi:10.1007/s00224-007-1309-3.

15 Christian Ikenmeyer and Stefan Mengel. On the relative power of reduction notions in arithmetic circuit complexity. *Inf. Process. Lett.*, 130:7–10, 2018. doi:10.1016/j.ipl.2017.09.009.

16 Mark Jerrum and Kitty Meeks. The parameterised complexity of counting even and odd induced subgraphs. *Combinatorica*, 37(5):965–990, 2017. doi:10.1007/s00493-016-3338-5.

17 Pascal Koiran and Sylvain Perifel. VPSPACE and a transfer theorem over the complex field. In Ludek Kucera and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Ceský Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*, pages 359–370. Springer, 2007. doi:10.1007/978-3-540-74456-6_33.

18 Pascal Koiran and Sylvain Perifel. VPSPACE and a transfer theorem over the reals. In Wolfgang Thomas and Pascal Weil, editors, *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, volume 4393 of *Lecture Notes in Computer Science*, pages 417–428. Springer, 2007. doi:10.1007/978-3-540-70918-3_36.

19 Meena Mahajan. Algebraic Complexity Classes. *CoRR*, abs/1307.3863, 2013. arXiv:1307.3863.

20 Meena Mahajan and B. V. Raghavendra Rao. Small-Space Analogues of Valiant's Classes. In Miroslaw Kutylowski, Witold Charatonik, and Maciej Gebala, editors, *Fundamentals of Computation Theory, 17th International Symposium, FCT 2009, Wroclaw, Poland, September*

2-4, 2009. Proceedings, volume 5699 of *Lecture Notes in Computer Science*, pages 250–261. Springer, 2009. doi:10.1007/978-3-642-03409-1_23.

21 Meena Mahajan and Nitin Saurabh. Some Complete and Intermediate Polynomials in Algebraic Complexity Theory. In Alexander S. Kulikov and Gerhard J. Woeginger, editors, *Computer Science - Theory and Applications - 11th International Computer Science Symposium in Russia, CSR 2016, St. Petersburg, Russia, June 9-13, 2016, Proceedings*, volume 9691 of *Lecture Notes in Computer Science*, pages 251–265. Springer, 2016. doi:10.1007/978-3-319-34171-2_18.

22 Gerhard Ringel. Das Geschlecht des vollständigen paaren Graphen. *Abh. Math. Sem. Univ. Hamburg 28*, pages 139–150, 1965.

23 Marc Roth. Counting Restricted Homomorphisms via Möbius Inversion over Matroid Lattices. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 63:1–63:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.63.

24 Leslie G. Valiant. Completeness Classes in Algebra. In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 249–261. ACM, 1979. doi:10.1145/800135.804419.

25 Virginia Vassilevska and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 455–464. ACM, 2009. doi:10.1145/1536414.1536477.

Hierarchy of Transportation Network Parameters and Hardness Results

Johannes Blum
University of Konstanz, Germany
johannes.blum@uni-konstanz.de

Abstract

The graph parameters highway dimension and skeleton dimension were introduced to capture the properties of transportation networks. As many important optimization problems like Travelling Salesperson, Steiner Tree or k-Center arise in such networks, it is worthwhile to study them on graphs of bounded highway or skeleton dimension.

We investigate the relationships between mentioned parameters and how they are related to other important graph parameters that have been applied successfully to various optimization problems. We show that the skeleton dimension is incomparable to any of the parameters distance to linear forest, bandwidth, treewidth and highway dimension and hence, it is worthwhile to study mentioned problems also on graphs of bounded skeleton dimension. Moreover, we prove that the skeleton dimension is upper bounded by the max leaf number and that for any graph on at least three vertices there are edge weights such that both parameters are equal.

Then we show that computing the highway dimension according to most recent definition is NP-hard, which answers an open question stated by Feldmann et al. [18]. Finally we prove that on graphs $G = (V, E)$ of skeleton dimension $\mathcal{O}(\log^2 |V|)$ it is NP-hard to approximate the k-Center problem within a factor less than 2.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Problems, reductions and completeness; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Graph Parameters, Skeleton Dimension, Highway Dimension, k-Center

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.4

Related Version A full version of the paper is available at `https://arxiv.org/abs/1905.11166`.

1 Introduction

Many important optimization problems arise in the context of road or flight networks, e.g. Travelling Salesperson or Steiner Tree, and have applications in domains like route planning or logistics. Therefore, several approaches have been developed that try to exploit the special structure of such transportation networks. Examples are the graph parameters highway dimension and skeleton dimension. Intuitively, a graph has low highway dimension hd or skeleton dimension κ, if there is only a limited number of options to leave a certain region of the network on a shortest path. Both parameters were originally used in the analysis of shortest path algorithms and it was shown that if hd or κ are small, there are preprocessing-based techniques to compute shortest paths significantly faster than the algorithm of Dijkstra [3, 2, 1, 24].

The highway dimension was also investigated in the context of NP-hard optimization problems, such as Travelling Salesperson (TSP), Steiner Tree and Facility Location [18], k-Center [17, 20, 10] or k-Median and Bounded-Capacity Vehicle Routing [10]. It was shown that in many cases, graphs of low highway dimensions allow better algorithms than general graphs. To our knowledge, the skeleton dimension has exclusively been studied in the context of shortest path algorithms so far. However, it was shown that real-world road networks exhibit a skeleton dimension that is clearly smaller than the

highway dimension [11]. Moreover, in contrast to the highway dimension, it can be computed in polynomial time. Hence it is natural to study the aforementioned problems on networks of low skeleton dimension.

Further graph classes that have been used to model transportation networks are for instance planar graphs and graphs of low treedwidth or doubling dimension. Moreover, many important optimization problems have been studied extensively for classic graph parameters like treewidth or pathwidth [12, 4]. Still, there are only partial results on how the highway dimension hd and skeleton dimension κ are related to these parameters. This is the starting point of the present paper. A better understanding of the relationships between hd, κ and different well-studied graph parameters will allow a deeper insight in the structure of transportation networks and might enable further algorithms custom-tailored for such networks.

1.1 Related Work

We now briefly sum up some algorithmic results in the context of optimization problems in transportation networks. Arora [5] developed a general framework that enables PTASs for several geometric problems where the network is embedded in the Euclidean plane. Building upon the work of Arora, Talwar [27] developed QPTASs for TSP, Steiner Tree, k-Median and Facility Location on graphs of low *doubling dimension* (for a formal definition, see Definition 2). This was improved by Bartal et al. [6], who obtained a PTAS for TSP. As the skeleton dimension of a graph upper bounds its doubling dimension (cf. Section 2.1) the aforementioned results immediately imply a PTAS for TSP and QPTASs for Steiner Tree, k-Median and Facility Location.

The k-Center problem is NP-complete on general graphs [28] and has been subject to extensive research. In fact, for any $\epsilon > 0$, it is NP-hard to compute a $(2-\epsilon)$-approximation, even when considering only planar graphs [25], geometric graphs using L_1 or L_∞ distances or graphs of highway dimension $\mathcal{O}(\log^2 |V|)$[17]. However, there is a fairly simple 2-approximation algorithm for general graphs by Hochbaum and Shmoys [22].

One way to approximate k-Center better than by a factor of 2 is the use of so called *fixed-parameter approximation algorithms (FPAs)*. The basic idea is to combine the concepts of fixed-parameter algorithms and approximation algorithms. Formally, for $\alpha > 1$, an α-FPA for a parameter p is an algorithm that computes an α-approximation in time $f(p) \cdot n^{\mathcal{O}(1)}$ where f is a computable function. Feldmann [17] showed there is a $3/2$-FPA for k-Center when parameterizing both by the number of center nodes k and the highway dimension hd. Later, Becker et al. [10] showed that for any $\epsilon > 0$ there is a $(1+\epsilon)$-FPA for k-Center when parameterizing by k and hd, using a slightly different definition for the highway dimension as in [17] (see also Section 2.2). Moreover, on graphs of doubling dimension d, it is possible to compute a $(1+\epsilon)$-approximation in time $\left(k^k/\epsilon^{\mathcal{O}(k \cdot d)}\right) \cdot n^{\mathcal{O}(1)}$ [20]. As the doubling dimension is a lower bound for the skeleton dimension κ, this implies a $(1+\epsilon)$-FPA for parameter (ϵ, k, κ). However, computing a $(2-\epsilon)$-approximation is $W[2]$-hard when parameterizing only by k, and unless the exponential time hypothesis (ETH) fails, it is not possible to compute a $(2-\epsilon)$-approximation in time $2^{2^{o(\sqrt{hd})}} \cdot n^{\mathcal{O}(1)}$ for highway dimension hd [17].

1.2 Contributions and Outline

We first give an overview of various graph parameters, in particular we review several slightly different definitions of the highway dimension that can be found in the literature. Then we show relationships between skeleton dimension, highway dimension and other important parameters. Our results include the following.

- The max leaf number ml is a tight upper bound for the bandwidth bw. This improves a result of Sorge et al. who showed that $bw \leq 2ml$ [26].
- The skeleton dimension is incomparable to any of the parameters distance to linear forest, bandwidth, treewidth and highway dimension (when using the definitions from [3] or [2]).
- The skeleton dimension κ is upper bounded by the max leaf number. Moreover, for any graph on at least 3 vertices there are edge weights for which both parameters are equal. As the max leaf number is an upper bound for the pathwidth pw, it follows that $\kappa \geq pw$. This improves a result of Blum and Storandt, who showed that one can choose edge weights for any graph such that the skeleton dimension is at least $(pw-1)/(\log_2 |V|+2)$ [11].

The resulting parameter hierarchy is illustrated in Figure 1. In the second part of the paper we show hardness for two problems in transportation networks.

- We show that computing the highway dimension is NP-hard when using the most recent definition from [1]. This answers an open question stated in [18], where NP-hardness was only shown for the definitions used in [3] and [2].
- We study the k-Center problem in graphs of low skeleton dimension. We extend a result from [17] and show how graphs of low doubling dimension can be embedded into graphs of low skeleton dimension. It follows that for any $\epsilon > 0$ it is NP-hard to compute a $(2-\epsilon)$-approximation on graphs of skeleton dimension $\mathcal{O}(\log^2 |V|)$.

2 Preliminaries

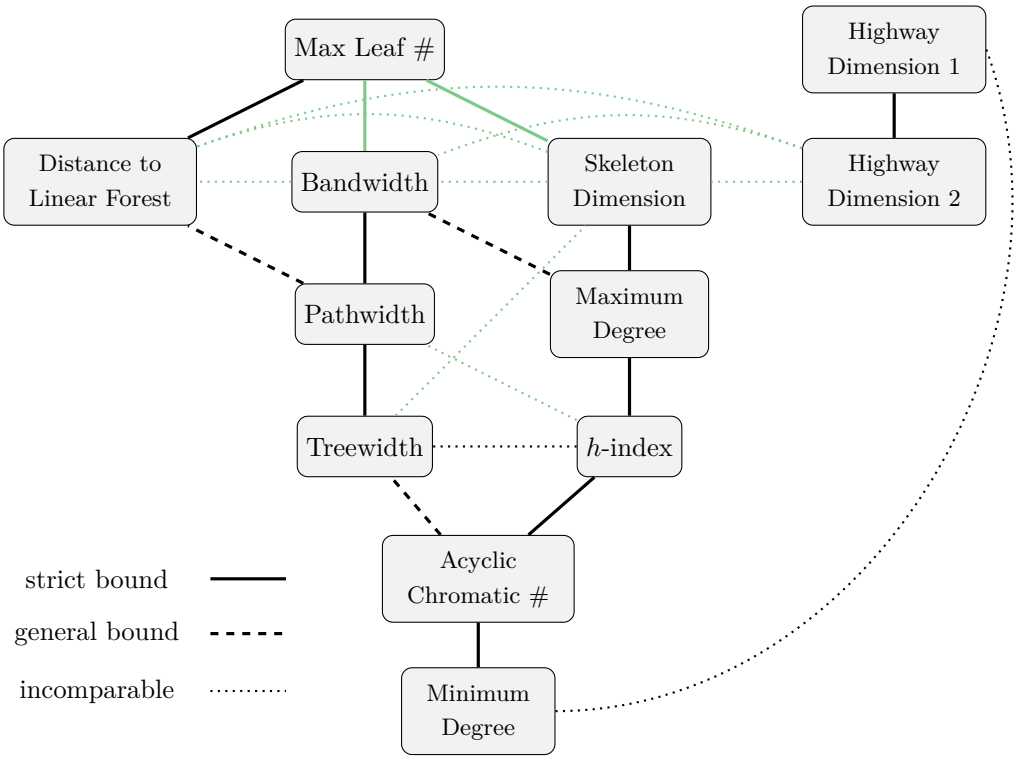
We consider undirected graphs $G = (V, E)$ and denote the number of nodes and edges by n and m, respectively. Let Δ be the maximum degree of G. For weighted graphs, let $\ell\colon E \to \mathbb{Q}^+$ be the cost function. For nodes $u, v \in V$, let $\text{dist}_G(u, v)$ (or simply $\text{dist}(u, v)$) be length of the shortest path from u to v in G. A weighted graph $G = (V, E)$ is metric if (V, dist_G) is a metric, i.e. its edge weights satisfy the triangle inequality, that is for all nodes $u, v, w \in V$ we have $\text{dist}(u, w) \leq \text{dist}(u, v) + \text{dist}(v, w)$. We assume that the shortest path between any two nodes of G is unique, which can be achieved e.g. by slightly perturbing the edge weights. For $u \in V$ and $r \in \mathbb{R}$, we define the ball around the node u of radius r as $B_r(u) = \{v \in V \mid \text{dist}(u, v) \leq r\}$. The length of a path π is denoted by $|\pi|$.

2.1 Skeleton Dimension and Doubling Dimension

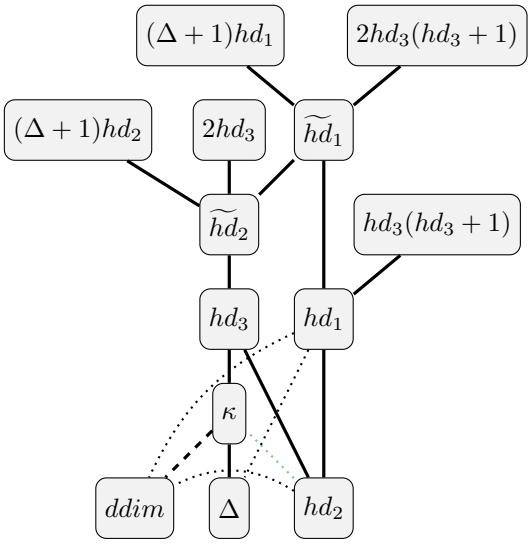
The skeleton dimension was introduced by Kosowski and Viennot to analyze the performance of hub labels, a route planning technique used for road networks [24]. To define it formally, we first need to introduce the geometric realization $\tilde{G} = (\tilde{V}, \tilde{E})$ of a graph $G = (V, E)$ with edge weights ℓ. Intuitively, $\tilde{G}$ is a continuous version of G, where every edge is subdivided into infinitely many infinitely short edges. This means that $V \subseteq \tilde{V}$, for all $u, v \in V$ we have $\text{dist}_{\tilde{G}}(u, v) = \text{dist}_G(u, v)$ and for every edge $\{u, v\}$ of G and every $0 \leq \alpha \leq \ell(\{u, v\})$ there is a node $w \in \tilde{V}$ satisfying $\text{dist}(u, w) = \alpha$ and $\text{dist}(w, v) = \ell(\{u, v\}) - \alpha$.

For a node $s \in V$ let T_s be the shortest path tree of s and let $\tilde{T}_s$ be its geometric realization. Recall that shortest paths are unique, and hence the same holds for T_s and $\tilde{T}_s$. The skeleton T_s^* is defined as the subtree of $\tilde{T}_s$ induced by the nodes $v \in \tilde{V}$ that have a descendant w in $\tilde{T}_s$ satisfying $\text{dist}(v, w) \geq 1/2 \cdot \text{dist}(s, v)$. Intuitively, we obtain T_s^* by taking every shortest path with source s, cutting off the last third of the path and taking the union of the truncated paths. For a radius $r \in \mathbb{R}$ let Cut_s^r be the set of all nodes u in T_s^* satisfying $\text{dist}(s, u) = r$.

▶ **Definition 1** (Skeleton Dimension). *The skeleton dimension κ of a graph G is* $\max_{s,r} |\text{Cut}_s^r|$.



(a) Relationships between general graph parameters.



(b) Relationships between maximum degree Δ, doubling dimension $ddim$, skeleton dimension κ and different highway dimensions.

Figure 1 Relationships between graph parameters. New results are highlighted in green. Solid lines denote strict bounds (e.g. treewidth $\leq$ pathwidth), dashed lines denote general bounds (e.g. pathwidth $\leq$ distance to linear forest + 1). Dotted lines denote incomparabilities.

Intuitively, a graph has low skeleton dimension, if for any starting node s there are only a few main roads that contain the major central part of ever shortest path originating from s. Clearly, the skeleton dimension can be computed in polynomial time by computing the shortest path tree and its skeleton for every node $s \in V$ and determining Cut_s^r for every radius $r \in \mathbb{R}$. On large networks, a naïve implementation is still impracticable, but in [11] it was shown that it is possible to compute κ even for networks with millions of vertices.

Related to the skeleton dimension is the doubling dimension, which was introduced as a generalization of several kinds of metrics, e.g. Euclidean or Manhattan metrics.

▶ **Definition 2** (Doubling Dimension). *A graph G is d-doubling, if for any radius r, any ball of radius r is contained in the union of d balls of radius $r/2$. If d is the smallest such integer, the doubling dimension of G is $\log_2 d$.*

Computing the doubling dimension is NP-hard [21]. Kosowski and Viennot showed that a graph with skeleton dimension κ is $(2\kappa + 1)$ doubling [24].

2.2 Highway Dimension

The highway dimension was introduced by Abraham et al., motivated by the observation of Bast et al. that in road networks, all shortest paths leaving a certain region pass through one of a small number of nodes [7, 8]. In the literature, several slightly different definitions of the highway dimension can be found. The first one was given in [3].

▶ **Definition 3** (Highway Dimension 1). *The highway dimension of a graph G is the smallest integer hd_1 such that for any radius r and any node u there is a hitting set $S \subseteq B_{4r}(u)$ of size hd_1 for the set of all shortest paths π satisfying $|\pi| > r$ and $\pi \subseteq B_{4r}(u)$.*

In [19, 20], a generalized version of hd_1 was used, where balls of radius $c \cdot r$ for $c \geq 4$ were considered. It was observed that the highway dimension is highly sensitive to the chosen radius, i.e. there are graphs of highway dimension 1 w.r.t. radius c and highway dimension of $\Omega(n)$ w.r.t. radius $c' > c$.

In [2] the highway dimension was defined as follows.

▶ **Definition 4** (Highway Dimension 2). *The highway dimension of a graph G is the smallest integer hd_2 such that for any radius r and any node u there is a hitting set $S \subseteq V$ of size hd_2 for the set of all shortest paths π satisfying $2r \geq |\pi| > r$ that intersect $B_{2r}(u)$.*

The definition of hd_1 requires to hit all shortest paths contained in the ball of radius $4r$, while for hd_2 only the shortest paths intersecting the ball of radius $2r$ need to be hit. Hence, we have $hd_2 \leq hd_1$. Abraham et al. motivate their new definition with the fact that a smaller highway dimension can be achieved on real-world instances, while previous results still hold [2]. Both previously defined highway dimensions are incomparable to the maximum degree and the doubling dimension [3].

In [1], a continuous version of the highway dimension hd_2 was introduced, which is based on the geometric realization. For the definition, assume w.l.o.g. that $\ell(e) \geq 1$ for all edges $e \in E$.

▶ **Definition 5** (Continuous Highway Dimension). *The continuous highway dimension of a graph G is the smallest integer $\widetilde{hd}_2$ such that for any radius $r \geq 1$ and any node $u \in \tilde{V}$ of the geometric realization $\tilde{G}$ there is a hitting set $S \subseteq V$ of size $\widetilde{hd}_2$ for the set of all shortest paths π satisfying $2r \geq |\pi| > r$ that intersect $B_{2r}(u)$.*

Clearly, we have $hd_2 \leq \widetilde{hd}_2$. In [24] it was observed that $\widetilde{hd}_2$ is upper bounded by $(\Delta+1)hd_2$. Along the lines of Definition 3, we can also introduce the continuous version $\widetilde{hd}_1$ of hd_1. It holds that $hd_1 \leq \widetilde{hd}_1 \leq (\Delta+1)hd_1$ and moreover $\widetilde{hd}_2 \leq \widetilde{hd}_1$. In [1], yet another definition of the highway dimension was given. It is based on the notion of r-significant shortest paths.

▶ **Definition 6** (r-significant shortest path). *For $r \in \mathbb{R}$, a shortest path $\pi = v_1 \dots v_k$ is r-significant iff it has an r-witness path π', which means that π' is a shortest path satisfying $|\pi'| > r$ and one of the following conditions hold: (i) $\pi' = \pi$, or (ii) $\pi' = v_0\pi$, or $\pi' = \pi v_{k+1}$, or (iv) $\pi' = v_0\pi v_{k+1}$ for nodes $v_0, v_{k+1} \in V$.*

In other words, π is r-significant, if by adding at most one vertex to every end we can obtain a shortest path π' of length more than r (the r-witness). For $r, d \in \mathbb{R}$, a shortest path π is (r,d)-close to a vertex v, if there is an r-witness path π' of π that intersects the ball $B_d(v)$.

▶ **Definition 7** (Highway Dimension 3). *The highway dimension of a graph G is the smallest integer hd_3 such that for any radius r and any node u there is a hitting set $S \subseteq V$ of size hd_3 for the set of all shortest paths π that are $(r, 2r)$-close to u.*

The advantage of the latest definition is that it also captures continuous graphs. In particular, it was shown that $hd_3 \leq \widetilde{hd}_2 \leq 2hd_3$ [1]. Hence there is no need for a continuous version of hd_3, apart from the fact that there is no meaningful notion of an r-witness in a continuous graph.

It can be easily seen that $hd_2 \leq hd_3$ as every shortest path π that is longer than r and intersects $B_{2r}(u)$ is also $(r, 2r)$-close to u (using π itself as the r-witness). Moreover, the skeleton dimension κ is a lower bound for hd_3, i.e. $\kappa \leq hd_3$ [24]. Feldmann et al. showed that $hd_1 \leq hd_3(hd_3+1)$ [18]. Combining their proof with [1] yields that $\widetilde{hd}_1 \leq 2hd_3(hd_3+1)$.

Computing the highway dimensions hd_1 and hd_2 is NP-hard [18]. In Section 4.1 we show that this also holds for hd_3, which answers an open question stated in [18].

2.3 Classic graph parameters

We now provide an overview of several classic graph parameters. They are all defined on unweighted graphs, but we can also apply them to weighted graphs, simply neglecting edge weights. We start with introducing the treewidth and the related parameters pathwidth and bandwidth.

▶ **Definition 8** (Treewidth). *A tree decomposition of a graph $G = (V, E)$ is a tree $T = (\mathcal{X}, \mathcal{E})$ where every node (also called* bag*) $X \in \mathcal{X}$ is a subset of V and the following properties are satisfied: (i) $\bigcup_{X \in \mathcal{X}} X = V$, (ii) for every edge $\{u, v\} \in E$ there is a bag $X \in \mathcal{X}$ containing both u and v, and (iii) for every $u \in V$, the set of all bags containing u induce a connected subtree of T. The width of a tree decomposition $T = (\mathcal{X}, \mathcal{E})$ is the size of the largest bag minus one, i.e. $\max_{X \in \mathcal{X}} (|X| - 1)$. The treewidth tw of a graph $G = (V, E)$ is defined as the minimum width of all tree decompositions of G.*

▶ **Definition 9** (Pathwidth). *A path decomposition of a graph G is a tree decomposition of G that is a path. The pathwidth pw of G is the minimum width of all path decompositions of G.*

It follows directly from the definitions, that the pathwidth is an upper bound for the treewidth and one can show that the minimum degree is a lower bound for the treewidth [26]. The maximum degree Δ is incomparable to both treewidth and pathwidth, as for a square grid graph we have $\Delta = 4$ and $tw \in \Omega(\sqrt{n})$ whereas for a star graph we obtain $\Delta \in \Omega(n)$ and $pw = 1$.

▶ **Definition 10** (Bandwidth). *A vertex labeling of a graph $G = (V, E)$ is a bijection $f\colon V \to \{1, \ldots, n\}$. The bandwidth of G is the minimum of $\max\{|f(u) - f(v)|\colon \{u, v\} \in E\}$, taken over all vertex labelings f of G.*

It was shown that the bandwidth bw is a tight upper bound for the pathwidth [23], and that $\Delta \leq 2 \cdot bw$ [26].

▶ **Definition 11** (Max Leaf Number). *The max leaf number ml of a graph G is the maximum number of leaves of all spanning trees of G.*

▶ **Definition 12** (Distance to Linear Forest). *The distance to linear forest (also known as distance to union of paths) of a graph $G = (V, E)$ is the size of the smallest set $S \subseteq V$ that separates G into a set of disjoint paths.*

▶ **Definition 13** (h-Index). *The h-index of a graph $G = (V, E)$ is the largest integer h such that G has h vertices of degree at least h.*

The max leaf number is closely related to the notion of a connected dominating set. It is an upper bound for several graph parameters. It was shown that for the max leaf number ml and the distance to linear forest dl we have $dl \leq ml - 1$ [15]. We will show that it also upper bounds the bandwidth and the skeleton dimension. For distance to linear forest dl and pathwidth pw it is known that $pw \leq dl + 1$ [13]. Clearly, the h-index is a lower bound for the maximum degree. It was shown that the h-index is incomparable to the treewidth [26].

3 Parameter Relationships

In this section we show relationships between skeleton dimension, highway dimension and other graph parameters. We will see that the max leaf number is an upper bound for the skeleton dimension and the bandwidth, whereas many of the remaining parameters are pairwise incomparable. This shows that they are all useful and worth studying.

3.1 Upper Bounds

We first relate the max leaf number to the skeleton dimension and the bandwidth. We will use the fact, that every tree has as least as many leaves as any subtree.

▶ **Observation 14.** *Let T' be a subtree of a tree T and let L and L' be the leaves of T and T', respectively. Then we have $|L'| \leq |L|$.*

This allows to show that the max leaf number is an upper bound for the skeleton dimension.

▶ **Theorem 15.** *For the skeleton dimension κ and the max leaf number ml we have $\kappa \leq ml$. For any unweighted undirected graph on $n \geq 3$ nodes there are metric edge weights such that $\kappa = ml$.*

Proof. Let $G = (V, E)$ be a graph. Consider the skeleton T_s^* of some node $s \in V$ that has a cut C of size κ. As for any two distinct nodes $u, v \in C$ the lowest common ancestor in T_s^* is distinct from u and v, T_s^* has at least κ leaves. The skeleton T_s^* is a subtree of the shortest path tree T_s of s, so Observation 14 implies that T_s has at least κ leaves. As T_s is a spanning tree of G it follows that $\kappa \leq ml$.

To show that the bound is tight, consider a spanning tree $T = (V, E_T)$ of an unweighted graph $G = (V, E)$ with ml leaves. We choose edge weights ℓ such that the skeleton dimension of the resulting weighted graph equals ml. Let

$$\ell(\{u,v\}) = \begin{cases} 2 & \text{if } \{u,v\} \in E_T \text{ and } u \text{ or } v \text{ is a leaf of } T \\ 1/n & \text{if } \{u,v\} \in E_T \text{ and neither } u \text{ nor } v \text{ is a leaf of } T \\ 5 & \text{else} \end{cases}$$

To examine the skeleton dimension of the resulting graph, consider an internal node s of T. Such a node exists if $n > 2$. We observe that the shortest path tree T_s of s is equal to T as for any vertex v we have $\text{dist}(s,v) < 3$, and hence no edge $e \in E \setminus E_T$ can be contained in T_s. Moreover, for any leaf v we have $\text{dist}(s,v) \geq 2$ and for any internal node v we have $\text{dist}(s,v) < 1$. Consider now the skeleton T_s^*. Any leaf of T_s^* has distance at least $2/3 \cdot 2 > 1$ from s. As T_s^* has ml leaves, the cut of T_s^* at radius $4/3$ has size ml.

Note that in general, the resulting graph is not metric. To fix this, let $\text{dist}_T(u,v)$ be the shortest path distance from u to v when applying the previously chosen edge weights. For $\{u,v\} \in E_T$ we define ℓ as previously, but for $\{u,v\} \notin E_T$ choose $\ell(u,v) = \text{dist}_T(u,v) - \epsilon$ where for every edge, ϵ is chosen from $(0, 1/n^2)$ such that shortest paths are unique. Consider an internal node s of T. The shortest path tree T_s of s may now differ from T, but the number of leaves of T_s is still ml. For any leaf v of T we have now $\text{dist}(s,v) > 2 - n/n^2 \geq 3/2$ and for any internal node v we have $\text{dist}(s,v) < 1$. Hence, the cut of T_s^* at radius 1 has size ml. ◀

As the max leaf number ml is an upper bound for the pathwidth pw, it follows that for any graph G on $n \geq 3$ nodes there are edge weights such that $\kappa \geq pw$. This improves a result of Blum and Storandt, who showed that there are edge weights such that $\kappa \geq (pw-1)/(\log_2 n + 2)$ [11].

Sorge et al. showed that the bandwidth can be upper bounded by two times the max leaf number [26]. We slightly modify their proof to remove the factor of 2 and show that the resulting bound is tight.

▶ **Lemma 16.** *For the max leaf number ml and the bandwidth bw we have $bw \leq ml$. This bound is tight.*

Proof. Let T be a BFS tree of a graph $G = (V, E)$ and let $f\colon V \to \{1, \ldots, n\}$ be a vertex labeling that assigns to every node the time of its BFS discovery. W.l.o.g. we assume that $f(v_i) = i$. Choose an edge $\{v_i, v_j\} \in E$ maximizing $f(v_j) - f(v_i)$. It follows that $bw \leq f(v_j) - f(v_i) = j - i$.

Observe that in the BFS tree T, the node v_i is the parent of v_j as by the choice of $\{v_i, v_j\}$ there is no $k < i$ such that $\{v_k, v_j\} \in E$. Consider the subtree T' of T induced by the nodes $\{v_1, \ldots, v_j\}$. As v_i is the parent of v_j and nodes are ordered by their discovery time, it follows that $v_{i+1}, \ldots, v_j$ are leaves of T'. Observation 14 implies T has at least $(j-i)$ leaves.

Tightness follows from the complete graph K_n where $bw = ml = n - 1$. ◀

3.2 Incomparabilities

We now show incomparabilities between several parameters, which means that they are all worth studying. In [26] it was proven that the treewidth is incomparable to the h-index. We observe that the same holds for the pathwidth.

▶ **Theorem 17.** *The pathwidth and h-index are incomparable.*

Proof. The $\sqrt{n} \times \sqrt{n}$ grid graph has pathwidth $\sqrt{n}$ and h-index at most 4. The caterpillar tree with d backbone vertices of degree d has pathwidth 1 and h-index d. ◀

We proceed with relating the highway dimensions hd_1 and hd_2 to the treewidth and pathwidth. In [19] it was observed that graphs of low highway dimension hd_1 do not have bounded treewidth, as the complete graph on vertex set $\{1, \ldots, n\}$ with edge weights $\ell(\{i, j\}) = 4^{\max(i,j)}$ has highway dimension $hd_1 = 1$ and treewidth $n - 1$.[1] The complete graph K_n has indeed a minimum degree of $n - 1$, which is a lower bound for the treewidth. On the other hand, there are graphs of constant bandwidth and a linear highway dimension hd_2. For instance, consider a complete caterpillar tree on b backbone vertices of degree 3. Its bandwidth is 2. Choose the weight of an edge as $1/n$ if it is a backbone edge and as 1 otherwise. Every edge of weight 1 is a shortest path intersecting the ball of radius 1 around some fixed backbone vertex and hence $hd_2 \geq b = n/2 - 2$. This gives us the follows theorem.

▶ **Theorem 18.** *The highway dimensions hd_1 and hd_2 are incomparable to the bandwidth and the minimum degree.*

We would also like to relate the skeleton dimension to bandwidth and treewidth. On general graphs, it is easy to show, that the skeleton dimension is incomparable to the other two parameters. For instance, a star graph has treewidth 1 and linear skeleton dimension, whereas a complete graph has linear treewidth, but we can choose edge weights such that the shortest path tree of every vertex becomes a path which implies a constant skeleton dimension. However, by choosing such weights for the latter graph, most edges become useless as they do not represent a shortest path and removing all unnecessary edges produces a graph of low treewidth. Still, we can show, that even on metric graphs the skeleton dimension is incomparable to both bandwidth and treewidth.

▶ **Theorem 19.** *On metric graphs the skeleton dimension and the bandwidth are incomparable.*

Proof. Consider the complete caterpillar tree on b backbone vertices of degree 3. It has a bandwidth of 2. Set the weight of every backbone edge to 1 and pick an arbitrary backbone vertex v. For the remaining edges, choose edge weights such that all leaves have the same distance $d \geq 2$ from v. It follows that the skeleton dimension of the weighted caterpillar tree equals the number of leaves which is $b + 2 = n/2 + 1$.

The complete binary tree B_{2d+1} of depth $2d+1$ has pathwidth d [14]. Choosing $\ell(\{u, v\}) = 3^{-j}$ for j and $(j + 1)$ being the depth of u and v in the tree, respectively, yields a graph of skeleton dimension at most 3. ◀

▶ **Theorem 20.** *On metric graphs the skeleton dimension and the treewidth are incomparable.*

Proof. The star graph S_n on n vertices has treewidth 1 and skeleton dimension $n - 1$.

We now construct a graph of treewidth $\Omega(\sqrt{n})$ and constant skeleton dimension. Consider a square grid graph G on the vertex set $V = \{v_1, \ldots, v_n\}$. Subdivide every edge $\{u, v\}$ by inserting two vertices x_{uv} and y_{uv}, i.e. replace the edge $\{u, v\}$ through a path $u\, x_{uv}\, y_{uv}\, v$. Connect the vertices $v_1, \ldots, v_n$ through a path P and denote the resulting graph by $G' = (V', E')$. The original grid graph G has treewidth $\sqrt{n}$ and is a minor of G'. Hence, G' has treewidth $\Omega(\sqrt{n})$.

[1] The edge weights chosen in [19] are actually $\ell(\{i, j\}) = 4^{\min(i,j)}$, which results in a non-metric graph. Removing all edges that are not a shortest path yields a star graph of treewidth 1.

We now choose edges weights for G' resulting in a constant skeleton dimension. For every edge e that is part of the path P, let $\ell(e) = 1$. Consider an edge $\{u, v\}$ of G that was replaced by the path $u\, x_{uv}\, y_{uv}\, v$ and denote the shortest path distance between u and v on the path P by $\text{dist}_P(u, v)$. We choose $\ell(\{u, x_{uv}\}) = \ell(\{y_{uv}, v\}) = 1$ and $\ell(\{x_{uv}, y_{uv}\}) = \text{dist}_P(u, v) + 1/2$. It is easy to verify that the resulting graph is metric.

To bound the skeleton dimension, we use the following claim: For every edge $\{u, v\}$ of G, neither of the shortest paths from u to x_{uv} or from v to y_{uv} contains the edge $\{x_{uv}, y_{uv}\}$. To prove the claim, observe that by concatenating the subpath of P between u and v and the edge $\{v, y_{uv}\}$, we obtain a path of length $\text{dist}_P(u, v) + 1$. Any path from u to y_{uv} containing the edge $\{x_{uv}, y_{uv}\}$ has length $\text{dist}_P(u, v) + 3/2$. The case of v and x_{uv} is symmetric.

It follows that in G' the shortest path tree of a vertex s cannot contain the edge $\{x_{uv}, y_{uv}\}$ unless $s \in \{x_{uv}, y_{uv}\}$, as any subpath of a shortest path must be a shortest path itself. Consider the shortest path tree T_s of some vertex $s \in V$. The previous claim implies that T_s is a caterpillar tree where P is the backbone path. Moreover, T_s ha maximum degree $\Delta \leq 6$ and all edges have unit length. Let $r > 0$ and consider the set Cut_s^r. For $r \leq 1$, the set Cut_s^r intersects only edges incident to s and hence $|\text{Cut}_s^r| \leq 6$. For $1 < r \leq 2$, the set Cut_s^r intersects only edges incident to the two neighbors of s on P, which implies $|\text{Cut}_s^r| \leq 10$. Finally, for $r > 2$ we have $|\text{Cut}_s^r| \leq 2$ because for any vertex $v \notin \tilde{P}$, the distance to its furthest descendant is less than $1 < r/2$ and hence, the set Cut_s^r intersects only edges from the path P. Similarly, it can be shown that $|\text{Cut}_s^r| \leq 6$ if $s \notin V$ (i.e. $s = x_{uv}$ or $s = y_{uv}$). It follows that the skeleton dimension of G' is $\kappa \leq 10$. ◀

So far, it was only known that there can be an exponential gap between skeleton and highway dimension [24]. However, we can use the graph G' from the previous proof to show that the skeleton dimension and the highway dimensions hd_1 and hd_2 are incomparable. Let $\{v^{1,1}, \ldots, v^{q,q}\}$ be the vertex set of the original grid graph and choose the path P used in the construction of G' as $v^{1,1} \ldots v^{1,q}\, v^{2,1} \ldots v^{2,q} \ldots v^{q,1} \ldots v^{q,q}$. In the resulting graph G', for $i \in \{1, \ldots, q\}$, the shortest path from $v_{1,i}$ to $v_{2,i}$ has length q and hence the edge $e_i = \{x_{v^{1,i},v^{2,i}}, y_{v^{1,i},v^{2,i}}\}$ has length $q + \frac{1}{2}$. As any edge of $\{e_1, \ldots, e_q\}$ intersects the ball around $v^{1,1}$ of radius $2q$ and no two of this edges share a common vertex, the highway dimension hd_2 of G' is at least $q = \sqrt{n}$. The star graph on n vertices with unit edge weights has a skeleton dimension of $n - 1$ and a highway dimension hd_1 of 1, so we obtain the following corollary.

▶ **Corollary 21.** *The skeleton dimension is incomparable to both highway dimensions hd_1 and hd_2.*

Finally it can be shown that the distance to linear forest dl is incomparable to the bandwidth bw, the skeleton dimension κ and the highway dimensions hd_1 and hd_2. For instance, a caterpillar tree of constant maximum degree has a distance to linear forest of $\Omega(n)$, but constant bandwidth, skeleton dimension and highway dimensions (for suitably chosen edge weights), whereas there are star-like graphs for which $dl \in \mathcal{O}(1)$ and $bw, \kappa, hd_1, hd_2 \in \Omega(n)$.

▶ **Theorem 22.** *The distance to linear forest is incomparable to the bandwidth, the skeleton dimension and the highway dimensions hd_1 and hd_2.*

Proof. We will use the fact that the caterpillar tree $\mathcal{C}_b$ on b backbone vertices of degree 3 has a distance to linear forest of $b = n/2 - 1$

Bandwidth. The caterpillar $\mathcal{C}_b$ has bandwidth 2. The star graph S_n on n vertices has a bandwidth of $\lfloor n/2 \rfloor$ and a distance to linear forest of 1.

Skeleton dimension. Consider the caterpillar $\mathcal{C}_b$ and choose the weight of an edge $\{u, v\}$ as 2 if u and v are both backbone vertices and as 1 otherwise. The skeleton T_s^* of any vertex s contains exactly one vertex of degree 3 (the backbone vertex that is closest to s) and no vertex of degree more than 3. Hence, the skeleton dimension is 3. The star graph S_n on n vertices with unit edge weights has a skeleton dimension of $n-1$ and a distance to linear forest of 1.

Highway dimensions. Consider the caterpillar $\mathcal{C}_b$ and choose the weight of an edge $\{u, v\}$ as 5 if u and v are both backbone vertices and as 1 otherwise. To bound the highway dimension hd_1, consider some node v and let $r > 0$. Consider a maximum path $P \subseteq B_{4r}(v)$ containing only backbone vertices. It holds that $|P| \leq 8r$. We can greedily choose a set $S \subseteq P$ such that $|S| \leq 7$ and any subpath π of P of length $|\pi| \geq r-2$ is hit by S. Consider a path $\pi' \subseteq B_{4r}(v)$ that is not hit by S. The path π' contains at most two edges of length 1 incident to a leaf and a subpath of P that has length less than $r-2$. Hence, π' has length at most r. It follows that for any $v \in V$ and any $r > 0$ we can hit all shortest paths π satisfying $|\pi| > r$ and $\pi \subseteq B_{4r}(v)$ with at most 7 vertices, which means that $hd_1 \leq 7$.

Take a star graph with l leaves, subdivide every edge by inserting one vertex and choose the weight of every edge in the resulting graph as 1. We obtain a graph of distance to linear forest 1 and highway dimension $hd_2 = l$, as every edge incident to a leaf is a shortest path of length 1 intersecting the ball of radius 1 around the central vertex. ◀

4 Hardness Results

In this section we show hardness for two problems in transportation networks. We first show that computing the highway dimension in NP-hard, even when using the most recent definition. Then we consider the k-Center problem and show that for any $\epsilon > 0$, computing a $(2-\epsilon)$-approximation is NP-hard on graphs of skeleton dimension $\mathcal{O}(\log^2 n)$.

4.1 Highway Dimension Computation

In [18] it was shown that computing the highway dimension hd_1 is NP-hard. The presented reduction is from Vertex Cover and also works for hd_2. It does not directly carry over to hd_3 as the constructed graph has maximum degree $\Delta = n-1$ and we have $hd_3 \geq \Delta$. Still, using a slightly different reduction, we can show NP-hardness for the computation of hd_3.

▶ **Theorem 23.** *Computing the highway dimension hd_3 is NP-hard.*

Proof. We present a reduction from Vertex Cover on graphs with maximum degree $\Delta \leq 3$. Consider therefore a graph $G = (V, E)$ on n nodes satisfying $\Delta \leq 3$. We construct a weighted graph $G' = (V', E')$ as follows. Add a single node x and for any node $v \in V$, add a new node v^* and the edges $\{v, v^*\}$ and $\{v^*, x\}$. For an edge $e \in E'$ choose edge weight $\ell(e) = 5$ if e is incident to x and $\ell(e) = 1$ otherwise.

Let C be a minimum vertex cover of G. We may assume that $|C| > ((\Delta+1)^6-1)/\Delta \in \mathcal{O}(1)$ as for any constant c we can decide in polynomial time whether G has a minimum vertex cover of size c. We show that G' has highway dimension $hd_3 = |C| + n + 1$. Observe that hd_3 is still linear in n, but it may vary between $n+1$ and $2n$, depending on $|C|$.

Let $0 < r < 5/2$. Consider a node $u \in V'$. Let N be the closed neighborhood of the ball around u of radius $2r$, i.e. $v \in N$ iff $v \in B_{2r}(u)$ or v is adjacent to a node $w \in B_{2r}(u)$. Clearly, N is a hitting set for all shortest paths that are $(r, 2r)$-close to u. For $u \neq x$, the ball $B_{2r}(u)$ contains at most $\sum_{i=0}^{4}(\Delta+1)^i$ nodes, as $2r < 5$. Moreover, every node in $B_{2r}(u)$ has at most $\Delta+1$ neighbors. Hence, N is a hitting set of size $\sum_{i=0}^{5}(\Delta+1)^i = ((\Delta+1)^6-1)/\Delta$

for all shortest paths that are $(r, 2r)$-close to u. For $u = x$, we have $N = V' \setminus V$ and therefore $|N| = n + 1$.

Let $r = 5/2$. The ball around x of radius $2r = 5$ is $B_{2r}(x) = V' \setminus V$. Any edge $\{u, v\} \in E$ is $(r, 2r)$-close to x, as $u^* u v v^*$ is an r-witness. Moreover, any node $u \in V' \setminus V$ is a shortest path that is $(r, 2r)$-close to x. However, a single node $u \in V$ is not r-significant, as it can only be extended to a witness of length $2 < r$. Hence, a shortest path π is $(r, 2r)$-close to x iff and only if $\pi \in E$ or $\pi \in V' \setminus V$. Consider a smallest hitting set $H \subseteq V$ for all shortest paths that are $(r, 2r)$-close to x. We have $(V' \setminus V) \subseteq H$, we H needs to hit all paths that consist of one single node $v \in V' \setminus V$. Moreover, H needs to hit all edges E. In other words, H consists of $V' \setminus V$ and a vertex cover for G. Hence, the hitting set H has size $|V' \setminus V| + |C| = |C| + n + 1$.

Observe that for $r = 5/2$, any r-significant shortest path in G' is $(r, 2r)$-close to x, as any node of G' has a neighbor contained in $B_{2r}(x)$. Hence, for any node $u \in V'$, there is a hitting set for all shortest paths that are $(r, 2r)$-close to u of size at most $|C| + n + 1$. Moreover, for any node u and any $r > 5/2$, a shortest path can only be $(r, 2r)$-close to u, if it is also $(5/2, 5)$-close to u. Hence, for any $u \in V'$ and any $r > 5/2$, for all shortest paths that are $(r, 2r)$-close to u there is a hitting set of size at most $|C| + n + 1$.

We conclude that the highway dimension of G' is $hd_3 = |C| + n + 1$ if and only if G has a minimum vertex cover of size $|C|$. ◀

4.2 Hardness of Approximating k-Center

In the k-Center problem, we are given a graph $G = (V, E)$ with positive edge weights and the goal is to select k center nodes $C \subseteq V$ while minimizing $\max_{u \in V} \min_{v \in C} \operatorname{dist}(u, v)$, that is the maximum distance from any node to the closest center node. A possible scenario is that one wants to place a limited number of hospitals on a map such that the maximum distance from any point to the closest hospital is minimized.

We will prove that computing a $(2 - \epsilon)$-approximation on graphs with low skeleton dimension is NP-hard. For that purpose, we first show the following lemma, which is a non-trivial extension of a result of Feldmann [17]. The aspect ratio of a metric $(X, \operatorname{dist}_X)$ is the ratio of the maximum distance between any pair of vertices in X and the minimum distance.

▶ **Lemma 24.** *Let $(X, \operatorname{dist}_X)$ be a metric of constant doubling dimension d and aspect ratio α. For any $0 < \epsilon < 1$ it is possible to compute a graph $G = (X, E)$ in polynomial time that has the following properties:*

(a) *for all $u, v \in X$ we have $\operatorname{dist}_X(u, v) \leq \operatorname{dist}_G(u, v) \leq (1 + \epsilon) \operatorname{dist}_X(u, v)$,*
(b) *the graph G has highway dimension $hd_2 \in \mathcal{O}((\log(\alpha)/\epsilon)^d)$, and*
(c) *the graph G has skeleton dimension $\kappa \in \mathcal{O}((\log(\alpha)/\epsilon)^d)$,*

Proof (Sketch). In [17] it was shown, how to compute a graph H that satisfies properties a and b. This was done by choosing so called *hub sets* $Y_i \subseteq X$ for $i = 0, \ldots, \lceil \log_2 \alpha \rceil$ such that in H any shortest path in the range $(2^i, 2^{i+1}]$ contains some node from Y_i. The hub sets form a hierarchy, i.e. $Y_i \supseteq Y_j$ for all $i < j$. The computed graph has $\binom{|X|}{2}$ edges and the weight of every edge $\{u, v\}$ depends on the hub sets containing u and v. However, we can show that many of these edges are not a shortest path and can be removed. Using properties of the hub sets Y_i and the fact that the metric has doubling dimension d and aspect ratio α, a thorough investigation yields that the skeleton dimension of the resulting graph is bounded by $\mathcal{O}((\log(\alpha)/\epsilon)^d)$. ◀

Feldmann [17] observed that due to a result of Feder and Greene [16], it is NP-hard for any $\epsilon > 0$ to compute a $(2 - \epsilon)$-approximation for k-Center on graphs of doubling dimension

4 and aspect ratio at most n. Lemma 24 hence implies that it is NP-hard to compute a $(2-\epsilon)$-approximation if the skeleton dimension is in $\mathcal{O}(\log^2 n)$. It remains open whether this also holds for $\kappa \in o(\log^2 n)$ and in particular for constant skeleton dimension.

It was also shown, that under the exponential time hypothesis (ETH) it is not possible to compute a $(2-\epsilon)$-approximation for k-Center on graphs of highway dimension hd_2 in time $2^{2^{o(\sqrt{hd_2})}} \cdot n^{\mathcal{O}(1)}$ [17]. Analogously, Lemma 24 implies a bound of $2^{2^{o(\sqrt{\kappa})}} \cdot n^{\mathcal{O}(1)}$ for skeleton dimension κ. We summarize our findings in the following theorem.

▶ **Theorem 25.** *For any $\epsilon > 0$, it is NP-hard to compute a $(2-\epsilon)$-approximation for the k-Center problem on graphs of skeleton dimension $\kappa \in \mathcal{O}(\log^2 n)$. Assuming ETH there is no $2^{2^{o(\sqrt{\kappa})}} \cdot n^{\mathcal{O}(1)}$ time algorithm that computes a $(2-\epsilon)$-approximation.*

5 Conclusion and Future Work

We showed that the skeleton dimension, the highway dimension (when defined as in [3] or [2]) and several other graph parameters are pairwise incomparable. Nevertheless, the skeleton dimension is upper bounded by the max leaf number and lower bounded through the maximum degree and the doubling dimension.

However, for the highway dimensions hd_1 and hd_2 there are still no tight upper or lower bounds. Using a grid graph and a complete graph, it can be shown that they are not even comparable to the minimum degree or the maximum clique size, which are lower bounds for a large number of graph parameters. Bauer et al. showed, that for any unweighted graph there are edge weights such that $hd_2 \geq (pw-1)/(\log_{3/2}|V|+2)$ where pw is the pathwidth [9]. It remains open whether this bound is tight.

It turned out that computing the highway dimension is NP-hard for all three different definitions used in the literature. Still, knowing the highway dimension of real-world networks will give further insight in the structure of transportation networks and hence it is worthwhile to study whether there are FPT algorithms to compute the highway dimension and to what extent it can be approximated.

We proved that on graphs of skeleton dimension $\mathcal{O}(\log^2 n)$ it is not possible to beat the well-known 2-approximation algorithm by Hochbaum and Shmoys for k-Center. Yet, the experimental results reported in [11] indicate that the skeleton dimension of real-world networks might actually be a constant independent of the size of the network. This raises the question whether there is a $(2-\epsilon)$-approximation algorithm for graphs of constant skeleton dimension.

References

1 Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. Highway Dimension and Provably Efficient Shortest Path Algorithms. *J. ACM*, 63(5):41:1–41:26, 2016. doi:10.1145/2985473.

2 Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato Fonseca F. Werneck. VC-Dimension and Shortest Path Algorithms. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 690–699, 2011. doi:10.1007/978-3-642-22006-7_58.

3 Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato Fonseca F. Werneck. Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 782–793, 2010. doi:10.1137/1.9781611973075.64.

4 Stefan Arnborg. Efficient Algorithms for Combinatorial Problems with Bounded Decomposability - A Survey. *BIT*, 25(1):2–23, 1985.

5 Sanjeev Arora. Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and other Geometric Problems. *J. ACM*, 45(5):753–782, 1998. doi:10.1145/290179.290180.

6 Yair Bartal, Lee-Ad Gottlieb, and Robert Krauthgamer. The Traveling Salesman Problem: Low-Dimensionality Implies a Polynomial Time Approximation Scheme. *SIAM J. Comput.*, 45(4):1563–1581, 2016. doi:10.1137/130913328.

7 H. Bast, Stefan Funke, and Domagoj Matijevic. Ultrafast Shortest-Path Queries via Transit Nodes. In *The Shortest Path Problem, Proceedings of a DIMACS Workshop*, volume 74 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 175–192. DIMACS/AMS, 2006. doi:10.1090/dimacs/074/07.

8 H. Bast, Stefan Funke, Domagoj Matijevic, Peter Sanders, and Dominik Schultes. In Transit to Constant Time Shortest-Path Queries in Road Networks. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2007. doi:10.1137/1.9781611972870.5.

9 Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. Search-space size in contraction hierarchies. *Theor. Comput. Sci.*, 645:112–127, 2016. doi:10.1016/j.tcs.2016.07.003.

10 Amariah Becker, Philip N. Klein, and David Saulpic. Polynomial-Time Approximation Schemes for k-center, k-median, and Capacitated Vehicle Routing in Bounded Highway Dimension. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *Proceedings of the 26th Annual European Symposium on Algorithms (ESA)*, volume 112 of *LIPIcs*, pages 8:1–8:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.8.

11 Johannes Blum and Sabine Storandt. Computation and Growth of Road Network Dimensions. In Lusheng Wang and Daming Zhu, editors, *Proceedings of the 24th International Computing and Combinatorics Conference (COCOON)*, volume 10976 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2018. doi:10.1007/978-3-319-94776-1_20.

12 Hans L. Bodlaender. Dynamic Programming on Graphs with Bounded Treewidth. In Timo Lepistö and Arto Salomaa, editors, *Proceedings of the 15th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 317 of *Lecture Notes in Computer Science*, pages 105–118. Springer, 1988. doi:10.1007/3-540-19488-6_110.

13 Hans L. Bodlaender. A Partial k-Arboretum of Graphs with Bounded Treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

14 Kevin Cattell, Michael J. Dinneen, and Michael R. Fellows. A Simple Linear-Time Algorithm for Finding Path-Decompositions of Small Width. *Inf. Process. Lett.*, 57(4):197–203, 1996. doi:10.1016/0020-0190(95)00190-5.

15 Ermelinda DeLaViña and Bill Waller. A note on a conjecture of Hansen et al. Unpublished manuscript, 2009. URL: http://cms.dt.uh.edu/faculty/delavinae/research/DelavinaWaller2009.pdf.

16 Tomás Feder and Daniel H. Greene. Optimal Algorithms for Approximate Clustering. In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 434–444. ACM, 1988. doi:10.1145/62212.62255.

17 Andreas Emil Feldmann. Fixed-Parameter Approximations for k-Center Problems in Low Highway Dimension Graphs. *Algorithmica*, 81(3):1031–1052, 2019. doi:10.1007/s00453-018-0455-0.

18 Andreas Emil Feldmann, Wai Shing Fung, Jochen Könemann, and Ian Post. A $(1 + \epsilon)$-Embedding of Low Highway Dimension Graphs into Bounded Treewidth Graphs. *CoRR*, abs/1502.04588, 2015. arXiv:1502.04588.

19 Andreas Emil Feldmann, Wai Shing Fung, Jochen Könemann, and Ian Post. A $(1+\epsilon)$-Embedding of Low Highway Dimension Graphs into Bounded Treewidth Graphs. *SIAM J. Comput.*, 47(4):1667–1704, 2018. doi:10.1137/16M1067196.

20 Andreas Emil Feldmann and Dániel Marx. The Parameterized Hardness of the k-Center Problem in Transportation Networks. In David Eppstein, editor, *Proceedings of the 16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, volume 101 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.SWAT.2018.19.

21 Lee-Ad Gottlieb and Robert Krauthgamer. Proximity Algorithms for Nearly Doubling Spaces. *SIAM J. Discrete Math.*, 27(4):1759–1769, 2013. doi:10.1137/120874242.

22 Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33(3):533–550, 1986. doi:10.1145/5925.5933.

23 Haim Kaplan and Ron Shamir. Pathwidth, Bandwidth, and Completion Problems to Proper Interval Graphs with Small Cliques. *SIAM J. Comput.*, 25(3):540–561, 1996. doi:10.1137/S0097539793258143.

24 Adrian Kosowski and Laurent Viennot. Beyond Highway Dimension: Small Distance Labels Using Tree Skeletons. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1462–1478. SIAM, 2017.

25 Jan Plesník. On the computational complexity of centers locating in a graph. *Aplikace matematiky*, 25(6):445–452, 1980. URL: https://dml.cz/bitstream/handle/10338.dmlcz/103883/AplMat_25-1980-6_8.pdf.

26 Manuel Sorge, Matthias Weller, Florent Foucaud, Ondřej Suchý, Pascal Ochem, Martin Vatshelle, and Gerhard J. Woeginger. The Graph Parameter Hierarchy. Unpublished manuscript, 2019. URL: https://manyu.pro/assets/parameter-hierarchy.pdf.

27 Kunal Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (SODA)*, pages 281–290. ACM, 2004. doi:10.1145/1007352.1007399.

28 Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001. URL: http://www.springer.com/computer/theoretical+computer+science/book/978-3-540-65367-7.

Metric Dimension Parameterized by Treewidth

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

Nidhi Purohit
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France
nidhi.purohit@ens-lyon.fr

Abstract

A resolving set S of a graph G is a subset of its vertices such that no two vertices of G have the same distance vector to S. The Metric Dimension problem asks for a resolving set of minimum size, and in its decision form, a resolving set of size at most some specified integer. This problem is NP-complete, and remains so in very restricted classes of graphs. It is also W[2]-complete with respect to the size of the solution. Metric Dimension has proven elusive on graphs of bounded treewidth. On the algorithmic side, a polytime algorithm is known for trees, and even for outerplanar graphs, but the general case of treewidth at most two is open. On the complexity side, no parameterized hardness is known. This has led several papers on the topic to ask for the parameterized complexity of Metric Dimension with respect to treewidth.

We provide a first answer to the question. We show that Metric Dimension parameterized by the treewidth of the input graph is W[1]-hard. More refinedly we prove that, unless the Exponential Time Hypothesis fails, there is no algorithm solving Metric Dimension in time $f(\mathrm{pw})n^{o(\mathrm{pw})}$ on n-vertex graphs of constant degree, with pw the pathwidth of the input graph, and f any computable function. This is in stark contrast with an FPT algorithm of Belmonte et al. [SIAM J. Discrete Math. '17] with respect to the combined parameter $\mathrm{tl} + \Delta$, where tl is the tree-length and Δ the maximum-degree of the input graph.

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

Keywords and phrases Metric Dimension, Treewidth, Parameterized Hardness

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.5

Related Version A full version of the paper is available at `https://arxiv.org/abs/1907.08093`.

1 Introduction

The Metric Dimension problem has been introduced in the 1970s independently by Slater [22] and by Harary and Melter [13]. Given a graph G and an integer k, Metric Dimension asks for a subset S of vertices of G of size at most k such that every vertex of G is uniquely determined by its distances to the vertices of S. Such a set S is called a *resolving set*, and a resolving set of minimum-cardinality is called a *metric basis*. The metric dimension of graphs finds application in various areas including network verification [2], chemistry [4], and robot navigation [18].

Metric Dimension is an entry of the celebrated book on intractability by Garey and Johnson [12] where the authors show that it is NP-complete. In fact Metric Dimension remains NP-complete in many restricted classes of graphs such as planar graphs [6], split, bipartite, co-bipartite graphs, and line graphs of bipartite graphs [9], interval graphs of diameter two [11], permutation graphs of diameter two [11], and in a subclass of unit disk graphs [16]. Furthermore Metric Dimension cannot be solved in subexponential-time unless 3-SAT can [1]. On the positive side, the problem is polynomial-time solvable on trees [22, 13, 18]. Diaz et al. [6] generalize this result to outerplanar graphs. Fernau et al. [10]

give a polynomial-time algorithm on chain graphs. Epstein et al. [9] show that Metric Dimension (and even its vertex-weighted variant) can be solved in polynomial time on co-graphs and forests augmented by a constant number of edges. Hoffmann et al. [15] obtain a linear algorithm on cactus block graphs.

Hartung and Nichterlein [14] prove that Metric Dimension is W[2]-complete (parameterized by the size of the solution k) even on subcubic graphs. Therefore an FPT algorithm solving the problem is unlikely. However Foucaud et al. [11] give an FPT algorithm with respect to k on interval graphs. This result is later generalized by Belmonte et al. [3] who obtain an FPT algorithm with respect to $\mathrm{tl} + \Delta$ (where tl is the tree-length and Δ is the maximum-degree of the input graph), implying one for parameter $\mathrm{tl} + k$. Indeed interval graphs, and even chordal graphs, have constant tree-length. Hartung and Nichterlein [14] presents an FPT algorithm parameterized by the vertex cover number, Eppstein [8], by the max leaf number, and Belmonte et al. [3], by the modular-width (a larger parameter than clique-width).

The complexity of Metric Dimension parameterized by treewidth is quite elusive. It is discussed [8] or raised as an open problem in several papers [3, 6]. On the one hand, it was not known, prior to our paper, if this problem is W[1]-hard. On the other hand, the complexity of Metric Dimension in graphs of treewidth at most two is still an open question.

1.1 Our contribution

We settle the parameterized complexity of Metric Dimension with respect to treewidth. We show that this problem is W[1]-hard, and we rule out, under the Exponential Time Hypothesis (ETH), an algorithm running in $f(\mathrm{tw})|V(G)|^{o(\mathrm{tw})}$, where G is the input graph, tw its treewidth, and f any computable function. Our reduction even shows that an algorithm in time $f(\mathrm{pw})|V(G)|^{o(\mathrm{pw})}$ is unlikely on constant-degree graphs, for the larger parameter pathwidth pw. This is in stark contrast with the FPT algorithm of Belmonte et al. [3] for the parameter $\mathrm{tl} + \Delta$ where tl is the tree-length and Δ is the maximum-degree of the graph. We observe that this readily gives an FPT algorithm for $\mathrm{ctw} + \Delta$ where ctw is the connected treewidth, since $\mathrm{ctw} \geqslant \mathrm{tl}$. This unravels an interesting behavior of Metric Dimension, at least on bounded-degree graphs: usual tree-decompositions are not enough for efficient solving. Instead one needs tree-decompositions with an additional guarantee that the vertices of a same bag are at a bounded distance from each other.

As our construction is quite technical, we chose to introduce an intermediate problem dubbed k-Multicolored Resolving Set in the reduction from k-Multicolored Independent Set to Metric Dimension. The first half of the reduction, from k-Multicolored Independent Set to k-Multicolored Resolving Set, follows a generic and standard recipe to design parameterized hardness with respect to treewidth. The main difficulty is to design an effective *propagation gadget* with a constant-size left-right cut. The second half brings some new local attachments to the produced graph, to bridge the gap between k-Multicolored Resolving Set and Metric Dimension. Along the way, we introduce a number of gadgets: edge, propagation, forced set, forced vertex. They are quite streamlined and effective. Therefore, we believe these building blocks may help in designing new reductions for Metric Dimension.

1.2 Organization of the paper

In Section 2 we introduce the definitions, notations, and terminology used throughout the paper. In Section 3 we present the high-level ideas to establish our result. We define the k-Multicolored Resolving Set problem which serves as an intermediate step for our reduction. In Section 4 we design a parameterized reduction from the W[1]-complete k-Multicolored Independent Set to k-Multicolored Resolving Set parameterized by treewidth. In Section 5 we show how to transform the produced instances of k-Multicolored Resolving Set to Metric Dimension-instances (while maintaining bounded treewidth). Due to space constraints, the proofs of lemmas marked with a star are deferred to the long version (in appendix).

2 Preliminaries

We denote by $[i, j]$ the set of integers $\{i, i+1, \ldots, j-1, j\}$, and by $[i]$ the set of integers $[1, i]$. If $\mathcal{X}$ is a set of sets, we denote by $\cup\mathcal{X}$ the union of them.

2.1 Graph notations

All our graphs are undirected and simple (no multiple edge nor self-loop). We denote by $V(G)$, respectively $E(G)$, the set of vertices, respectively of edges, of the graph G. For $S \subseteq V(G)$, we denote the *open neighborhood* (or simply *neighborhood*) of S by $N_G(S)$, i.e., the set of neighbors of S deprived of S, and the *closed neighborhood* of S by $N_G[S]$, i.e., the set $N_G(S) \cup S$. For singletons, we simplify $N_G(\{v\})$ into $N_G(v)$, and $N_G[\{v\}]$ into $N_G[v]$. We denote by $G[S]$ the subgraph of G induced by S, and $G - S := G[V(G) \setminus S]$. For $S \subseteq V(G)$ we denote by $\overline{S}$ the complement $V(G) \setminus S$. For $A, B \subseteq V(G)$, $E(A, B)$ denotes the set of edges in $E(G)$ with one endpoint in A and the other one in B.

The length of a path in an unweighted graph is simply the number of edges of the path. For two vertices $u, v \in V(G)$, we denote by $\text{dist}_G(u, v)$, the distance between u and v in G, that is the length of the shortest path between u and v. The diameter of a graph is the longest distance between a pair of its vertices. The diameter of a subset $S \subseteq V(G)$, denoted by $\text{diam}_G(S)$, is the longest distance between a pair of vertices in S. Note that the distance is taken in G, *not* in $G[S]$. In particular, when G is connected, $\text{diam}_G(S)$ is finite for every S. A *pendant* vertex is a vertex with degree one. A vertex u is *pendant to* v if v is the only neighbor of u. Two distinct vertices u, v such that $N(u) = N(v)$ are called *false twins*, and *true twins* if $N[u] = N[v]$. In particular, true twins are adjacent. In all the above notations with a subscript, we omit it whenever the graph is implicit from the context.

2.2 Exponential Time Hypothesis, FPT reductions, and W[1]-hardness

The *Exponential Time Hypothesis* (ETH) is a conjecture by Impagliazzo et al. [17] asserting that there is no $2^{o(n)}$-time algorithm for 3-SAT on instances with n variables. Lokshtanov et al. [20] survey conditional lower bounds under the ETH.

A standard use of an FPT reduction is to derive conditional lower bounds: if a problem (Π, κ) is thought not to admit an FPT algorithm, then an FPT reduction from (Π, κ) to (Π', κ') indicates that (Π', κ') is also unlikely to admit an FPT algorithm. We refer the reader to the textbooks [7, 5] for a formal definition of W[1]-hardness. For the purpose of this paper, we will just state that W[1]-hard are parameterized problems that are unlikely to be FPT, and that the following problem is W[1]-complete even when all the V_i have the same number of elements, say t (see for instance [21]).

k-Multicolored Independent Set (k-MIS)	**Parameter:** k
Input: An undirected graph G, an integer k, and $(V_1, \ldots, V_k)$ a partition of $V(G)$.	
Question: Is there a set $I \subseteq V(G)$ such that $\lvert I \cap V_i \rvert = 1$ for every $i \in [k]$, and $G[I]$ is edgeless?	

Every parameterized problem that k-Multicolored Independent Set FPT-reduces to is W[1]-hard. Our paper is thus devoted to designing an FPT reduction from k-Multicolored Independent Set to Metric Dimension parameterized by tw. Let us observe that the ETH implies that one (equivalently, every) W[1]-hard problem is not in the class of problems solvable in FPT time (FPT$\neq$W[1]). Thus if we admit that there is no subexponential algorithm solving 3-SAT, then k-Multicolored Independent Set is not solvable in time $f(k)|V(G)|^{O(1)}$. Actually under this stronger assumption, k-Multicolored Independent Set is not solvable in time $f(k)|V(G)|^{o(k)}$. A concise proof of that fact can be found in the survey on the consequences of ETH [20].

2.3 Metric dimension, resolved pairs, distinguished vertices

A pair of vertices $\{u, v\} \subseteq V(G)$ is said to be *resolved* by a set S if there is a vertex $w \in S$ such that $\mathrm{dist}(w, u) \neq \mathrm{dist}(w, v)$. A vertex u is said to be *distinguished* by a set S if for any $w \in V(G) \setminus \{u\}$, there is a vertex $v \in S$ such that $\mathrm{dist}(v, u) \neq \mathrm{dist}(v, w)$. A *resolving set* of a graph G is a set $S \subseteq V(G)$ such that every two distinct vertices $u, v \in V(G)$ are resolved by S. Equivalently, a resolving set is a set S such that every vertex of G is distinguished by S. Then Metric Dimension asks for a resolving set of size at most some threshold k. Note that a resolving set of minimum size is sometimes called a *metric basis* for G.

Metric Dimension (MD)	**Parameter:** $\mathrm{tw}(G)$
Input: An undirected graph G and an integer k.	
Question: Does G admit a resolving set of size at most k?	

Here we anticipate on the fact that we will mainly consider Metric Dimension parameterized by treewidth. Henceforth we sometimes use the notation Π/tw to emphasize that Π is not parameterized by the natural parameter (size of the resolving set) but by the treewidth of the input graph.

3 Outline of the W[1]-hardness proof of Metric Dimension/tw

We will show the following.

▶ **Theorem 1.** *Unless the ETH fails, there is no computable function f such that* Metric Dimension *can be solved in time $f(pw)n^{o(pw)}$ on constant-degree n-vertex graphs.*

We first prove that the following generalized version of Metric Dimension is W[1]-hard.

k-Multicolored Resolving Set (k-MRS)	**Parameter:** $\mathrm{tw}(G)$
Input: An undirected graph G, an integer k, a set $\mathcal{X}$ of q disjoint subsets of $V(G)$: $X_1, \ldots, X_q$, and a set $\mathcal{P}$ of pairs of vertices of G: $\{x_1, y_1\}, \ldots, \{x_h, y_h\}$.	
Question: Is there a set $S \subseteq V(G)$ of size q such that	
▪ (i) for every $i \in [q]$, $\lvert S \cap X_i \rvert = 1$, and	
▪ (ii) for every $p \in [h]$, there is an $s \in S$ satisfying $\mathrm{dist}_G(s, x_p) \neq \mathrm{dist}_G(s, y_p)$?	

In words, in this generalized version the resolving set is made by picking exactly one vertex in each set of $\mathcal{X}$, and not all the pairs should be resolved but only the ones in a prescribed set $\mathcal{P}$. We call *critical pair* a pair of $\mathcal{P}$. In the context of k-Multicolored

Resolving Set, we call *legal set* a set which satisfies the former condition, and *resolving set* a set which satisfies the latter. Thus a solution for k-Multicolored Resolving Set is a legal resolving set.

The reduction from k-Multicolored Independent Set starts with a well-established trick to show parameterized hardness by treewidth. We create m "empty copies" of the k-MIS-instance $(G, k, (V_1, \ldots, V_k))$, where $m := |E(G)|$ and $t := |V_i|$. We force exactly one vertex in each color class of each copy to be in the resolving set, using the set $\mathcal{X}$. In each copy, we introduce an edge gadget for a single (distinct) edge of G. Encoding an edge of k-MIS in the k-MRS-instance is fairly simple: we build a pair (of $\mathcal{P}$) which is resolved by every choice but the one *selecting both its endpoints* in the resolving set. We now need to force a *consistent choice of the vertex chosen in* V_i over all the copies. We thus design a propagation gadget. A crucial property of the propagation gadget, for the pathwidth of the constructed graph to be bounded, is that it admits a cut of size $O(k)$ disconnecting one copy from the other. Encoding a choice in V_i in the distances to four special vertices, called *gates*, we manage to build such a gadget with constant-size "left-right" separator per color class. This works by introducing t pairs (of $\mathcal{P}$) which are resolved by the south-west and north-east gates but not by the south-east and north-west ones. Then we link the vertices of a copy of V_i in a way that the higher their index, the more pairs they resolve in the propagation gadget to their left, and the fewer pairs they resolve in the propagation gadget to their right.

We then turn to the actual Metric Dimension problem. We design a gadget which simulates requirement (i) by forcing a vertex of a specific set X in the resolving set. This works by introducing two pairs that are only resolved by vertices of X. We attach this new gadget, called *forcing set* gadget, to all the k color classes of the m copies. Finally we have to make sure that a candidate solution resolves all the pairs, and not only the ones prescribed by $\mathcal{P}$. For that we attach two adjacent "pendant" vertices to strategically chosen vertices. One of these two vertices have to be in the resolving set since they are true twins, hence not resolved by any other vertex. Then everything is as if the unique common neighbor v of the true twins was added to the resolving set. Therefore we can perform this operation as long as v does not resolve any of the pairs of $\mathcal{P}$.

To facilitate the task of the reader, henceforth we stick to the following conventions:

- Index $i \in [k]$ ranges over the k *rows* of the (G)MD-instance or color classes of k-MIS.
- Index $j \in [m]$ ranges over the m *columns* of the (G)MD-instance or edges of k-MIS.
- Index $\gamma \in [t]$, ranges over the t vertices of a color class.

We invite the reader to look up Table 1 when in doubt about a notation/symbol relative to the construction.

4 Parameterized hardness of k-Multicolored Resolving Set/tw

In this section, we give an FPT reduction from the W[1]-complete k-Multicolored Independent Set to k-Multicolored Resolving Set parameterized by treewidth. More precisely, given a k-Multicolored Independent Set-instance $(G, k, (V_1, \ldots, V_k))$ we produce in polynomial-time an equivalent k-Multicolored Resolving Set-instance $(G', k', \mathcal{X}, \mathcal{P})$ where G' has pathwidth (hence treewidth) $O(k)$.

4.1 Construction

Let $(G, k, (V_1, \ldots, V_k))$ be an instance of k-Multicolored Independent Set where $(V_1, \ldots, V_k)$ is a partition of $V(G)$ and $V_i := \{v_{i,\gamma} \mid 1 \leqslant \gamma \leqslant t\}$. We arbitrarily number $e_1, \ldots, e_j, \ldots, e_m$ the m edges of G.

4.1.1 Overall picture

We start with a high-level description of the k-MRS-instance $(G', k', \mathcal{X}, \mathcal{P})$. For each color class V_i, we introduce m copies $V_i^1, \ldots, V_i^j, \ldots, V_i^m$ of a *selector gadget* to G'. Each set V_i^j is added to $\mathcal{X}$, so a solution has to pick exactly one vertex within each selector gadget. One can imagine the vertex-sets $V_i^1, \ldots, V_i^m$ to be aligned on the *i-th row*, with V_i^j occupying the *j-th column* (see Figure 1). Each V_i^j has t vertices denoted by $v_{i,1}^j, v_{i,2}^j, \ldots, v_{i,t}^j$, where each $v_{i,\gamma}^j$ "corresponds" to $v_{i,\gamma} \in V_i$. We make $v_{i,1}^j v_{i,2}^j \ldots v_{i,t}^j$ a path with $t-1$ edges.

For each edge $e_j \in E(G)$, we insert an *edge gadget* $\mathcal{G}(e_j)$ containing a pair of vertices $\{c_j, c_j'\}$ that we add to $\mathcal{P}$. Gadget $\mathcal{G}(e_j)$ is attached to V_i^j and $V_{i'}^j$, where $e_j \in E(V_i, V_{i'})$. The edge gadget is designed in a way that the only legal sets that do *not* resolve $\{c_j, c_j'\}$ are the ones that precisely pick $v_{i,\gamma}^j \in V_i^j$ and $v_{i',\gamma'}^j \in V_{i'}^j$ such that $e_j = v_{i,\gamma} v_{i',\gamma'}$. We add a *propagation gadget* $P_i^{j,j+1}$ between two consecutive copies V_i^j and V_i^{j+1}, where the indices in the superscript are taken modulo m. The role of the propagation gadget is to ensure that the choices in each V_i^j $(j \in [m])$ corresponds to the same vertex in V_i.

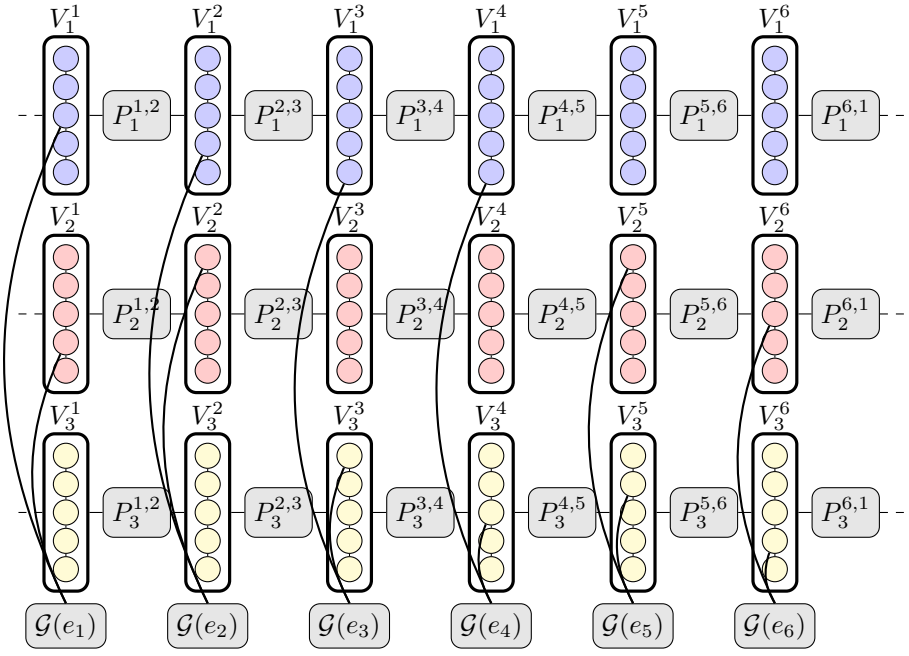


Figure 1 The overall picture with $k = 3$ color classes, $t = 5$ vertices per color class, $m = 6$ edges, $e_1 = v_{1,3}v_{2,4}$, $e_2 = v_{1,4}v_{2,1}$, $e_3 = v_{1,5}v_{3,1}$, etc. The dashed lines on the left and right symbolize that the construction is cylindrical.

The intuitive idea of the reduction is the following. We say that a vertex of G' is *selected* if it is put in the resolving set of G', a tentative solution. The propagation gadget $P_i^{j,j+1}$ ensures a consistent choice among the m copies $V_i^1, \ldots, V_i^m$. The edge gadget ensures that the selected vertices of G' correspond to an independent set in the original graph G. If both the endpoints of an edge e_j are selected, then the pair $\{c_j, c_j'\}$ is not resolved. We now detail the construction.

4.1.2 Selector gadget

For each $i \in [k]$ and $j \in [m]$, we add to G' a path on $t-1$ edges $v^j_{i,1}, v^j_{i,2}, \ldots, v^j_{i,t}$, and denote this set of vertices by V^j_i. Each $v^j_{i,\gamma}$ *corresponds* to $v_{i,\gamma} \in V_i$. We call *j-th column* the set $\bigcup_{i\in[k]} V^j_i$, and *i-th row*, the set $\bigcup_{j\in[m]} V^j_i$. We set $\mathcal{X} := \{V^j_i\}_{i\in[k],j\in[m]}$. By definition of k-Multicolored Resolving Set, a solution S has to satisfy that for every $i \in [k], j \in [m]$, $|S \cap V^j_i| = 1$. We call *legal set* a set S of size $k' = km$ that satisfies this property. We call *consistent set* a legal set S which takes the "same" vertex in each row, that is, for every $i \in [k]$, for every pair $(v^j_{i,\gamma}, v^{j'}_{i,\gamma'}) \in (S \cap V^j_i) \times (S \cap V^{j'}_i)$, then $\gamma = \gamma'$.

4.1.3 Edge gadget

For each edge $e_j = v_{i,\gamma}v_{i',\gamma'} \in E(G)$, we add an edge gadget $\mathcal{G}(e_j)$ in the j-th column of G'. $\mathcal{G}(e_j)$ consists of a path on three vertices: $c_j g_j c'_j$. The pair $\{c_j, c'_j\}$ is added to the list of critical pairs $\mathcal{P}$. We link both $v^j_{i,\gamma}$ and $v^j_{i',\gamma'}$ to g_j by a private path[1] of length $t+2$. We link the at least two and at most four vertices $v^j_{i,\gamma-1}, v^j_{i,\gamma+1}, v^j_{i',\gamma'-1}, v^j_{i',\gamma'+1}$ (whenever they exist) to c_j by a private path of length $t+2$. This defines at most six paths from $V^j_i \cup V^j_{i'}$ to $\mathcal{G}(e_j)$. Let us denote by W_j the at most six endpoints of these paths in $V^j_i \cup V^j_{i'}$. For each $v \in W_j$, we denote by $P(v,j)$ the path from v to $\mathcal{G}(e_j)$. We set $E^j_i := \bigcup_{v\in W_j\cap V^j_i} P(v,j)$ and $E^j_{i'} := \bigcup_{v\in W_j\cap V^j_{i'}} P(v,j)$. We denote by X_j the set of the at most six neighbors of W_j on the paths to $\mathcal{G}(e_j)$. Henceforth we may refer to the vertices in some X_j as the *cyan vertices*. Individually we denote by $e^j_{i,\gamma}$ the cyan vertex neighbor of $v^j_{i,\gamma}$ in $P(v^j_{i,\gamma}, j)$. We observe that for fixed i and j, $e^j_{i,\gamma}$ exists for at most three values of γ. We add an edge between two cyan vertices if their respective neighbors in V^j_i are also linked by an edge (or equivalently, if they have consecutive "indices γ"). These extra edges are useless in the k-MRS-instance, but will turn out useful in the MD-instance. See Figure 2 for an illustration of the edge gadget.

The rest of the construction will preserve that for every $v \in (V^j_i \cup V^j_{i'}) \setminus \{v^j_{i,\gamma}, v^j_{i',\gamma'}\}$, $\mathrm{dist}(v, c'_j) = \mathrm{dist}(v, c_j) + 2$, and for each $v \in \{v^j_{i,\gamma}, v^j_{i',\gamma'}\}$, $\mathrm{dist}(v, c_j) = \mathrm{dist}(v, g_j) + 1 = \mathrm{dist}(v, c'_j)$. In other words, the only two vertices of $V^j_i \cup V^j_{i'}$ not resolving the critical pair $\{c_j, c'_j\}$ are $v^j_{i,\gamma}$ and $v^j_{i',\gamma'}$, corresponding to the endpoints of e_j.

4.1.4 Propagation gadget

Between each pair (V^j_i, V^{j+1}_i), where $j+1$ is taken modulo m, we insert an identical copy of the propagation gadget, and we denote it by $P^{j,j+1}_i$. It ensures that if the vertex $v^j_{i,\gamma}$ is in a legal resolving set S, then the vertex of $S \cap V^{j+1}_i$ should be some $v^{j+1}_{i,\gamma'}$ with $\gamma \leqslant \gamma'$. The cylindricity of the construction and the fact that exactly one vertex of V^j_i is selected, will therefore impose that the set S is consistent.

$P^{j,j+1}_i$ comprises four vertices sw^j_i, se^j_i, nw^j_i, ne^j_i, called *gates*, and a set A^j_i of $2t$ vertices $a^j_{i,1}, \ldots, a^j_{i,t}, \alpha^j_{i,1}, \ldots, \alpha^j_{i,t}$. We make both $a^j_{i,1}a^j_{i,2}\ldots a^j_{i,t}$ and $\alpha^j_{i,1}\alpha^j_{i,2}\ldots\alpha^j_{i,t}$ a path with $t-1$ edges. For each $\gamma \in [t]$, we add the pair $\{a^j_{i,\gamma}, \alpha^j_{i,\gamma}\}$ to the set of critical pairs $\mathcal{P}$. Removing the gates disconnects A^j_i from the rest of the graph.

We now describe how we link the gates to V^j_i, V^{j+1}_i, and A^j_i. We link $v^j_{i,1}$ (the "top" vertex of V^j_i) to sw^j_i and $v^j_{i,t}$ (the "bottom" vertex of V^j_i) to nw^j_i both by a path of length 2.

[1] We use the expression *private path* to emphasize that the different sources get a pairwise internally vertex-disjoint path to the target.

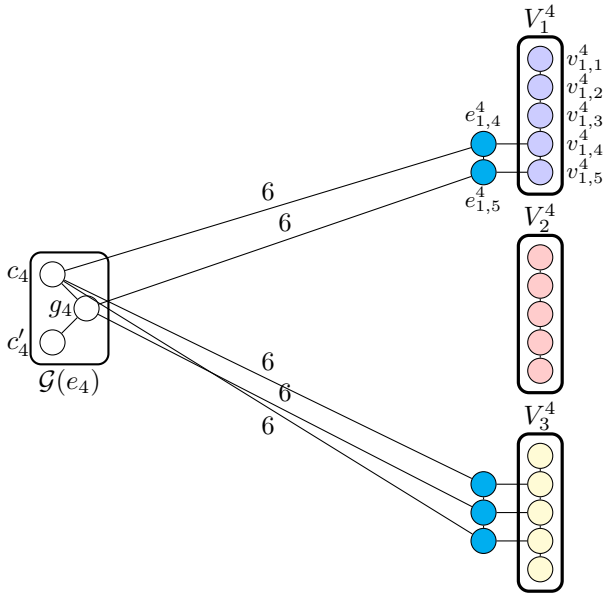


Figure 2 The edge gadget $\mathcal{G}(e_4)$ with $e_4 = v_{1,5}v_{3,3}$. Weighted edges are short-hands for subdivisions of the corresponding length. The edges between the cyan vertices will not be useful for the k-MRS-instance, but will later simplify the construction of the MD-instance.

We also link $v_{i,1}^{j+1}$ to se_i^j by a path of length 3, and $v_{i,t}^{j+1}$ to ne_i^j by a path of length 2. Then we make nw_i^j adjacent to $a_{i,1}^j$ and $\alpha_{i,1}^j$, while we make ne_i^j adjacent to $\alpha_{i,1}^j$ only. We make se_i^j adjacent to $a_{i,t}^j$ and $\alpha_{i,t}^j$, while we make sw_i^j adjacent to $a_{i,t}^j$ only. Finally, we add an edge between ne_i^j and nw_i^j, and between sw_i^j and se_i^j. See Figure 3 for an illustration of the propagation gadget $P_i^{j,j+1}$ with $t = 5$.

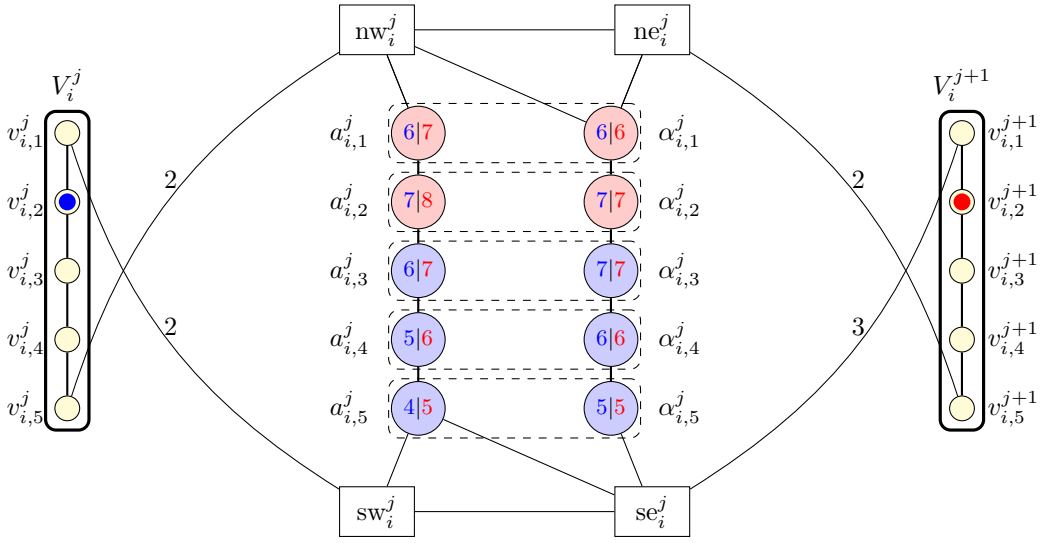


Figure 3 The propagation gadget $P_i^{j,j+1}$. The critical pairs $\{a_{i,\gamma}^j, \alpha_{i,\gamma}^j\}$ are surrounded by thin dashed lines. The blue (resp. red) integer on a vertex of A_i^j is its distance to the blue (resp. red) vertex in V_i^j (resp. V_i^{j+1}). Note that the blue vertex distinguishes the critical pairs below it, while the red vertex distinguishes critical pairs at its level or above.

Let us motivate the gadget $P_i^{j,j+1}$. One can observe that the gates ne_i^j and sw_i^j resolve the critical pairs of the propagation gadget, while the gates nw_i^j and se_i^j do not. Consider that the vertex added to the resolving set in V_i^j is $v_{i,\gamma}^j$. Its shortest paths to critical pairs *below* it (that is, with index $\gamma' > \gamma$) go through the gate sw_i^j, whereas its shortest paths to critical pairs at its level or above (that is, with index $\gamma' \leqslant \gamma$) go through the gate nw_i^j. Thus $v_{i,\gamma}^j$ only resolves the critical pairs $\{a_{i,\gamma'}^j, \alpha_{i,\gamma'}\}$ with $\gamma' > \gamma$. On the contrary, the vertex of the resolving set in V_i^{j+1} only resolves the critical pairs $\{a_{i,\gamma'}^j, \alpha_{i,\gamma'}^j\}$ at its level or above. This will force that its level is γ or below. Hence the vertices of the resolving in V_i^j and V_i^{j+1} should be such that $\gamma' \geqslant \gamma$. Since there is also a propagation gadget between V_i^m and V_i^1, this circular chain of inequalities forces a global equality.

4.1.5 Wrapping up

We put the pieces together as described in the previous subsections. At this point, it is convenient to give names to the neighbors of V_i^j in the propagation gadgets $P_i^{j-1,j}$ and $P_i^{j,j+1}$. We may refer to them as *blue vertices* (as they appear in Figure 4). We denote by tl_i^j the neighbor of $v_{i,1}^j$ in $P_i^{j-1,j}$, tr_i^j, the neighbor of $v_{i,1}^j$ in $P_i^{j,j+1}$, bl_i^j, the neighbor of $v_{i,t}^j$ in $P_i^{j-1,j}$, and br_i^j, the neighbor of $v_{i,t}^j$ in $P_i^{j,j+1}$. We add the following edges and paths.

For any pair i, j such that the edge e_j has an endpoint in V_i, the vertices $\mathrm{tl}_i^j, \mathrm{tr}_i^j, \mathrm{bl}_i^j, \mathrm{br}_i^j$ are linked to g_j by a private path of length the distance of their unique neighbor in V_i^j to c_j. We add an edge between se_i^j and se_i^{j+1}, and between nw_i^j and nw_i^{j+1} (where $j+1$ is modulo m). Finally, for every $e_j \in E(V_i, V_{i'})$, we add four paths between $\mathrm{se}_i^j, \mathrm{se}_{i'}^j, \mathrm{nw}_i^j, \mathrm{nw}_{i'}^j$ and $g_j \in \mathcal{G}(e_j)$. More precisely, for each $i'' \in \{i, i'\}$, we add a path from g_j to $\mathrm{se}_{i''}^j$ of length $\mathrm{dist}(g_j, \mathrm{sw}_{i''}^j) - 4$, and a path from g_j to $\mathrm{nw}_{i''}^j$ of length $\mathrm{dist}(g_j, \mathrm{nw}_{i''}^j) - 4$. These distances are taken in the graph before we introduced the new paths, and one can observe that the length of these paths is at least t. This finishes the construction.

4.2 Correctness of the reduction

We now check that the reduction is correct. We start with the following technical lemma. If a set X contains a pair that no vertex of $N(X)$ (that is $N[X] \setminus X$) resolves, then no vertex outside X can distinguish the pair.

▶ **Lemma 2.** *Let X be a subset of vertices, and $a, b \in X$ be two distinct vertices. If for every vertex $v \in N(X)$, $dist(v, a) = dist(v, b)$, then for every vertex $v \notin X$, $dist(v, a) = dist(v, b)$.*

Proof. Let v be a vertex outside of X. We further assume that v is not in $N(X)$, otherwise we can already conclude that it does not distinguish $\{a, b\}$. A shortest path from v to a, has to go through $N(X)$. Let w_a be the first vertex of $N(X)$ met in this shortest path from v to a. Similarly, let w_b be the first vertex of $N(X)$ met in a shortest path from v to b. Since $w_a, w_b \in N(X)$, they satisfy $\mathrm{dist}(w_a, a) = \mathrm{dist}(w_a, b)$ and $\mathrm{dist}(w_b, a) = \mathrm{dist}(w_b, b)$. Then, $\mathrm{dist}(v, a) \leqslant \mathrm{dist}(v, w_b) + \mathrm{dist}(w_b, a) = \mathrm{dist}(v, w_b) + \mathrm{dist}(w_b, b) = \mathrm{dist}(v, b)$, and $\mathrm{dist}(v, b) \leqslant \mathrm{dist}(v, w_a) + \mathrm{dist}(w_a, b) = \mathrm{dist}(v, w_a) + \mathrm{dist}(w_a, a) = \mathrm{dist}(v, a)$. Thus $\mathrm{dist}(v, a) = \mathrm{dist}(v, b)$. ◀

We use the previous lemma to show that every vertex of a V_i^j only resolves critical pairs in gadgets it is attached to. This will be useful in the two subsequent lemmas.

▶ **Lemma 3 (⋆).** *For any $i \in [k]$, $j \in [m]$, and $v \in V_i^j$, v does not resolve any critical pair outside of $P_i^{j-1,j}$, $P_i^{j,j+1}$ (where indices in the superscript are taken modulo m), and $\{c_j, c'_j\}$. Furthermore, if $e_j \in E(G)$ has no endpoint in $V_i \subseteq V(G)$, then v does not resolve $\{c_j, c'_j\}$.*

The two following lemmas show the equivalences relative to the expected use of the edge and propagation gadgets. They will be useful in Sections 4.2.1 and 4.2.2.

▶ **Lemma 4** (⋆). *A legal set S resolves the critical pair $\{c_j, c'_j\}$ with $e_j = v_{i,\gamma}v_{i',\gamma'}$ if and only if the vertex v^j_{i,γ_i} in $V^j_i \cap S$ and the vertex $v^j_{i',\gamma_{i'}}$ in $V^j_{i'} \cap S$ satisfy $(\gamma, \gamma') \neq (\gamma_i, \gamma_{i'})$.*

▶ **Lemma 5** (⋆). *A legal set S resolves all the critical pairs of $P^{j,j+1}_i$ if and only if the vertex $v^j_{i,\gamma}$ in $V^j_i \cap S$ and the vertex $v^{j+1}_{i,\gamma'}$ in $V^{j+1}_i \cap S$ satisfy $\gamma \leqslant \gamma'$.*

We can now prove the correctness of the reduction. The construction can be computed in polynomial time in $|V(G)|$, and G' itself has size bounded by a polynomial in $|V(G)|$. We postpone checking that the pathwidth is bounded by $O(k)$ to the end of the second step, where we produce an instance of MD whose graph G'' admits G' as an induced subgraph.

4.2.1 k-Multicolored Independent Set in $G \Rightarrow$ legal resolving set in G'

Let $\{v_{1,\gamma_1}, \ldots, v_{k,\gamma_k}\}$ be a k-multicolored independent set in G. We claim that $S := \bigcup_{j\in[m]}\{v^j_{1,\gamma_1}, \ldots, v^j_{k,\gamma_k}\}$ is a legal resolving set in G' (of size km). The set S is legal by construction. Since for every $i \in [k]$, and $j \in [m]$, v^j_{i,γ_i} and v^{j+1}_{i,γ_i} are in S ($j+1$ is modulo m), all the critical pairs in the propagation gadgets are resolved by S, by Lemma 5. Since $\{v_{1,\gamma_1}, \ldots, v_{k,\gamma_k}\}$ is an independent set in G, there is no $e_j = v_{i,\gamma}v_{i',\gamma'} \in E(G)$, such that $(\gamma, \gamma') = (\gamma_i, \gamma_{i'})$. Thus every critical pair $\{c_j, c'_j\}$ is resolved by S, by Lemma 4.

4.2.2 Legal resolving set in $G' \Rightarrow$ k-Multicolored Independent Set in G

Assume that there is a legal resolving set S in G'. For every $i \in [k]$, for every $j \in [m]$, the vertex $v^j_{i,\gamma(i,j)}$ in $V^j_i \cap S$ and the vertex $v^{j+1}_{i,\gamma(i,j+1)}$ in $V^{j+1}_i \cap S$ ($j+1$ is modulo m) are such that $\gamma(i,j) \leqslant \gamma(i,j+1)$, by Lemma 5. Thus $\gamma(i,1) \leqslant \gamma(i,2) \leqslant \ldots \leqslant \gamma(i,m-1) \leqslant \gamma(i,m) \leqslant \gamma(i,1)$, and $\gamma_i := \gamma(i,1) = \gamma(i,2) = \ldots = \gamma(i,m-1) = \gamma(i,m)$. We claim that $\{v_{1,\gamma_1}, \ldots, v_{k,\gamma_k}\}$ is a k-multicolored independent set in G. Indeed, there cannot be an edge $e_j = v_{i,\gamma_i}v_{i',\gamma_{i'}} \in E(G)$, since otherwise the critical pair $\{c_j, c'_j\}$ is not resolved, by Lemma 4.

5 Parameterized hardness of Metric Dimension/tw

In this section, we produce in polynomial time an instance (G'', k'') of Metric Dimension equivalent to $(G', \mathcal{X}, km, \mathcal{P})$ of k-Multicolored Resolving Set. The graph G'' has also pathwidth $O(k)$. Now, an instance is just a graph and an integer. There is no longer $\mathcal{X}$ and $\mathcal{P}$ to constrain and respectively loosen the "resolving set" at our convenience. This creates two issues: (1) the vertices outside the former set $\mathcal{X}$ can now be put in the resolving set, potentially yielding undesired solutions[2] and (2) our candidate solution (when there is a k-multicolored independent set in G) may not distinguish all the vertices.

5.1 Construction

5.1.1 Forced set gadget

To deal with the issue (1), we introduce two new pairs of vertices for each V^j_i. The intention is that the only vertices resolving both these pairs simultaneously are precisely the vertices

[2] Also, it is now possible to put two or more vertices of the same V^j_i in the resolving set S

of V_i^j. For any $i \in [k]$ and $j \in [m]$, we add to G' two pairs of vertices $\{p_i^j, q_i^j\}$ and $\{r_i^j, s_i^j\}$, and two *gates* π_i^j and ρ_i^j. Vertex π_i^j is adjacent to p_i^j and q_i^j, and vertex ρ_i^j is adjacent to r_i^j and s_i^j.

We link $v_{i,1}^j$ to p_i^j, and $v_{i,t}^j$ to r_i^j, each by a path of length t. It introduces two new neighbors of $v_{i,1}^j$ and $v_{i,t}^j$ (the brown vertices in Figure 4). We denote them by tb_i^j and bb_i^j, respectively. The blue and brown vertices are linked to π_i^j and ρ_i^j in the following way. We link tl_i^j and tr_i^j to π_i^j by a private path of length t, and to ρ_i^j by a private path of length $2t-1$. We link bl_i^j and br_i^j to π_i^j by a private path of length $2t-1$, and to ρ_i^j by a private path of length t. (Let us clarify that the names of the blue vertices bl_i^j and br_i^j are for "bottom-left" and "bottom-right", and *not* for "blue" and "brown".) We link tb_i^j (neighbor of $v_{i,1}^j$) to ρ_i^j by a private path of length $2t-1$. We link bb_i^j (neighbor of $v_{i,t}^j$) to π_i^j by a private path of length $2t-1$. Note that the general rule to set the path length is to match the distance between the neighbor in V_i^j and p_i^j (resp. r_i^j). With that in mind we link, if it exists, the *top cyan vertex* tc_i^j (the one with smallest index γ) neighboring V_i^j to π_i^j with a path of length $\mathrm{dist}(v_{i,\gamma}^j, p_i^j) = t + \gamma - 1$ where $v_{i,\gamma}^j$ is the unique vertex in $N(\mathrm{tc}_i^j) \cap V_i^j$. Observe that with the notations of the previous section $\mathrm{tc}_i^j = e_{i,\gamma}^j$. We also link, if it exists, the *bottom cyan vertex* bc_i^j (the one with largest index γ) to ρ_i^j with a path of length $\mathrm{dist}(v, r_i^j)$ where v is again the unique neighbor of bc_i^j in V_i^j.

It can be observed that we only have two paths (and not all six) from the at most three cyan vertices to the gates π_i^j and ρ_i^j. This is where the edges between the cyan vertices will become relevant. See Figure 4 for an illustration of the forced vertex gadget, keeping in mind that, for the sake of legibility, four paths to $\{\pi_i^j, \rho_i^j\}$ are not represented.

5.1.2 Forced vertex gadget

We now deal with the issue (2). By *we add* (or *attach*) *a forced vertex* to an already present vertex v, we mean that we add two adjacent neighbors to v, and that these two vertices remain of degree 2 in the whole graph G''. Hence one of the two neighbors will have to be selected in the resolving set since they are true twins. We call *forced vertex* one of these two vertices (picking arbitrarily).

For every $i \in [k]$ and $j \in [m]$, we add a forced vertex to the gates nw_i^j and se_i^j of $P_i^{j,j+1}$. We also add a forced vertex to each vertex in $N(\{\pi_i^j, \rho_i^j\}) \setminus \{p_i^j, q_i^j, r_i^j, s_i^j\}$. This represents a total of 12 vertices (6 neighbors of π_i^j and 6 neighbors of ρ_i^j). For every $j \in [m]$, we attach a forced vertex to each vertex in $N(g_j) \setminus \{c_j, c'_j\}$. This constitutes 14 neighbors (hence 14 new forced vertices). Therefore we set $k'' := km + 12km + 2km + 14m = 15km + 14m$.

5.1.3 Finishing touches and useful notations

We use the convention that $P(u, v)$ denotes the path from u to v which was specifically built from u to v. In other words, for $P(u, v)$ to make sense, there should be a point in the construction where we say that we add a (private) path between u and v. For the sake of legibility, $P(u, v)$ may denote either the set of vertices or the induced subgraph. We also denote by $\nu(u, v)$ the neighbor of u in the path $P(u, v)$. Observe that $P(u, v)$ is a symmetric notation but not $\nu(u, v)$.

We add a path of length $\mathrm{dist}(\nu(\pi_i^j, \mathrm{tr}_i^j), \mathrm{sw}_i^j) = t$ between $\nu(\pi_i^j, \mathrm{tr}_i^j)$ and se_i^j, and a path of length $\mathrm{dist}(\nu(\pi_i^j, \mathrm{bl}_i^j), \mathrm{ne}_i^{j-1}) = 2t - 1$ between $\nu(\pi_i^j, \mathrm{bl}_i^j)$ and nw_i^{j-1}. Similarly, we add a path of length $\mathrm{dist}(\nu(\rho_i^j, \mathrm{tr}_i^j), \mathrm{sw}_i^j) = 2t - 1$ between $\nu(\rho_i^j, \mathrm{tr}_i^j)$ and se_i^j, and a path of length $\mathrm{dist}(\nu(\rho_i^j, \mathrm{bl}_i^j), \mathrm{ne}_i^{j-1}) = t$ between $\nu(\rho_i^j, \mathrm{bl}_i^j)$ and nw_i^{j-1}. We added these four paths so that no forced vertex resolves any critical pair in the propagation gadgets $P_i^{j-1,j}$ and $P_i^{j,j+1}$.

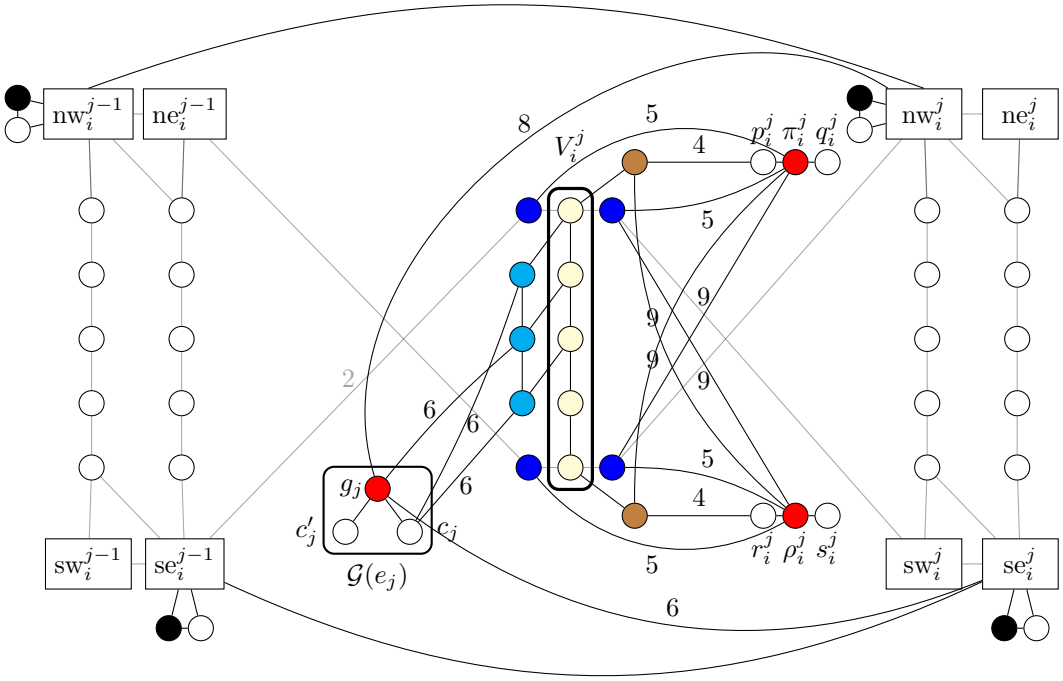


Figure 4 Vertices $\mathrm{tl}_i^j, \mathrm{tr}_i^j, \mathrm{bl}_i^j, \mathrm{br}_i^j$ (blue vertices) are linked to π_i^j, ρ_i^j by paths of appropriate lengths (see Section 5.1.1). Vertex tb_i^j is linked by a path to ρ_i^j, while bb_i^j is linked by a path to π_i^j. To avoid cluttering the figure, we did not represent four paths: from tl_i^j and bc_i^j to ρ_i^j, and from bl_i^j and tc_i^j to π_i^j. We also did not represent the paths already in the k-MRS-instance from the blue vertices to g_j. Black vertices are forced vertices. Gray edges are the edges in the propagation gadgets already depicted in Figure 3. Not represented on the figure, we add a forced vertex to each neighbor of the red vertices, except $p_i^j, q_i^j, r_i^j, s_i^j, c_j, c'_j$. Finally we add four more paths and potentially two edges (see Section 5.1.3).

Finally we add an edge between $\nu(g_j, \mathrm{nw}_i^j)$ and $\nu(c_j, \mathrm{bc}_i^j)$ whenever V_i^j have exactly three cyan vertices. We do that to resolve the pair $\{\nu(c_j, \mathrm{tc}_i^j), \nu(c_j, \mathrm{bc}_i^j)\}$, and more generally every pair $\{x, y\} \in P(c_j, \mathrm{tc}_i^j) \times P(c_j, \mathrm{bc}_i^j)$ such that $\mathrm{dist}(c_j, x) = \mathrm{dist}(c_j, y)$. This finishes the construction of the instance $(G'', k'' := 15km + 14m)$ of Metric Dimension.

5.2 Correctness of the reduction

The two next lemmas will be crucial in Section 5.2.1. The first lemma shows how the forcing set gadget simulates the action of former set $\mathcal{X}$.

▶ **Lemma 6** (⋆). *For every $i \in [k]$ and $j \in [m]$,*
- *$\forall v \in V_i^j$, v resolves both pairs $\{p_i^j, q_i^j\}$ and $\{r_i^j, s_i^j\}$,*
- *$\forall v \notin V_i^j$, v resolves at most one pair of $\{p_i^j, q_i^j\}$ and $\{r_i^j, s_i^j\}$,*
- *$\forall v \notin V_i^j \cup P(v_{i,1}^j, p_i^j) \cup P(v_{i,t}^j, r_i^j) \cup \{q_i^j, s_i^j\}$, v does not resolve $\{p_i^j, q_i^j\}$ nor $\{r_i^j, s_i^j\}$.*

For Section 5.2.1, we also need the following lemma, which states that the forced vertices do not resolve critical pairs.

▶ **Lemma 7** (⋆). *No forced vertex resolves a pair of $\mathcal{P}$.*

5.2.1 MD-instance has a solution ⇒ k-MRS-instance has a solution

Let S be a resolving set for the Metric Dimension-instance. We show that $S' := S \cap \bigcup_{i\in[k],j\in[m]} V_i^j$ is a solution for k-Multicolored Resolving Set. The set $S \setminus S'$ is made of $14km+14m$ forced vertices, none of which is in some $V_i^j \cup P(v_{i,1}^j, p_i^j) \cup \{q_i^j\} \cup P(v_{i,t}^j, r_i^j) \cup \{s_i^j\}$. Thus by Lemma 6, $S \setminus S'$ does not resolve any pair $\{p_i^j, q_i^j\}$ or $\{r_i^j, s_i^j\}$. Now S' is a set of $k'' - (14km + 14m) = km$ vertices resolving all the $2km$ pairs $\{p_i^j, q_i^j\}$ and $\{r_i^j, s_i^j\}$. Again by Lemma 6, this is only possible if $|S' \cap V_i^j| = 1$. Thus S' is a legal set of size $k' = km$. Let us now check that S' resolves every pair of $\mathcal{P}$ in the graph G'.

By Lemma 7, $S \setminus S'$ does not resolve any pair of $\mathcal{P}$ in the graph G''. Thus S' resolves all the pairs of $\mathcal{P}$ in G''. Since the distances between V_i^j and the critical pairs in the edge and propagation gadgets V_i^j is attached to are the same in G' and in G'', S' also resolves every pair of $\mathcal{P}$ in G'. Thus S' is a solution for the k-MRS-instance.

5.2.2 k-MRS-instance has a solution ⇒ MD-instance has a solution

Let S be a solution for k-Multicolored Resolving Set. We show that $S' := S \cup F$, where F is the set of forced vertices, is a solution for Metric Dimension.

▶ **Lemma 8** ($\star$). *Every vertex in G'' is distinguished by S'.*

The reduction is correct and it takes polynomial-time in $|V(G)|$ to compute G''. The maximum degree of G'' is 16. It is the degree of the vertices g_j (nw_i^j and se_i^j have degree at most 11, π_i^j and ρ_i^j have degree 8, and the other vertices have degree at most 5). We use the pathwidth characterization of Kirousis and Papadimitriou [19], to show:

▶ **Lemma 9** ($\star$). *$pw(G'') \leqslant 90k + 83$.*

Then solving Metric Dimension on constant-degree graphs in time $f(\mathrm{pw})n^{o(\mathrm{pw})}$ could be used to solve k-Multicolored Independent Set in time $f(k)n^{o(k)}$, disproving the ETH.

References

1. Florian Barbero, Lucas Isenmann, and Jocelyn Thiebaut. On the Distance Identifying Set Meta-Problem and Applications to the Complexity of Identifying Problems on Graphs. In *13th International Symposium on Parameterized and Exact Computation, IPEC 2018, August 20-24, 2018, Helsinki, Finland*, pages 10:1–10:14, 2018. doi:10.4230/LIPIcs.IPEC.2018.10.
2. Zuzana Beerliova, Felix Eberhard, Thomas Erlebach, Alexander Hall, Michael Hoffmann, Matús Mihalák, and L. Shankar Ram. Network Discovery and Verification. *IEEE Journal on Selected Areas in Communications*, 24(12):2168–2181, 2006. doi:10.1109/JSAC.2006.884015.
3. Rémy Belmonte, Fedor V. Fomin, Petr A. Golovach, and M. S. Ramanujan. Metric Dimension of Bounded Tree-length Graphs. *SIAM J. Discrete Math.*, 31(2):1217–1243, 2017. doi:10.1137/16M1057383.
4. Gary Chartrand, Linda Eroh, Mark A. Johnson, and Ortrud Oellermann. Resolvability in graphs and the metric dimension of a graph. *Discrete Applied Mathematics*, 105(1-3):99–113, 2000. doi:10.1016/S0166-218X(00)00198-0.
5. 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.
6. Josep Díaz, Olli Pottonen, Maria J. Serna, and Erik Jan van Leeuwen. Complexity of metric dimension on planar graphs. *J. Comput. Syst. Sci.*, 83(1):132–158, 2017. doi:10.1016/j.jcss.2016.06.006.

7 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.

8 David Eppstein. Metric Dimension Parameterized by Max Leaf Number. *J. Graph Algorithms Appl.*, 19(1):313–323, 2015. doi:10.7155/jgaa.00360.

9 Leah Epstein, Asaf Levin, and Gerhard J. Woeginger. The (Weighted) Metric Dimension of Graphs: Hard and Easy Cases. *Algorithmica*, 72(4):1130–1171, 2015. doi:10.1007/s00453-014-9896-2.

10 Henning Fernau, Pinar Heggernes, Pim van 't Hof, Daniel Meister, and Reza Saei. Computing the metric dimension for chain graphs. *Inf. Process. Lett.*, 115(9):671–676, 2015. doi:10.1016/j.ipl.2015.04.006.

11 Florent Foucaud, George B. Mertzios, Reza Naserasr, Aline Parreau, and Petru Valicov. Identification, Location-Domination and Metric Dimension on Interval and Permutation Graphs. II. Algorithms and Complexity. *Algorithmica*, 78(3):914–944, 2017. doi:10.1007/s00453-016-0184-1.

12 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

13 Frank Harary and Robert A Melter. On the metric dimension of a graph. *Ars Combin*, 2(191-195):1, 1976.

14 Sepp Hartung and André Nichterlein. On the Parameterized and Approximation Hardness of Metric Dimension. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, K.lo Alto, California, USA, 5-7 June, 2013*, pages 266–276, 2013. doi:10.1109/CCC.2013.36.

15 Stefan Hoffmann, Alina Elterman, and Egon Wanke. A linear time algorithm for metric dimension of cactus block graphs. *Theor. Comput. Sci.*, 630:43–62, 2016. doi:10.1016/j.tcs.2016.03.024.

16 Stefan Hoffmann and Egon Wanke. Metric Dimension for Gabriel Unit Disk Graphs Is NP-Complete. In *Algorithms for Sensor Systems, 8th International Symposium on Algorithms for Sensor Systems, Wireless Ad Hoc Networks and Autonomous Mobile Entities, ALGOSENSORS 2012, Ljubljana, Slovenia, September 13-14, 2012. Revised Selected Papers*, pages 90–92, 2012. doi:10.1007/978-3-642-36092-3_10.

17 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63(4):512–530, December 2001.

18 Samir Khuller, Balaji Raghavachari, and Azriel Rosenfeld. Landmarks in Graphs. *Discrete Applied Mathematics*, 70(3):217–229, 1996. doi:10.1016/0166-218X(95)00106-2.

19 Lefteris M. Kirousis and Christos H. Papadimitriou. Interval graphs and searching. *Discrete Mathematics*, 55(2):181–184, 1985. doi:10.1016/0012-365X(85)90046-9.

20 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/92.

21 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. Comput. Syst. Sci.*, 67(4):757–771, 2003. doi:10.1016/S0022-0000(03)00078-3.

22 Peter J Slater. Leaves of trees. *Congr. Numer*, 14(549-559):37, 1975.

Table 1 Glossary of the construction.

Symbol/term	Definition/action
$\{a^j_{i,\gamma}, \alpha^j_{i,\gamma}\}$	critical pair of the propagation gadget $P_i^{j,j+1}$
A^j_i	set of vertices $\bigcup_{\gamma\in[t]}\{a^j_{i,\gamma}, \alpha^j_{i,\gamma}\}$
bb^j_i	bottom brown vertex, $\nu(v^j_{i,t}, r^j_i)$
bc^j_i	bottom cyan vertex (smallest index γ)
bl^j_i	neighbor of $v^j_{i,t}$ in $P_i^{j-1,j}$
blue vertex	one of the four neighbors of V^j_i in the propagation gadgets
br^j_i	neighbor of $v^j_{i,t}$ in $P_i^{j,j+1}$
brown vertex	vertices $\nu(v^j_{i,1}, p^j_i)$ and $\nu(v^j_{i,t}, r^j_i)$
$\{c_j, c'_j\}$	critical pair of the edge gadget $\mathcal{G}(e_j)$
cyan vertex	neighbor of V^j_i in the paths to $\mathcal{G}(e_j)$
E^j_i	vertices in the paths from V^j_i to $\mathcal{G}(e_j)$
$e^j_{i,\gamma}$	alternative labeling of the cyan vertices, neighbor of $v^j_{i,\gamma}$
F	set of all forced vertices
$\mathcal{G}(e_j)$	edge gadget on $\{g_j, c_j, c'_j\}$ between V^j_i and $V^j_{i'}$, where $e_j \in E(V_i, V_{i'})$
mc^j_i	middle cyan vertex (not top nor bottom)
ne^j_i	north-east gate of $P_i^{j,j+1}$
nw^j_i	north-west gate of $P_i^{j,j+1}$
ne^j_i, sw^j_i	resolve the critical pairs of $P_i^{j,j+1}$
nw^j_i, se^j_i	do not resolve the critical pairs of $P_i^{j,j+1}$
$\nu(u, v)$	neighbor of u in the path $P(u, v)$
$\mathcal{P}$	list of critical pairs
$\{p^j_i, q^j_i\}$	pair only resolved by vertices in $V^j_i \cup P(v^j_{i,1}, p^j_i) \cup \{q^j_i\}$
π^j_i	gate of $\{p^j_i, q^j_i\}$, linked by paths to most neighbors of V^j_i
$P_i^{j,j+1}$	propagation gadget between V^j_i and V^{j+1}_i
$P(u, v)$	path added in the construction expressly between u and v
$\{r^j_i, s^j_i\}$	pair only resolved by vertices in $V^j_i \cup P(v^j_{i,t}, r^j_i) \cup \{s^j_i\}$
ρ^j_i	gate of $\{r^j_i, s^j_i\}$, linked by paths to most neighbors of V^j_i
se^j_i	south-east gate of $P_i^{j,j+1}$
sw^j_i	south-west gate of $P_i^{j,j+1}$
t	size of each V_i
tb^j_i	top brown vertex, $\nu(v^j_{i,1}, p^j_i)$
tc^j_i	top cyan vertex (largest index γ)
tl^j_i	neighbor of $v^j_{i,1}$ in $P_i^{j-1,j}$
tr^j_i	neighbor of $v^j_{i,1}$ in $P_i^{j,j+1}$
V_i	partite set of G
V^j_i	"copy of V_i", stringed by a path, in G' and G''
$v^j_{i,\gamma}$	vertex of V^j_i representing $v_{i,\gamma} \in V(G)$
W_j	endpoints in $V^j_i \cup V^j_{i'}$ of paths from $V^j_i \cup V^j_{i'}$ to $\mathcal{G}(e_j)$
$\mathcal{X}$	set containing all the sets V^j_i for $i \in [k]$ and $j \in [m]$
X_j	neighbors of W_j on the paths to $\mathcal{G}(e_j)$ (cyan vertices)

Faster Subgraph Counting in Sparse Graphs

Marco Bressan
Department of Computer Science, Sapienza University of Rome, Italy
bressan@di.uniroma1.it

Abstract

A fundamental graph problem asks to compute the number of induced copies of a k-node pattern graph H in an n-node graph G. The fastest algorithm to date is still the 35-years-old algorithm by Nešetřil and Poljak [28], with running time $f(k) \cdot O(n^{\omega \lfloor \frac{k}{3} \rfloor + 2})$ where $\omega \leq 2.373$ is the matrix multiplication exponent. In this work we show that, if one takes into account the degeneracy d of G, then the picture becomes substantially richer and leads to faster algorithms when G is sufficiently sparse. More precisely, after introducing a novel notion of graph width, the *DAG-treewidth*, we prove what follows. If H has DAG-treewidth $\tau(H)$ and G has degeneracy d, then the induced copies of H in G can be counted in time $f(d,k) \cdot \tilde{O}(n^{\tau(H)})$; and, under the Exponential Time Hypothesis, no algorithm can solve the problem in time $f(d,k) \cdot n^{o(\tau(H)/\ln \tau(H))}$ for all H. This result characterises the complexity of counting subgraphs in a d-degenerate graph. Developing bounds on $\tau(H)$, then, we obtain natural generalisations of classic results and faster algorithms for sparse graphs. For example, when $d = O(\text{poly} \log(n))$ we can count the induced copies of any H in time $f(k) \cdot \tilde{O}(n^{\lfloor \frac{k}{4} \rfloor + 2})$, beating the Nešetřil-Poljak algorithm by essentially a cubic factor in n.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Theory of computation → Design and analysis of algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases subgraph counting, tree decomposition, degeneracy

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.6

Related Version `https://arxiv.org/abs/1805.02089`

Funding Marco Bressan is supported in part by the ERC Starting Grant DMAP 680153, by a Google Focused Research Award "ALL4AI", and by the "Dipartimenti di Eccellenza 2018-2022" grant awarded to the Department of Computer Science of the Sapienza University of Rome.

1 Introduction

Given a host graph G on n nodes and a pattern graph H on k nodes, we want to count the number of induced copies of H in G. This problem is at the heart of many algorithmic applications but, unfortunately, is largely intractable. The fastest algorithm known has running time $O(n^{\omega \lfloor \frac{k}{3} \rfloor + 2})$ where ω is the matrix multiplication exponent [28]; and the margin to improve the dependence on k is limited, since under the Exponential Time Hypothesis [21] $n^{\Omega(k)}$ operations are required even just to detect a clique [8, 9]. The picture changes, however, if we make additional assumptions on G. A natural assumption is that G be *sparse*, as is often the case in practice. Under certain notions of sparsity, indeed, it is known that subgraph counting becomes tractable: for example, any H can be counted in time $f(k) \cdot O(n)$ if G has bounded maximum degree $\Delta(G) = O(1)$ [29]. Alternatively, any H can be counted in time $f(k) \cdot O(n)$ if G has bounded treewidth $\mathrm{t}(G) = O(1)$, as a consequence of Courcelle's theorem [27]. Similar bounds can be proved when G is planar [14]. These assumptions are much stronger than just having $O(1)$ average degree, and often do not hold in practice. For example, in social networks $\mathrm{t}(G)$ is typically large [26].

In this work we adopt a different measure of sparsity: the *degeneracy* of G, denoted by $d = d(G)$. The degeneracy can be defined as the minimum, over all acyclic orientations of G, of the maximum outdegree of a node; it is a notion strictly stronger than the average

degree (which it bounds from above), but strictly weaker than the maximum degree or the treewidth. Unlike $\Delta(G)$ or $\mathrm{t}(G)$, in social networks d is typically small [16]. Moreover, low-outdegree orientations of G, like the one that defines d, seem to help subgraph counting in practice [35, 22, 30]. Therefore, d seems a good candidate for a parameterization. Yet, no good bounds in terms of d exist, save for specific patterns such as cliques or complete bipartite graphs. This work aims at filling this gap. We develop techniques for counting subgraphs that exploit the low-outdegree orientation of G. This leads to a rich picture, and to faster algorithms to count subgraphs in sparse graphs.

1.1 Results

We present algorithms for counting homomorphisms, non-induced copies, and induced copies of a k-node pattern graph H in an n-node graph G, parameterized by n, k and the degeneracy d of G. Our contributions are of two kinds: bounds and techniques. For simplicity we assume $k = O(1)$ (see Subsection 1.2 for more details).

1.1.1 Bounds

Let $\hom(H, G)$, $\mathrm{sub}(H, G)$, and $\mathrm{ind}(H, G)$ denote respectively the number of homomorphisms, copies, and induced copies of H in G. Our first result is a general-purpose bound, that is, one holding for all H (including disconnected ones):

▶ **Theorem 1.** *For any k-node pattern H one can compute* $\hom(H, G)$*,* $\mathrm{sub}(H, G)$*, and* $\mathrm{ind}(H, G)$ *in time $f(d, k) \cdot \tilde{O}(n^{\lfloor \frac{k}{4} \rfloor + 2})$.*

This bound improves to $0.25k + O(1)$ the exponent of n, which is $0.791k + O(1)$ in the state-of-the-art algorithm of [28]. As an immediate consequence, we have:

▶ **Theorem 2.** *Suppose G has degeneracy $d = O(\mathrm{polylog}(n))$. Then for any k-node pattern H one can compute* $\hom(H, G)$*,* $\mathrm{sub}(H, G)$*, and* $\mathrm{ind}(H, G)$ *in time $f(k) \cdot \tilde{O}(n^{0.25k+2})$.*

Hence, when $d = O(\mathrm{polylog}(n))$, for all sufficiently large k our algorithm beats [28] by a cubic factor. In fact, our algorithm is faster than [28] already for $d < n^{0.721}$ assuming $\omega \approx 2.373$ [25], and in any case for $d < n^{\frac{5}{9}} \approx n^{0.556}$ since $\omega \geq 2$. A second consequence of Theorem 1 comes from the well-known fact that $|E(G)| \geq \binom{d}{2}$. Indeed, this implies:

▶ **Theorem 3.** *Suppose G has average degree $O(\mathrm{polylog}(n))$. Then for any k-node pattern H one can compute* $\hom(H, G)$*,* $\mathrm{sub}(H, G)$*, and* $\mathrm{ind}(H, G)$ *in time $f(k) \cdot \tilde{O}(n^{0.625k+1})$.*

Again, this holds for *all* patterns H, even disconnected ones. To the best of our knowledge, this is the first general-purpose algorithm faster than [28] for graphs with small *average degree* – the weakest possible notion of sparsity.

We give bounds for special classes of patterns, too. First, we consider quasi-cliques, a typical pattern in social graph mining [33, 32, 31]. We prove:

▶ **Theorem 4.** *If H is the clique minus ϵ edges, then one can compute* $\hom(H, G)$*,* $\mathrm{sub}(H, G)$*, and* $\mathrm{ind}(H, G)$ *in time $f(d, k) \cdot \tilde{O}(n^{\lceil \frac{1}{2} + \sqrt{\frac{\epsilon}{2}} \rceil})$.*

This generalizes the classic $O(nd^{k-1})$ bound for counting cliques [10], at the price of a polylogarithmic factor. Next, we look at complete quasi-multipartite graphs. We prove:

▶ **Theorem 5.** *If H is a complete multipartite graph, then one can compute* $\hom(H, G)$ *and* $\mathrm{sub}(H, G)$ *in time $f(d, k) \cdot \tilde{O}(n)$. If H is a complete multipartite graph plus ϵ edges, then one can compute* $\mathrm{sub}(H, G)$ *in time $f(d, k) \cdot \tilde{O}(n^{\lfloor \frac{\epsilon}{4} \rfloor + 2})$.*

This generalizes a classic $f(d,k) \cdot O(n)$ bound to count non-induced complete *bi*-partite graphs [13], again at the price of an additional factor.

1.1.2 Techniques

The bounds above are instantiations of a more general result, stated in terms of a novel notion of width, that we call *dag treewidth* $\tau(H)$ of H. Formally, we prove:

▶ **Theorem 6.** *For any k-node pattern H one can compute* $\hom(H,G)$, $\mathrm{sub}(H,G)$, *and* $\mathrm{ind}(H,G)$ *in time* $f(d,k) \cdot \tilde{O}(n^{\tau(H)})$.

This bound, and the width measure $\tau(H)$, arise as follows. As a first step, we orient G acyclically so that it has maximum outdegree d (see below). The problem then becomes counting the copies of all acyclic orientations P of H in G. By inclusion-exclusion arguments, we reduce the problem to counting homomorphisms between a dag P and the dag G. At this point we introduce our technical tool, the *dag tree decomposition* of P. This decomposition allows one to count homomorphisms naturally via dynamic programming, exactly like the standard tree decomposition of a graph; and the running time of the dynamic program is $f(d,k) \cdot \tilde{O}(n^{\tau(H)})$, where the *dag treewidth* $\tau(H)$ is, simplifying a little, the width of the decomposition. The crucial fact is that for $\tau(H)$ we can provide bounds better than just k or $\omega\lfloor\frac{k}{3}\rfloor + 2$ (for example, we prove $\tau(H) \leq \lfloor\frac{k}{4}\rfloor + 2$).

We complement Theorem 6 with a conditional lower bound, showing how $\tau(H)$ characterises the complexity of counting subgraphs in d-degenerate graphs:

▶ **Theorem 7.** *Under the Exponential Time Hypothesis [21], no algorithm can compute* $\mathrm{sub}(H,G)$ *or* $\mathrm{ind}(H,G)$ *in time* $f(d,k) \cdot n^{o(\tau(H)/\ln \tau(H))}$ *for all H.*

▶ Remark 8. Our algorithms work for the colored versions of the problem (count only copies of H with prescribed vertex and/or edge colors) as well as the weighted versions of the problem (compute the total weight of copies of H in G where G has weights on nodes or edges). This follows immediately by adapting our homomorphism counting algorithms.

1.2 Preliminaries and notation

The host graph $G = (V,E)$ and the pattern graph $H = (V_H, E_H)$ are simple, arbitrary graphs. For any subset $V' \subseteq V$ we denote by $G[V']$ the subgraph of G induced by V'; the same notation applies to any graph. A *homomorphism* from H to G is a map $\phi : V_H \to V$ such that $\{u,u'\} \in E_H$ implies $\{\phi(u), \phi(u')\} \in E$. We write $\phi : H \to G$ to highlight the edges that ϕ preserves. When H and G are oriented, ϕ must preserve the direction of the arcs. If ϕ is injective then we have an injective homomorphism. We denote by $\hom(H,G)$ and $\mathrm{inj}(H,G)$ the number of homomorphisms and injective homomorphisms from H to G. We denote by ψ a map that is not necessarily a homomorphism. The symbol $\simeq$ denotes isomorphism. A *copy* of H in G is a subgraph $F \subseteq G$ such that $F \simeq H$. If moreover $F \simeq G[V_F]$ then F is an induced copy. We denote by $\mathrm{sub}(H,G)$ and $\mathrm{ind}(H,G)$ the number of copies and induced copies of H in G; we may omit G if clear from the context. We denote by P a generic dag obtained by orienting H acyclically. All the notation described above applies to directed graphs in the natural way.

We denote by Δ the maximum degree of G. The *degeneracy* of G is the smallest d such that there is an acyclic orientation of G with maximum outdegree bounded by d. Such an orientation can be found in time $O(|E|)$ by repeatedly removing from G a minimum-degree node [27]. From now on, we assume G has this orientation. We also assume G is encoded

via sorted adjacency lists (every node keeps a sorted list of its out-neighbors). Checking for an arc in G thus takes time $O(\log(d))$, but to lighten the notation we assume it is $O(1)$. We always assume $k = O(1)$. Nonetheless, most of our bounds hold in their current form for $k = O(\ln n)$ or $k = O(\sqrt{\ln n})$. All asymptotic notations hide $\text{poly}(k)$ factors, and the $\tilde{O}(\cdot)$ notation hides $\text{polylog}(ndk)$ factors.

Finally, we recall the definitions of tree decomposition and treewidth of a graph. For any two nodes X, Y in a tree T, we denote by $T(X, Y)$ the unique path between X and Y in T.

▶ **Definition 9** (see [12], Ch. 12.3)**.** *Given a graph $G = (V, E)$, a tree decomposition of G is a tree $D = (V_D, E_D)$ such that each node $X \in V_D$ is a subset $X \subseteq V$, and that*[1]*:*
1. $\cup_{X \in V_D} X = V$
2. *for every edge $e = \{u, v\} \in G$ there exists $X \in D$ such that $u, v \in X$*
3. $\forall\, X, X', X'' \in V_T$*, if $X \in D(X', X'')$ then $X' \cap X'' \subseteq X$*
The width of a tree decomposition T is $\mathrm{t}(T) = \max_{X \in V_T} |X| - 1$. The treewidth $\mathrm{t}(G)$ of a graph G is the minimum of $\mathrm{t}(T)$ over all tree decompositions T of G.

1.3 Related work

The fastest algorithms known for computing $\text{ind}(H, G)$ are based on matrix multiplication and have running time $O(n^{\omega\lfloor \frac{k}{3} \rfloor + (k \bmod 3)})$ [28] or $O(n^{\omega(\lfloor k/3 \rfloor, \lceil (k-1)/3 \rceil, \lceil k/3 \rceil)})$ [17], where $\omega(p, q, r)$ is the cost of multiplying an $n^p \times n^q$ matrix by an $n^q \times n^r$ matrix and $\omega = \omega(n, n, n)$. With the current matrix multiplication algorithms, these bounds are essentially $O(n^{0.791k+2})$. These algorithms ignore the sparsity of G, and do not run faster if $d = O(1)$. In contrast, our goal is to reduce the running time when G is sparse. We mention that alternative techniques exist for probabilistic *approximate* counting. Notably, the color coding technique of Alon et al. [2] can be used to sample pattern copies uniformly at random from G in time $O(c^k|G|)$ for some constant $c > 0$ [5, 6, 7].

It is known that several notions of sparsity lead to bounds linear in n. If G has bounded maximum degree $\Delta = O(1)$, then we can compute $\text{ind}(H, G)$ in time $f(k) \cdot O(n)$ via multivariate graph polynomials [29]. If instead G has bounded treewidth $\mathrm{t}(G) = O(1)$, and we are given a tree decomposition of G of width $O(1)$, then by an extension of Courcelle's theorem we can compute $\text{ind}(H, G)$ in time $f(k) \cdot O(n)$ [27]. Similarly, if G is planar, then it can be partitioned into pieces of small treewidth, leading again to an $f(k) \cdot O(n)$ bound [14]. All these conditions are strictly stronger than (and they imply) bounded degeneracy, $d = O(1)$. The techniques used for these bounds are radically different from ours.

For d-degenerate graphs, bounds are known only for special classes of patterns. Chiba and Nishizeki [10] show how to count k-cliques in time $O(d^{k-1}n)$, which can be improved to $O(d^{\omega\lceil (k-1)/3 \rceil}n)$ via matrix multiplication [1]. Eppstein shows how to list all complete bipartite subgraphs in time $O(d^3 2^{2d} n)$ [13] and all maximal cliques in $O(d3^{d/3}n)$ [15]. All these algorithms exploit the degeneracy orientation of G. We exploit such orientation as well, but in a more systematic way and without listing explicitly all occurrences of the pattern; we exploit the structure of H, too, which results in richer bounds.

Concerning the structure of H, $\hom(H, G)$ can be computed in time $f(k) \cdot O(n^{\mathrm{t}(H)+1})$ [18], and $\text{sub}(H, G)$ can be computed in time $f(k) \cdot n^{c(H)+O(1)}$ where $c(H)$ is the vertex-cover number of H [24, 34, 3]. Our bounds are instead parameterized by a novel notion of width, $\tau(H)$, that is within constant factors of the independence number but gives tighter bounds.

[1] Formally, we should define a tree together with a mapping between its nodes and the subsets of V. To lighten the notation, we opt for a less formal definition where the nodes are themselves subsets of V.

Similarly, although several notions of tree decomposition for directed graphs exist [19], our dag tree decomposition is novel and different from all of them.

No lower bound in terms of d and of the structure of H, such as those we give, was known before. The existing lower bounds, based on the Exponential Time Hypothesis (ETH) [21], adopt the parameterizations mentioned above in terms of $\mathrm{t}(H)$ or $c(H)$; see [11] and [8, 9].

Manuscript organisation. Section 2 is a gentle and intuitive introduction to our approach. Section 3 introduces our dag tree decomposition, the dynamic programming for counting homomorphisms, and the corresponding running time bounds. Section 4 bounds the dag treewidth for several classes of patterns, leading to our faster algorithms. Finally, Section 5 proves our lower bounds. All proofs omitted due to space limitations can be found in [4].

2 Exploiting degeneracy orientations

We build the intuition behind our approach, starting from the classic algorithm for counting cliques of [10]. The algorithm begins by orienting G acyclically so that $\max_{v \in G} d_{\mathrm{out}}(v) \leq d$, which requires linear time. With G oriented acyclically, we take each $v \in G$ in turn and enumerate every subset of $(k-1)$ out-neighbors of v. In this way we can explicitly find all k-cliques of G in time $f(k) \cdot O(nd^{k-1}) = f(d,k) \cdot O(n)$. What makes the algorithm tick is the fact that an acyclically oriented clique has exactly one *source*, that is, a node with no incoming arcs. We would like to extend this approach to an arbitrary pattern H. Since every copy of H in G appears with *some* acyclic orientation, we can just take every possible acyclic orientation P of H, count the copies of P in G, and sum all the counts. Thus, we can reduce the problem to the following one: given a k-node dag P, and an n-node dag G with maximum outdegree d, count the copies of P in G.

Let us try a first approach. If P has $s = s(P)$ sources, we enumerate all the $\binom{n}{s} = O(n^s)$ ordered s-uples of V to which those sources can be mapped. For each such s-uple, we list the possible mappings of the remaining $k-s$ nodes, which can be done in time $O(d^{k-s})$ by listing the mappings of a fixed spanning forest of P. Finally, we check if the k nodes induce P in G. The total running time is $f(k) \cdot O(n^s d^{k-s})$. Unfortunately, if P is an independent set then $s = k$ and the running time is $O(n^k)$. The situation does not improve even if P is connected, as we can have $s = k-1$ (for the inward-oriented star).

Here our approach comes into play. We use the pattern P in Figure 1 as a toy example. Instead of listing all occurrences of P in G, we decompose P into two *pieces*, $P(1)$ and $P(2,3)$, where $P(u)$ denotes the subgraph of P reachable from u (that is, the transitive closure of u in P), and $P(u_1, \ldots, u_r) = \cup_{i=1}^{r} P(u_i)$. The idea is to compute the count of P by combining the counts of the two pieces, $P(1)$ and $P(2,3)$.

To simplify the task, we focus on counting the *homomorphisms* between P and G; we can then recover the number of induced copies by inclusion-exclusion arguments. In fact, we solve a slightly more complex problem: for each pair of nodes $x, y \in G$, we count the homomorphisms from P to G that map nodes 2 and 4 (see Figure 1) to x and y respectively. To recover $\hom(P,G)$ we then just sum over all pairs x, y. Formally, for a given pair x, y let $\phi : \{2,4\} \mapsto V$ be the map given by $\phi(2) = x$ and $\phi(4) = y$, and let $\hom(P,G,\phi)$ be the number of homomorphisms from P to G whose restriction to $\{2,4\}$ is ϕ. In the same way define $\hom(P(1),G,\phi)$ and $\hom(P(2,3),G,\phi)$. It is easy to see that:

$$\hom(P,G,\phi) = \hom(P(1),G,\phi) \cdot \hom(P(2,3),G,\phi) \tag{1}$$

To compute $\hom(P,G,\phi)$ we then just need $\hom(P(1),G,\phi)$ and $\hom(P(2,3),G,\phi)$. But we can compute $\hom(P(1),G,\phi)$ simultaneously for all ϕ in time $f(d,k) \cdot \tilde{O}(n)$, using our

listing technique to build a dictionary mapping each ϕ to its count. (The $\tilde{O}(\cdot)$ factor comes from the cost of accessing the dictionary, which has size $\text{poly}(n)$). Similarly, we can compute $\hom(P(2,3), G, \phi)$ in time $f(d,k) \cdot \tilde{O}(n^2)$. The total running time is $f(d,k) \cdot \tilde{O}(n^2)$, whereas our first approach would take time $f(d,k) \cdot O(n^3)$.

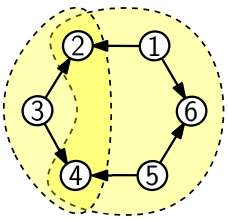


Figure 1 Toy example: an acyclic orientation P of $H = C_6$, decomposed into two pieces.

Abstracting from our toy example, we want to decompose P into a set of pieces $P_1, \ldots, P_\kappa$ with the following properties: (i) each piece P_i has a small number of sources $s(P_i)$, and (ii) we can obtain $\hom(P, G, \phi)$ by combining the homomorphism counts of the P_i. This is achieved precisely by the *dag tree decomposition*, which we introduce in Section 3. Like the tree decomposition of an undirected graph, the dag tree decomposition leads to a dynamic program to compute $\hom(P, G)$. The running time is $\tilde{O}(n^{\max_i s(P_i)})$, hence to make the algorithm useful we must show that a decomposition with "small" $\max_i s(P_i)$ always exists, which we do in Section 4.

3 DAG tree decompositions

Let $P = (V_P, A_P)$ be a directed acyclic graph. We denote by S_P, or simply S, the set of nodes of P having no incoming arc. These are the *sources* of P. We denote by $V_P(u)$ the transitive closure of u in P, i.e. the set of nodes of P reachable from u, and we let $P(u) = P[V_P(u)]$ be the corresponding subgraph of P. More generally, for a subset of sources $B \subseteq S$ we let $V_P(B) = \cup_{u \in B} V_P(u)$ and $P(B) = P[V_P(B)]$. Thus, $P(B)$ is the subgraph of P induced by all nodes reachable from any source in B. We call B a *bag* of sources. We can now formally introduce our decomposition.

▶ **Definition 10** (Dag tree decomposition). *Let $P = (V_P, A_P)$ be a dag. A dag tree decomposition (d.t.d.) of P is a rooted tree $T = (\mathcal{B}, \mathcal{E})$ with the following properties:*

1. *each node $B \in \mathcal{B}$ is a bag of sources $B \subseteq S_P$*
2. $\bigcup_{B \in \mathcal{B}} B = S_P$
3. *for all $B, B_1, B_2 \in T$, if $B \in T(B_1, B_2)$ then $V_P(B_1) \cap V_P(B_2) \subseteq V_P(B)$*

One can see immediately the resemblance to the standard tree decomposition of a graph (Definition 9). However, our dag tree decomposition differs crucially in two aspects. First, the bags of the tree are subsets of S rather than subsets of V_P. This is because the time needed to list the homomorphisms between $P(B_i)$ and G is driven by $|B_i|$, which is the number of sources in $P(B_i)$. Second, the path-intersection property concerns not the bags themselves, but the pieces reachable from the bags themselves. The reason is that, to combine the counts of two pieces together, their intersection must form a separator in G (similarly to what happens with the standard tree decomposition).

The dag tree decomposition induces immediately the following notions of *width*, that we use throughout the rest of the article.

▶ **Definition 11.** *The* width *of* T *is* $\tau(T) = \max_{B \in \mathcal{B}} |B|$. *The* dag treewidth $\tau(P)$ *of* P *is the minimum of* $\tau(T)$ *over all dag tree decompositions* T *of* P.

Figure 2 shows a pattern P, together with a d.t.d. T of width 1. Note that $1 \leq \tau(P) \leq k$ for any P. Note also that $\tau(P)$ has no relationship with the treewidth $\mathrm{t}(H)$ of H: they can both be $\Theta(k)$, or we can have $\tau(P) = 1$ but $\mathrm{t}(H) = k$ (when H is a clique). In fact, $\tau(P)$ is within constant factors of the independence number $\alpha(H)$ of H (see Section 4.3), and thus decreases as H becomes denser. The intuition is that adding arcs increases the number of nodes reachable from the sources, hence we need smaller bags to cover all of P. When H is a clique, P is reachable from just one source and $\tau(P) = 1$.

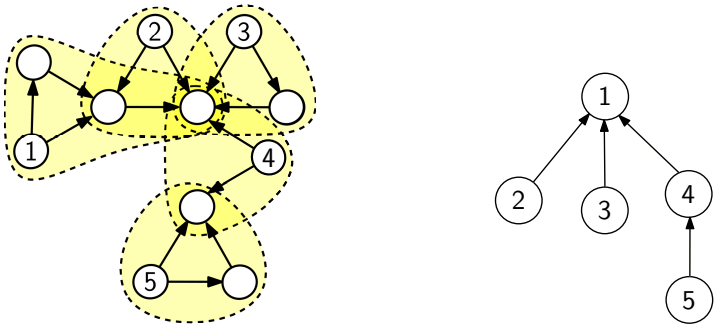


Figure 2 A dag P with one possible decomposition into five pieces (left), and one possible dag tree decomposition T for P (right). Since $\tau(T) = 1$, we can compute $\hom(P, G)$ in time $f(d, k) \cdot O(n)$.

3.1 Counting homomorphisms via dag tree decompositions

For any $B \in \mathcal{B}$ let $T(B)$ be the subtree of T rooted at B. We let $\Gamma[B]$ be the down-closure of B in T, that is, the union of all bags in $T(B)$. Consider $P(\Gamma[B])$, the subgraph of P induced by the nodes reachable from any $u \in \Gamma[B]$ (note the difference with $P(B)$, which contains only the nodes reachable from any $u \in B$). We compute $\hom(P(\Gamma[B]), G)$ in a bottom-up fashion over all B, starting with the leaves of T and moving towards the root. This is, in essence, the dynamic program given by the standard tree decomposition (see [18])[2].

As anticipated, we actually compute a refined count: $\hom(P(\Gamma[B]), \phi)$, the number of homomorphisms that extend a fixed mapping ϕ. Formally:

▶ **Definition 12.** *Let* $P_1 = (V_{P_1}, A_{P_1}), P_2 = (V_{P_2}, A_{P_2})$ *be two subgraphs of* P, *and let* $\phi_1 : P_1 \to G$ *and* $\phi_2 : P_2 \to G$ *be two homomorphisms. We say* ϕ_1 *and* ϕ_2 respect *each other if* $\phi_1(u) = \phi_2(u)$ *for all* $u \in V_{P_1} \cap V_{P_2}$. *We denote by* $\hom(P_1, G, \phi_2)$ *or simply* $\hom(P_1, \phi_2)$ *the number of homomorphisms from* P_1 *to* G *that respect* ϕ_2.

We can now present our main algorithmic result.

▶ **Lemma 13.** *Let* P *be any* k*-node dag, and* $T = (\mathcal{B}, \mathcal{E})$ *be a d.t.d. for* P. *Fix any* $B \in \mathcal{B}$ *as the root of* T. *There exists a dynamic programming algorithm HomCount*(P, T, B) *that computes* $\hom(P(\Gamma[B]), \phi_B)$ *for all* $\phi_B : P(B) \to G$ *in time* $f(d, k) \cdot \tilde{O}(n^{\tau(T)})$.

[2] The correctness of our dynamic program does not follow automatically from the dynamic program over standard tree decompositions. A proof of correctness is given in the full version of the manuscript.

Note that $\hom(P,G)$ is simply the sum of all the counts $\hom(P(\Gamma[B]),\phi_B)$ returned by the algorithm. Therefore, we can compute $\hom(P,G)$ in time $f(d,k)\cdot\tilde{O}(n^{\tau(T)})$. Now, a d.t.d. that minimises $\tau(T)$ can obviously be found in time $f(k)=O(f(d,k))$. Therefore:

▶ **Theorem 14.** *We can compute* $\hom(P,G)$ *in time* $f(d,k)\cdot\tilde{O}(n^{\tau(P)})$.

Equipped with Theorem 14, we can turn to the original problem of counting the copies of H.

3.2 Inclusion-exclusion arguments

We turn to computing $\hom(H,G)$, $\mathrm{sub}(H,G)$ and $\mathrm{ind}(H,G)$. We do so via standard inclusion-exclusion arguments, using our algorithm for computing $\hom(P,G)$ as a primitive. To this end, we shall define appropriate notions of width for undirected pattern graphs. Let $\Sigma(H)$ be the set of all dags P that can be obtained by orienting H acyclically. Let $\Theta(H)$ be the set of all equivalence relationships on V_H, and for $\theta\in\Theta$ let H/θ be the pattern obtained from H by identifying equivalent nodes according to θ and removing loops and multiple edges. Let $D(H)$ be the set of all supergraphs of H (including H) on the same node set V_H.

▶ **Definition 15.** *The* dag treewidth *of* H *is* $\tau(H)=\tau_3(H)$*, where:*

$$\tau_1(H)=\max\{\tau(P):P\in\Sigma\} \tag{2}$$
$$\tau_2(H)=\max\{\tau_1(H/\theta):\theta\in\Theta\} \tag{3}$$
$$\tau_3(H)=\max\{\tau_2(H'):H'\in D(H)\} \tag{4}$$

We can then state:

▶ **Theorem 16.** *One can compute:*
- $\hom(H,G)$ *in time* $f(d,k)\cdot\tilde{O}(n^{\tau_1(H)})$,
- $\mathrm{sub}(H,G)$ *in time* $f(d,k)\cdot\tilde{O}(n^{\tau_2(H)})$,
- $\mathrm{ind}(H,G)$ *in time* $f(d,k)\cdot\tilde{O}(n^{\tau(H)})$.

The algorithmic part of our work is complete. We shall now focus on bounding $\tau_1(H)$, $\tau_2(H)$, and $\tau(H)$, so to instantiate Theorem 16 and prove the upper bounds of Section 1.1.

4 Bounds on the dag treewidth

In this section we develop upper bounds on $\tau_1(H),\tau_2(H),\tau(H)$ as a function of H. First, we bound $\tau(H)$ for cliques minus ϵ edges, obtaining a generalization of the classic clique counting bound of [10]. Then, we bound $\tau_2(H)$ for complete multipartite graphs plus ϵ edges, obtaining a generalization of a result by Eppstein [13]. Next, we show that $\Omega(\alpha(H))\leq\tau(H)\leq\alpha(H)$. Finally, we show that $\tau(H)\leq\lfloor\frac{k}{4}\rfloor+2$ for *every* pattern H, including disconnected ones. This requires a nontrivial proof.

Before proceeding, we need some definitions. We say a node $v\in P$ is a *joint* if it is reachable from two or more sources, i.e. if $v\in V_P(u)\cap V_P(u')$ for some $u,u'\in S$ with $u\neq u'$. We write J_P or simply J for the set of all joints of P. We write $J(u)$ for the set of joints reachable from u, and for any $X\subseteq V_P$ we let $J(X)=\cup_{u\in X}J(u)$.

4.1 Quasi-cliques

▶ **Lemma 17.** *If* H *has* $\binom{k}{2}-\epsilon$ *edges then* $\tau(H)\leq\lceil\frac{1}{2}+\sqrt{\frac{\epsilon}{2}}\rceil$.

Proof. The source set $|S|$ of P is an independent set, hence $|E_H| \leq \binom{k}{2} - \binom{|S|}{2}$. Therefore $\epsilon \geq \binom{|S|}{2}$, which implies $|S| \leq 1 + \sqrt{2\epsilon}$. A d.t.d. for P is the tree on two bags B_1, B_2 that satisfy $B_1 \cup B_2 = S$, $|B_1| = \lfloor |S|/2 \rfloor$, and $|B_2| = \lceil |S|/2 \rceil$. Hence $\tau(P) \leq \lceil |S|/2 \rceil \leq \lceil \frac{1}{2} + \sqrt{\frac{\epsilon}{2}} \rceil$. Now consider any H' obtained from H by adding edges or identifying nodes. Obviously $|E_{H'}| \geq \binom{k'}{2} - \epsilon$ where $k' = |V_{H'}|$, and the argument above implies $\tau(P') \leq \lceil \frac{1}{2} + \sqrt{\frac{\epsilon}{2}} \rceil$ for any orientation P' of H'. By Definition 15, then, $\tau(H) \leq \lceil \frac{1}{2} + \sqrt{\frac{\epsilon}{2}} \rceil$. ◀

By Theorem 16 and since $\tau_1(H) \leq \tau_2(H) \leq \tau(H)$ it follows that we can compute $\hom(H, G)$, $\mathrm{sub}(H, G)$ and $\mathrm{ind}(H, G)$ in time $f(d, k) \cdot \tilde{O}(n^{\lceil \frac{1}{2} + \sqrt{\frac{\epsilon}{2}} \rceil})$, proving Theorem 4.

4.2 Quasi-multipartite graphs (non-induced)

▶ **Lemma 18.** *If H is a complete multipartite graph, then $\tau_2(H) = 1$. If H is a complete multipartite graph plus ϵ edges, then $\tau_2(H) \leq \lfloor \frac{\epsilon}{4} \rfloor + 2$.*

Proof. Suppose $H = (V_H, E_H)$ is a complete multipartite graph. Let $V_H = V_H^1 \cup \ldots \cup V_H^\kappa$ where each $H[V_H^j]$ is a maximal independent set. Note that, in any orientation P of H, all sources are contained in exactly one V_H^j. Moreover, $V_P(u) = V_P(u')$ for any two sources u, u'. A d.t.d. T of width $\tau(T) = 1$ is the trivial one with one source per bag.

Suppose then we add ϵ edges to H. Again, in any orientation P of H, all sources are contained in exactly one V_H^j, but we might have $V_P(u) \neq V_P(u')$ for two distinct sources u, u'. Note however that all nodes in $V_H \setminus V_H^j$ are reachable from all sources and are thus irrelevant to a d.t.d.. More precisely, any d.t.d. for $P[V_H^j]$ is a d.t.d. for P. But $P[V_H^j]$ has at most ϵ edges and by Theorem 20 (see below) it has a d.t.d. of width at most $\lfloor \frac{\epsilon}{4} \rfloor + 2$.

This arguments apply also to any pattern H' obtained by identifying nodes of H: if there is a source node u in H' that in H is in V_H^j, then every node of H' that in H is in $V_H \setminus V_H^j$ is reachable from u. In addition, if a node in $V_H \setminus V_H^j$ has been identified with a node in V_H^j then all nodes are reachable from all sources and there is a trivial d.t.d. of width 1. Otherwise, in H' the nodes of V_H^j have been identified with a subset of V_H^j itself and we just need a d.t.d. of width at most $\lfloor \frac{\epsilon}{4} \rfloor + 2$ as above. ◀

By Theorem 16, if H is complete multipartite then we can compute $\hom(H, G)$ and $\mathrm{sub}(H, G)$ in time $f(d, k) \cdot \tilde{O}(n)$. If instead H is complete multipartite plus ϵ edges, then we can compute $\hom(H, G)$ and $\mathrm{sub}(H, G)$ in time $f(d, k) \cdot \tilde{O}(n^{\lfloor \frac{\epsilon}{4} \rfloor + 2})$. This proves Theorem 5.

4.3 Independence number and dag treewidth

▶ **Lemma 19.** *Any k-node graph H satisfies $\Omega(\alpha(H)) \leq \tau(H) \leq \alpha(H)$.*

Proof. Let H be any pattern graph on k nodes. For the upper bound, note that in any acyclic orientation P of H the sources form an independent set, and that $\alpha(H') \leq \alpha(H)$ for any H' obtained by adding edges or identifying nodes of H.

For the lower bound, we exhibit a pattern H' obtained by adding edges to H such that $\tau(P) = \Omega(\alpha(H))$ for all its acyclic orientations P. Let $I \subseteq V_H$ be an independent set of H with $|I| = \Omega(\alpha(H))$ and $|I| \bmod 5 \equiv 0$. Partition I in I_1, I_2 where $|I_1| = \frac{2}{5}|I|$ and $|I_2| = \frac{3}{5}|I|$. On top of I_1 we virtually build a 3-regular expander $\mathcal{E} = (I_1, E_\mathcal{E})$ of linear treewidth $t(\mathcal{E}) = \Omega(|I_1|)$. It is well known that such expanders exist (see e.g. Proposition 1 and Theorem 5 of [20]). For each edge $uv \in E_\mathcal{E}$ we choose a distinct node $e_{uv} \in I_2$ and add to $H[I]$ the edges $e_{uv}u$ and $e_{uv}v$. In words, $H[I]$ is the 1-subdivision of $\mathcal{E}$. Let H' be the resulting pattern. Note that $t(\mathcal{E}) = \Omega(|I_1|) = \Omega(|I|) = \Omega(\alpha(H))$.

Let now $P = (V_P, A_P)$ be any acyclic orientation of H' having I_2 as source set, and let T be any d.t.d. of P. We show that $\tau(T) \geq \frac{1}{2}(\mathrm{t}(\mathcal{E}) + 1)$. To this end, we build a tree D by replacing each bag $B \in T$ with the bag $J(B)$. We can show that D is a tree decomposition of $\mathcal{E}$ (see Definition 9). First, by point (2) of Definition 10 we have $\cup_{B \in T} B = S_P$. It follows that $\cup_{J(B) \in D} J(B) = I_1$. This proves property (1). Second, by construction for every $e \in E_{\mathcal{E}}$ we have a node $u = u_e \in V_P$. Then, again by point (2) of Definition 10, there is $B \in T$ such that $u \in B$; and by construction of D it holds $e = \{v, w\} \subseteq J(B)$. This proves property (2). Third, fix any $J(B_1), J(B_2), J(B_3) \in D$ such that $J(B_1)$ is on the path from $J(B_2)$ to $J(B_3)$ in D, and consider any $v \in J(B_2) \cap J(B_3)$. There is $u \in B_2$ such that $v \in J(u)$, and $u' \in B_3$ such that $z \in J(u')$. Thus $v \in J(u) \cap J(u') \subseteq J(B_2) \cap J(B_3)$; but since B_1 is on the path from B_2 to B_3 in T, point (3) of Definition 9 implies $v \subseteq J(B_1)$. This proves property (3). Hence D is a tree decomposition of $\mathcal{E}$. Finally, by construction $|I_2| \leq 2|J(I_2)|$. Then by Definition 9 and Definition 11 we have $\mathrm{t}(\mathcal{E}) \leq 2\tau(P) - 1$, that is, $\tau(P) \geq \frac{1}{2}(\mathrm{t}(\mathcal{E}) + 1)$. But $t(\mathcal{E}) = \Omega(\alpha(H))$, thus $\tau(P) = \Omega(\alpha(H))$. ◀

4.4 All patterns

This subsection is devoted entirely to prove:

▶ **Theorem 20.** *For any dag $P = (V_P, A_P)$, in time $O(1.7549^k)$ we can compute a dag tree decomposition $T = (\mathcal{B}, \mathcal{E})$ with $\tau(T) \leq \min(\lfloor \frac{e}{4} \rfloor, \lfloor \frac{k}{4} \rfloor) + 2$, where $k = |V_P|$ and $e = |A_P|$.*

Combined with Definition 15, this gives:

▶ **Corollary 21.** *Any k-node graph H satisfies $\tau(H) \leq \lfloor \frac{k}{4} \rfloor + 2$.*

The proof of Theorem 20 proceeds as follows. First, we greedily find a subset $B^* \subseteq S$ such that $|V_P(B^*)| \geq 4|B^*|$. If this subset coincides with S, we are done. Otherwise we delete B^* and $V_P(B^*)$ from P, partitioning P itself in $\ell \geq 1$ connected components. We can easily show that the d.t.d.'s of the individual components can be combined into a d.t.d. for P, if we add B^* to every bag. The crux, then, is bounding the dag treewidth of the individual components. We show that, if the i-th component has k_i nodes, then it admits a d.t.d. of width $\frac{k_i}{4} + 2$. This requires to first "peel" the component, getting rid of the tree-like parts, and then decomposing its core using tree decompositions.

The proof makes heavy use of the *skeleton* of P, defined as follows.

▶ **Definition 22.** *The* skeleton *of a dag $P = (V_P, A_P)$ is the directed bipartite graph $\Lambda(P) = (S \cup J, E_\Lambda)$ where $E_\Lambda \subseteq S \times J$ and $(u, v) \in E_\Lambda$ if and only if $v \in J(u)$.*

Figure 3 gives an example. Note that $\Lambda(P)$ does not contain nodes that are neither sources nor joints; the reason is that those nodes are irrelevant to the construction of a d.t.d.. Note also that building $\Lambda(P)$ takes time $O(\mathrm{poly}(k))$.

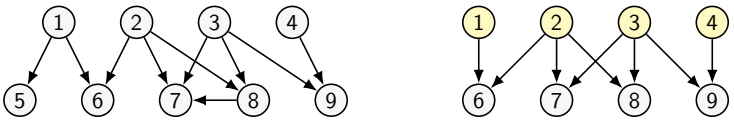


Figure 3 Left: a dag P. Right: its skeleton $\Lambda(P)$ (sources S above, joints J below).

Let us now delve into the proof. For any node x, we denote by d_x the current degree of x in the skeleton.

1. Shattering the skeleton. Set $B^{(0)} = \emptyset$ and let $\Lambda^{(0)} = (S^{(0)} \cup J^{(0)}, E^{(0)}_\Lambda)$ be a copy of Λ. Set $j = 0$ and proceed iteratively as follows. If there is a source $u \in S^{(j)}$ with $d_u \geq 3$, let $B^{(j+1)} = B^{(j)} \cup \{u\}$, and let $\Lambda^{(j+1)} = (S^{(j+1)} \cup J^{(j+1)}, E^{(j+1)}_\Lambda)$ be obtained from $\Lambda^{(j)}$ by removing u and $J^{(j)}(u)$; otherwise stop. Suppose the procedure stops at $j = j^*$, producing the subset $B^* = B^{(j^*)}$ and the sub-skeleton $\Lambda^* = \Lambda^{(j^*)} = (S^* \cup J^*, E^*_\Lambda)$. We prove:

▶ **Lemma 23.** *$|B^*| \leq \min\left(\frac{k-|\Lambda^*|}{4}, \frac{e-|E^*_\Lambda|}{4}\right)$, where $k = |V_P|$ and $e = |A_P|$.*

Proof. Since each step removes at least 4 nodes from $\Lambda^{(j)}$, then $4|B^*| \leq |\Lambda \setminus \Lambda^*| \leq (k - |\Lambda^*|)$, and $|B^*| \leq \frac{k-|\Lambda^*|}{4}$. Now consider the nodes $\{u\} \cup J^{(j)}(u)$ removed at step j. Note that $\Lambda^{(j)}$ is just the skeleton of $P^{(j)} = P \setminus P(B^{(j)})$, and that $J^{(j)}(u) \subseteq P^{(j)}(u)$. This implies $P^{(j)}(u)$ contains at least 3 arcs. Moreover, there must be at least one arc from $P^{(j)} \setminus P^{(j)}(u)$ to $P^{(j)}(u)$, otherwise $P^{(j)}(u)$ would not contain joints of $P^{(j)}$. We have therefore at least 4 arcs pointing to nodes of $P^{(j)}(u)$. These arcs are counted only once, since $P^{(j)}(u)$ is then removed from $P^{(j)}$. Hence $e \geq 4|B^*| + |E^*_\Lambda|$, and $|B^*| \leq \frac{e-|E^*_\Lambda|}{4}$. ◀

Now, if $B^* = S$, then obviously $T = (\{B^*\}, \emptyset)$ is a d.t.d. of P. Moreover in this case Λ^* is empty, so Lemma 23 gives $\tau(T) = |B^*| \leq \min(\frac{k}{4}, \frac{e}{4})$, proving Theorem 20.

Suppose instead $B^* \subset S$. Let $P^* = P \setminus P(B^*)$, and let $\Lambda_i = (S_i \cup J_i, E_i) : i = 1, \ldots, \ell$ be the connected components of Λ^*. One can check that Λ_i is precisely the skeleton of $P^*(S_i)$, the i-th connected component of P^*. As the next lemma says, we can obtain a d.t.d. for P by simply combining the d.t.d.'s of the $P^*(S_i)$ in a tree and adding B^* to all the bags. For simplicity, we say "a d.t.d of Λ_i" for "a d.t.d. of $P^*(S_i)$".

▶ **Lemma 24.** *For $i = 1, \ldots, \ell$ let $T_i = (\mathcal{B}_i, \mathcal{E}_i)$ be a d.t.d. of Λ_i. Consider the tree T obtained as follows. The root of T is the bag B^*, and the subtrees below B^* are $T_1, \ldots, T_\ell$, where each bag $B \in T_i$ has been replaced by $B \cup B^*$. Then $T = (\mathcal{B}, \mathcal{E})$ is a d.t.d. of P with $\tau(T) \leq |B^*| + \max_{i=1,\ldots,\ell} \tau(T_i)$ and $|\mathcal{B}| = 1 + \sum_{i=1}^{\ell} |\mathcal{B}_i|$.*

Proof. The claims on $\tau(T)$ and $|\mathcal{B}|$ are trivial. Let us check via Definition 10 that T is a d.t.d. of P. Point (1) is immediate. For point (2), note that $\cup_{B \in \mathcal{B}_i} = S_i$ because T_i is by hypothesis a d.t.d. of Λ_i; by construction, then, $\cup_{B \in \mathcal{B}} = B^* \cup_{i=1}^{\ell} S_i = S_P$. Now point (3). Choose any two bags $B' \cup B^*$ and $B'' \cup B^*$ of T, where $B' \in T_i$ and $B'' \in T_j$ for some $i, j \in \{1, \ldots, \ell\}$, and any bag $B \cup B^* \in T(B' \cup B^*, B'' \cup B^*)$. Suppose first $i = j$; thus by construction $B \in T(B', B'')$. Since T_i is a d.t.d., then $J_i(B') \cap J_i(B'') \subseteq J_i(B)$, and in T this implies $V_P(B' \cup B^*) \cap V_P(B'' \cup B^*) \subseteq V_P(B \cup B^*)$. Suppose instead $i \neq j$. Thus $J_i(S_i) \cap J_j(S_j) = \emptyset$ and this means that $J(S_i) \cap J(S_j) \subseteq J(B^*)$. But $V_P(B_i) \cap V_P(B_j) \subseteq J(S_i) \cap J(S_j)$ and $J(B^*) \subseteq V_P(B^*)$, thus $V_P(B_i) \cap V_P(B_j) \subseteq V_P(B^*)$. It follows that for every bag $B \cup B^*$ of T we have $V_P(B_i \cup B^*) \cap V_P(B_j \cup B^*) \subseteq V_P(B \cup B^*)$. ◀

2. Peeling Λ_i. We now remove the tree-like parts of $\Lambda_i = (S_i \cup J_i, E_i)$; for instance, sources that point to only a single joint. The intuition is that those parts do not increase the dag treewidth. As a base case, if $|S_i| = 1$ then $T_i = (\{S_i\}, \emptyset)$ is a d.t.d. for Λ_i of width 1. Suppose instead $|S_i| > 1$. Note that every $u \in S_i$ satisfies $d_u = |J_i(u)| \leq 2$, for otherwise u would have been removed in the previous phase. Consider the following conditions:

1. $\exists u \in S_i : d_u = 1$. Then fix any such u, and fix any $u' \in S_i \setminus \{u\}$ with $J_i(u) \cap J_i(u') \neq \emptyset$.
2. $J_i(u) = J_i(u')$ for some $u, u' \in S_i$ with $u \neq u'$. Then, fix u and u' as above.
3. $\exists v \in J_i : d_v = 1$ (this is initially false). Then fix any such v, let u be the unique source such that $v \in J_i(u)$, and let $u' \neq u$ be any source with $J_i(u) \cap J_i(u') \neq \emptyset$.

Note that, in any case, u' must exist since $|S_i| > 1$ and Λ_i is always connected. We then "peel" Λ_i by defining T_i recursively, as follows. Let $\Lambda'_i = \Lambda_i \setminus \{u\}$, and assume we have a d.t.d. T'_i of Λ'_i. Since $u' \neq u$ then $u' \in S_i \setminus \{u\}$, and thus for some $B' \in T'_i$ we have $u' \in B'$. Create the bag $B_u = \{u\}$ and set it as a child of B'. We obtain a tree T_i where B_u is a leaf; and note that, by construction, for any $u'' \in S_i \setminus \{u, u'\}$ we have $J_i(u) \cap J_i(u'') \subseteq J_i(u')$. This implies that T_i is a d.t.d. for Λ_i. Then remove u from Λ_i, as well as any $v : d_v = 0$.

We repeat this peeling process until we meet the base case, or until $|S_i| > 1$ and all three conditions above fail. In the latter case, we move to the next phase.

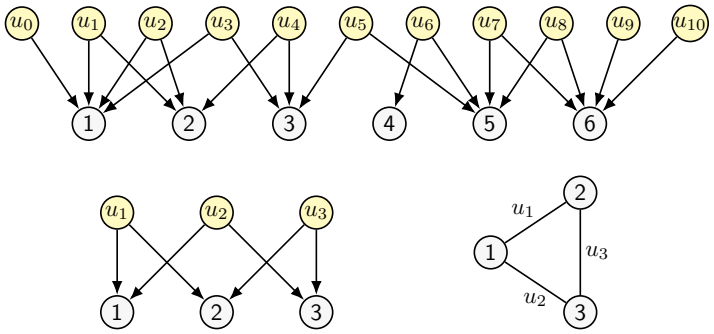


Figure 4 Above: example of a skeleton component Λ_i. Below: the core $\Lambda_i^\bullet$ obtained from Λ_i after peeling (left), and its encoding as C_i (right).

3. Decomposing the core. We denote by $\Lambda_i^\bullet = (S_i^\bullet \cup J_i^\bullet, E_i^\bullet)$ the subgraph of Λ_i left after the peeling. We say $\Lambda_i^\bullet$ is the *core* of Λ_i; intuitively, it is the part determining the dag treewidth of Λ_i. Now, since $\Lambda_i^\bullet$ violates all three conditions of the peeling step, certainly $d_u = 2$ for every source u and $d_v \geq 2$ for every joint v. This means that the joints and sources of $\Lambda_i^\bullet$ can be represented as nodes and edges of a simple graph. Formally, we encode $\Lambda_i^\bullet$ as $C_i = (V_{C_i}, E_{C_i})$ where $V_{C_i} = J_i^\bullet$ and $E_{C_i} = \{e_u : u \in S_i^\bullet\}$, as Figure 4 shows.

Using C_i, we can find a good bound on $\tau(\Lambda_i^\bullet)$ via tree decompositions. The key fact is that any tree decomposition for C_i of width t can be turned in time $poly(k)$ into a d.t.d. for $\Lambda_i^\bullet$ of width $t+1$ (intuitively, the tree decomposition covers the edges of C_i, which are the sources of $\Lambda_i^\bullet$). By a bound of [23], C_i admits a tree decomposition of width at most $\frac{|E_{C_i}|}{5} + 2$, and this can be computed in time $O(1.7549^k)$ [18]. In the end, this yields:

▶ **Lemma 25.** *Let $k_i = |S_i^\bullet \cup J_i^\bullet|$. In time $O(1.7549^{k_i})$ we can compute a d.t.d. $T_i^\bullet = (\mathcal{B}_i^\bullet, \mathcal{E}_i^\bullet)$ of $\Lambda_i^\bullet$ such that $\tau(T_i^\bullet) \leq \lfloor \frac{k_i}{4} \rfloor + 2$.*

With Lemma 25 we have essentially finished. It remains to wrap all our bounds together.

4. Assembling the tree. Let T_i be the d.t.d. for Λ_i, as returned by the recursive peeling followed by the core decomposition. Note that $\tau(T_i) \leq \tau(T_i^\bullet)$, since the peeling phase only add bags of width 1. Therefore, by Lemma 25, $\tau(T_i) \leq \lfloor \frac{k_i}{4} \rfloor + 2$ where $k_i = |S_i^\bullet \cup J_i^\bullet|$.

Let now $T = (\mathcal{B}, \mathcal{E})$ be the d.t.d. for P obtained by composing $T_1, \ldots, T_\ell$ (Lemma 24). By Lemma 24 itself, $\tau(T) \leq |B^*| + \max_{i=1,\ldots,\ell} \tau(T_i)$, thus:

$$\tau(T) \leq |B^*| + \max_{i=1,\ldots,\ell} \lfloor \frac{k_i}{4} \rfloor + 2 \tag{5}$$

Now, from Lemma 23 we know that $P(B^*)$ has at least $4|B^*|$ nodes and $4|B^*|$ arcs. Similarly, since each Λ_i has at least k_i nodes and k_i arcs, then $P \setminus P(B^*)$ has at least $\sum_{i=1,\ldots,\ell} k_i$

nodes and $\sum_{i=1,\dots,\ell} k_i$ arcs. By a simple summation, then, we have $\tau(T) \leq \lfloor \frac{k}{4} \rfloor + 2$ and $\tau(T) \leq \lfloor \frac{e}{4} \rfloor + 2$, hence $\tau(T) \leq \min(\lfloor \frac{k}{4} \rfloor, \lfloor \frac{e}{4} \rfloor) + 2$. Finally, by Lemma 25 the time to build T_i is $O(1.7549^{k_i})$, since the peeling phase clearly takes time $\text{poly}(k_i)$. The total time to build T is therefore $O(1.7549^k)$, which concludes the proof of Theorem 20.

5 Lower bounds

We prove the lower bound of Theorem 7. Note that, since $\tau(H) = \Theta(\alpha(H))$ by Lemma 19, the bound still holds if one replaces $s(H)$ by $\alpha(H)$ in the statement.

▶ **Theorem 26.** *For any function $a : [k] \to [1, k]$ there exists an infinite family of patterns $\mathcal{H}$ such that (1) $s(H) = \Theta(a(k))$ for each $H \in \mathcal{H}$, and (2) if there exists an algorithm that computes $\text{ind}(H, G)$ or $\text{sub}(H, G)$ in time $f(d, k) \cdot n^{o(a(k)/\ln a(k))}$ for all $H \in \mathcal{H}$ where d is the degeneracy of G, then ETH fails.*

Proof. We reduce counting cycles in an arbitrary graph to counting a gadget pattern on k nodes and dag treewidth $O(\tau(k))$, where $\tau(k) = a(k)$, in a d-degenerate graph.

The gadget is the following. Consider a simple cycle on $k_0 \geq 3$ nodes. Choose an integer $d = d(k) \geq 2$ with $d(k) \in \Omega(\frac{k}{\tau(k)})$. For each edge $e = uv$ of the cycle create a clique C_e on $d - 1$ nodes; delete e and connect both u and v to every node of C_e. The resulting pattern H has $dk_0 = k$ nodes. Let us prove $\tau(H) \leq k_0$. This implies $\tau(H) = O(\tau(k))$ since $k_0 = \frac{k}{d} \in O(\tau(k))$. Consider again the generic edge $e = uv$. Since $C_e \cup u$ is itself a clique, it has independent set size 1; and thus in any orientation H_σ of H, $C_e \cup u$ contains at most one source. Applying the argument to all e shows $S(H_\sigma) \leq k_0$, and since $\tau(H_\sigma) \leq |S(H_\sigma)|$, we have $\tau(H_\sigma) \leq k_0$. Note any H'_σ obtained from H_σ by adding edges or identifying nodes has at most k_0 roots, too. Hence $\tau(H) \leq k_0$.

Now consider the task of counting the cycles of length $k_0 \geq 3$ in a simple graph G_0 on n_0 nodes and m_0 edges. We replace each edge of G_0 as described above. The resulting graph G has $n = m_0(d - 1) + n_0 = O(dn_0^2)$ nodes, has degeneracy d, and can be built in $\text{poly}(n_0)$ time. Note that every k_0-cycle of G_0 is univocally associated to a(n induced) copy of H in G. Suppose then there exists an algorithm that computes $\text{ind}(H, G)$ or $\text{sub}(H, G)$ in time $f(d, k) \cdot n^{o(\tau(H)/\ln \tau(H))}$. Since $\tau(H) \leq k_0$, $k = f(d, k_0)$, $n = O(dn_0^2)$, and $d = f(k_0)$, the running is time $f(d, k_0) \cdot n^{o(k_0/\ln k_0)}$. This implies one can count the number of k_0-cycles in G in time $f(k_0) \cdot n^{o(k_0/\ln k_0)}$. The proof is completed by invoking:

▶ **Theorem 27** ([11], Theorem I.2)**.** *The following problems are $\#W[1]$-hard and, assuming ETH, cannot be solved in time $f(k) \cdot n^{o(k/\log k)}$ for any computable function f: counting (directed) paths or cycles of length k, and counting edge-colorful or uncolored k-matchings in bipartite graphs.* ◀

6 Conclusions

We have shown how one can exploit the sparsity of a graph to count subgraphs faster than with state-of-the-art algorithms. Our main technical ingredient, the dag tree decomposition, not only yields better algorithms, but sheds light on the algorithmic role of degeneracy in subgraph counting, too. It would be interesting to know if our decomposition can be applied to problems other than subgraph counting.

An obvious line of future research is to tighten the bounds. For all patterns, one could improve the upper bound by reducing the exponent by constant or logarithmic factors; larger improvements seem unlikely, due to the lower bounds. For special classes of patterns, instead,

the situation is different: our lower bounds hold for *some* infinite family of patterns, not for *any* infinite family. This leaves open the question of finding special classes of patterns that can be counted even faster, or of tightening the lower bounds. In the second case, the dag treewidth would completely characterise the complexity of subgraph counting when parameterized by the degeneracy of the host graph.

References

1 N. Alon, P. Dao, I. Hajirasouliha, F. Hormozdiari, and S. C. Sahinalp. Biomolecular network motif counting and discovery by color coding. *Bioinformatics*, 24(13):i241–249, July 2008. doi:10.1093/bioinformatics/btn163.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

3 Andreas Björklund, Petteri Kaski, and Łukasz Kowalik. Counting Thin Subgraphs via Packings Faster Than Meet-in-the-middle Time. In *Proc. of ACM-SIAM SODA*, pages 594–603, 2014. doi:10.1137/1.9781611973402.45.

4 Marco Bressan. Faster subgraph counting in sparse graphs. *CoRR*, abs/1805.02089, 2018. arXiv:1805.02089.

5 Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. Counting Graphlets: space vs. time. In *Proc. of ACM WSDM*, pages 557–566, 2017. doi:10.1145/3018661.3018732.

6 Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. Motif Counting Beyond Five Nodes. *ACM Transactions on Knowledge Discovery from Data*, 20(2):1–25, April 2018. doi:10.1145/3186586.

7 Marco Bressan, Stefano Leucci, and Alessandro Panconesi. Motivo: fast motif counting via succinct color coding and adaptive sampling. *Proc. of the VLDB Endowment*, 12(11):1651–1663, July 2019. doi:10.14778/3342263.3342640.

8 Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation*, 201(2):216–231, 2005. doi:10.1016/j.ic.2005.05.001.

9 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006. doi:10.1016/j.jcss.2006.04.007.

10 Norishige Chiba and Takao Nishizeki. Arboricity and Subgraph Listing Algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985. doi:10.1137/0214017.

11 Radu Curticapean and Dániel Marx. Complexity of Counting Subgraphs: Only the Boundedness of the Vertex-Cover Number Counts. In *Proc. of IEEE FOCS*, pages 130–139, Washington, DC, USA, 2014. IEEE Computer Society. doi:10.1109/FOCS.2014.22.

12 Reinhard Diestel. *Graph Theory*. Springer Publishing Company, Incorporated, 5th edition, 2017. doi:10.1007/978-3-662-53622-3.

13 David Eppstein. Arboricity and Bipartite Subgraph Listing Algorithms. *Inf. Process. Lett.*, 51(4):207–211, August 1994. doi:10.1016/0020-0190(94)90121-X.

14 David Eppstein. Subgraph Isomorphism in Planar Graphs and Related Problems. *Journal of Graph Algorithms and Applications*, 3(3):1–27, 1999. doi:10.7155/jgaa.00014.

15 David Eppstein, Maarten Löffler, and Darren Strash. Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time. In *Algorithms and Computation*, pages 403–414. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-17517-6_36.

16 David Eppstein and Darren Strash. Listing All Maximal Cliques in Large Sparse Real-World Graphs. In *Experimental Algorithms*, pages 364–375. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-20662-7_31.

17 Peter Floderus, Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Detecting and Counting Small Pattern Graphs. *SIAM J. Discrete Math.*, 29(3):1322–1339, 2015. doi:10.1137/140978211.

18 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer-Verlag Berlin Heidelberg, 1 edition, 2010. doi:10.1007/978-3-642-16533-7.

19 Robert Ganian, Petr Hliněný, Joachim Kneis, Daniel Meister, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Are there any good digraph width measures? *Journal of Combinatorial Theory, Series B*, 116:250–286, January 2016. doi:10.1016/j.jctb.2015.09.001.

20 Martin Grohe and Dániel Marx. On tree width, bramble size, and expansion. *Journal of Combinatorial Theory, Series B*, 99(1):218–228, 2009. doi:10.1016/j.jctb.2008.06.004.

21 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? In *Proc. of IEEE FOCS*, pages 653–662, 1998. doi:10.1109/sfcs.1998.743516.

22 Shweta Jain and C. Seshadhri. A Fast and Provable Method for Estimating Clique Counts Using Turán's Theorem. In *Proc. of WWW*, pages 441–449, 2017. doi:10.1145/3038912.3052636.

23 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. Algorithms Based on the Treewidth of Sparse Graphs. In *Graph-Theoretic Concepts in Computer Science*, pages 385–396. Springer Berlin Heidelberg, 2005. doi:10.1007/11604686_34.

24 Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and Detecting Small Subgraphs via Equations. *SIAM Journal on Discrete Mathematics*, 27(2):892–909, January 2013. doi:10.1137/110859798.

25 François Le Gall. Powers of Tensors and Fast Matrix Multiplication. In *Proc. of ISSAC*, pages 296–303, 2014. doi:10.1145/2608628.2608664.

26 Silviu Maniu, Pierre Senellart, and Suraj Jog. An Experimental Study of the Treewidth of Real-World Graph Data. In *Proc. of ICDT*, pages 12:1–12:18, 2019. doi:10.4230/LIPIcs.ICDT.2019.12.

27 J. Nešetřil and P.O. de Mendez. *Sparsity: Graphs, Structures, and Algorithms*. Algorithms and Combinatorics. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-27875-4.

28 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 026(2):415–419, 1985. URL: http://eudml.org/doc/17394.

29 Viresh Patel and Guus Regts. Computing the Number of Induced Copies of a Fixed Graph in a Bounded Degree Graph. *Algorithmica*, 81(5):1844–1858, September 2018. doi:10.1007/s00453-018-0511-9.

30 Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. ESCAPE: Efficiently counting all 5-vertex subgraphs. In *Proc. of WWW*, pages 1431–1440, 2017. doi:10.1145/3038912.3052597.

31 Ahmet Erdem Sariyüce and Ali Pinar. Peeling Bipartite Networks for Dense Subgraph Discovery. In *Proc. of ACM WSDM*, pages 504–512, 2018. doi:10.1145/3159652.3159678.

32 Ahmet Erdem Sariyüce, C. Seshadhri, Ali Pinar, and Ümit V. Çatalyürek. Nucleus Decompositions for Identifying Hierarchy of Dense Subgraphs. *ACM Trans. Web*, 11(3):16:1–16:27, July 2017. doi:10.1145/3057742.

33 Charalampos E. Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. Scalable Motif-aware Graph Clustering. In *Proc. of WWW*, pages 1451–1460, 2017. doi:10.1145/3038912.3052653.

34 Virginia Vassilevska Williams and Ryan Williams. Finding, Minimizing, and Counting Weighted Subgraphs. *SIAM J. Comput.*, 42(3):831–854, 2013. doi:10.1137/09076619X.

35 Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. Local Higher-Order Graph Clustering. In *Proc. of ACM KDD*, pages 555–564, 2017. doi:10.1145/3097983.3098069.

Towards a Theory of Parameterized Streaming Algorithms

Rajesh Chitnis
School of Computer Science, University of Birmingham, UK
rajeshchitnis@gmail.com

Graham Cormode
University of Warwick, UK
g.cormode@warwick.ac.uk

Abstract

Parameterized complexity attempts to give a more fine-grained analysis of the complexity of problems: instead of measuring the running time as a function of only the input size, we analyze the running time with respect to additional parameters. This approach has proven to be highly successful in delineating our understanding of NP-hard problems. Given this success with the TIME resource, it seems but natural to use this approach for dealing with the SPACE resource. First attempts in this direction have considered a few individual problems, with some success: Fafianie and Kratsch [MFCS'14] and Chitnis et al. [SODA'15] introduced the notions of streaming kernels and parameterized streaming algorithms respectively. For example, the latter shows how to refine the $\Omega(n^2)$ bit lower bound for finding a minimum Vertex Cover (VC) in the streaming setting by designing an algorithm for the parameterized k-VC problem which uses $O(k^2 \log n)$ bits.

In this paper, we initiate a systematic study of graph problems from the paradigm of parameterized streaming algorithms. We first define a natural hierarchy of space complexity classes of FPS, SubPS, SemiPS, SupPS and BrutePS, and then obtain tight classifications for several well-studied graph problems such as Longest Path, Feedback Vertex Set, Dominating Set, Girth, Treewidth, etc. into this hierarchy (see Figure 1 and Table 1). On the algorithmic side, our parameterized streaming algorithms use techniques from the FPT world such as bidimensionality, iterative compression and bounded-depth search trees. On the hardness side, we obtain lower bounds for the parameterized streaming complexity of various problems via novel reductions from problems in communication complexity. We also show a general (unconditional) lower bound for space complexity of parameterized streaming algorithms for a large class of problems inspired by the recently developed frameworks for showing (conditional) kernelization lower bounds.

Parameterized algorithms and streaming algorithms are approaches to cope with TIME and SPACE intractability respectively. It is our hope that this work on parameterized streaming algorithms leads to two-way flow of ideas between these two previously separated areas of theoretical computer science.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Parameterized Algorithms, Streaming Algorithms, Kernels

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.7

Related Version Full version of the paper is available at https://arxiv.org/abs/1911.09650

Funding *Rajesh Chitnis*: Work done while at University of Warwick, UK and supported by ERC grant 2014-CoG 647557.
Graham Cormode: Supported by ERC grant 2014-CoG 647557.

Acknowledgements We thank MohammadTaghi Hajiaghayi, Robert Krauthgamer and Morteza Monemizadeh for helpful discussions. Algorithm 1 was suggested to us by Arnold Filtser.

1 Introduction

Designing and implementing efficient algorithms is at the heart of computer science. Traditionally, efficiency of algorithms has been measured with respect to running time as a function of instance size. From this perspective, algorithms are said to be efficient if they can be solved in time which is bounded by some polynomial function of the input size. However, very many interesting problems are NP-complete, and so are grouped together as "not known to be efficient". This fails to discriminate within a large heterogenous group of problems, and in response the theory of *parameterized (time) algorithms* was developed in late 90's by Downey and Fellows [20]. Parameterized complexity attempts to delineate the complexity of problems by expressing the costs in terms of additional parameters. Formally, we say that a problem is *fixed-parameter tractable* (FPT) with respect to parameter k if the problem can be solved in time $f(k) \cdot n^{O(1)}$ where f is a computable function and n is the input size. For example, the problem of checking if a graph on n vertices has a vertex cover of size at most k can be solved in $2^k \cdot n^{O(1)}$ time. The study of various parameters helps to understand which parameters make the problem easier (FPT) and which ones cause it to be hard. The parameterized approach towards NP-complete problems has led to development of various algorithmic tools such as kernelization, iterative compression, color coding, and more [21, 14].

Kernelization: A key concept in fixed parameter tractability is that of kernelization which is an efficient preprocessing algorithm to produce a smaller, equivalent output called the "kernel". Formally, a kernelization algorithm for a parameterized problem Q is an algorithm which takes as an instance $\langle x, k\rangle$ and outputs in time polynomial in $(|x| + k)$ an equivalent[1] instance $\langle x', k'\rangle$ such that $\max\{|x'|, k'\} \leq f(k)$ for some computable function f. The output instance $\langle x', k'\rangle$ is called the kernel, while the function f determines the size of the kernel. Kernelizability is equivalent to fixed-parameter tractability, and designing compact kernels is an important question. In recent years, (conditional) lower bounds on kernels have emerged [3, 16, 17, 22, 27].

Streaming Algorithms: A very different paradigm for handling large problem instances arises in the form of streaming algorithms. The model is motivated by sources of data arising in communication networks and activity streams that are considered to be too big to store conveniently. This places a greater emphasis on the space complexity of algorithms. A streaming algorithm processes the input in one or a few read-only passes, with primary focus on the storage space needed. In this paper we consider streaming algorithms for graph problems over fixed vertex sets, where information about the edges arrives edge by edge [29]. We consider variants where edges can be both inserted and deleted, or only insertions are allowed. We primarily consider single pass streams, but also give some multi-pass results.

1.1 Parameterized Streaming Algorithms and Kernels

Given that parameterized algorithms have been extremely successful for the TIME resource, it seems natural to also use it attack the SPACE resource. In this paper, we advance the model of parameterized streaming algorithms, and start to flesh out a hierarchy of complexity classes. We focus our attention on graph problems, by analogy with FPT, where the majority of results have addressed graphs. From a space perspective, there is perhaps less headroom

[1] By equivalent we mean that $\langle x, k\rangle \in Q \Leftrightarrow \langle x', k'\rangle \in Q$

than when considering the time cost: for graphs on n vertices, the entire graph can be stored using $O(n^2)$ space[2]. Nevertheless, given that storing the full graph can be prohibitive, there are natural space complexity classes to consider. We formalize these below, but informally, the classes partition the dependence on n as: (i) (virtually) independent of n; (ii) sublinear in n; (iii) (quasi)linear in n; (iv) superlinear but subquadratic in n; and (v) quadratic in n.

Naively, several graph problems have strong lower bounds: for example, the problem of finding a minimum vertex cover on graphs of n vertices has a lower bound of $\Omega(n^2)$ bits. However, when we adopt the parameterized view, we seek streaming algorithms for (parameterized) graph problems whose space can be expressed as a function of *both* the number of vertices n and the parameter k. With this relaxation, we can separate out the problem space and start to populate our hierarchy. We next spell out our results, which derive from a variety of upper and lower bounds building on the streaming and FPT literature.

1.2 Our Results & Organization of the paper

For a graph problem with parameter k, there can be several possible choices for the space complexity needed to solve it in the streaming setting. In this paper, we first define some natural space complexity classes below:

1. $\widetilde{O}(f(k))$ *space:* Due to the connection to running time of `FPT` algorithms, we call the class of parameterized problems solvable using $\widetilde{O}(f(k))$ bits as `FPS` (fixed-parameterized streaming)[3].
2. *Sublinear space:* When the dependence on n is sublinear, we call the class of parameterized problems solvable using $\widetilde{O}(f(k) \cdot n^{1-\epsilon})$ bits as `SubPS` (sublinear parameterized streaming)
3. *Quasi-linear space:* Due to the connection to the semi-streaming model [26, 33], we call the set of problems solvable using $\widetilde{O}(f(k) \cdot n)$ bits as `SemiPS` (parameterized semi-streaming).
4. *Superlinear, subquadratic space:* When the dependence on n is superlinear (but subquadratic), we call the class of parameterized problems solvable using $\widetilde{O}(f(k) \cdot n^{1+\epsilon})$ bits (for some $1 > \epsilon > 0$) as `SupPS` (superlinear parameterized streaming).
5. *Quadratic space:* We call the set of graph problems solvable using $O(n^2)$ bits as `BrutePS` (brute-force parameterized streaming). Note that every graph problem is in `BrutePS` since we can just store the entire adjacency matrix using $O(n^2)$ bits (see Remark 2).

▶ Remark 1. Formally, we need t consider the following 7-tuple when we attempt to find its correct position in the aforementioned hierarchy of complexity classes:

[Problem, Parameter, Space,# of Passes, Type of Algorithm, Approx. Ratio, Type of Stream]

By type of algorithm, we mean that the algorithm could be deterministic or randomized. For the type of stream, the standard alternatives are (adversarial) insertion, (adversarial) insertion-deletion, random order, etc. Table 2 gives a list of results for the k-VC problem (as a case study) in various different settings. Unless stated otherwise, throughout this paper, we consider the space requirement for 1-pass exact deterministic algorithms for problems with the standard parameter (size of the solution)on insertion-only streams.

▶ Remark 2. There are various different models for streaming algorithms depending on how much computation is allowed on the stored data. In this paper, we consider the most general model by allowing *unbounded computation* at each edge update, and also at the end of the stream.

[2] Throughout the paper, by space we mean words/edges/vertices. Each word can be represented using $O(\log n)$ bits

[3] Throughout this paper, we use the $\tilde{O}$ notation to hide $\log^{O(1)} n$ factors

Our goal is to provide a tight classification of graph problems into the aforementioned complexity classes. We make progress towards this goal as follows: Section 2 shows how various techniques from the FPT world such as iterative compression, branching, bidimensionality, etc. can also be used to design parameterized streaming algorithms. First we investigate whether one can further improve upon the FPS algorithm of Chitnis et al. [9] for k-VC which uses $O(k^2 \cdot \log n)$ bits and one pass. We design two algorithms for k-VC which use $O(k \cdot \log n)$ bits[4]: an 2^k-pass algorithm using bounded-depth search trees (Section 2.1) and an $(k \cdot 2^{2k})$-pass algorithm using iterative compression (Section 2.2). Finally, Section 2.3 shows that any minor-bidimensional problem belongs to the class SemiPS.

Section 3 deals with lower bounds for parameterized streaming algorithms. First, in Section 3.1 we show that some parameterized problems are tight for the classes SemiPS and BrutePS. In particular, we show that k-Treewidth, k-Path and k-Feedback-Vertex-Set are tight for the class SemiPS, i.e., they belong to SemiPS but do not belong to the sub-class SubPS. Our SemiPS algorithms are based on problem-specific structural insights. Via reductions from the Perm problem [35], we rule out algorithms which use $\widetilde{O}(f(k) \cdot n^{1-\epsilon})$ bits (for any function f and any $\epsilon \in (0,1)$) for these problems by proving $\Omega(n \log n)$ bits lower bounds for constant values of k. Then we show that some parameterized problems such as k-Girth and k-Dominating-Set are tight for the class BrutePS, i.e, they belong to BrutePS but do not belong to the sub-class SupPS. Every graph problem belongs to BrutePS since we can store the entire adjacency matrix of the graph using $O(n^2)$ bits. Via reductions from the Index problem [30], we rule out algorithms which use $\widetilde{O}(f(k) \cdot n^{1+\epsilon})$ bits (for any function f and any $\epsilon \in (0,1)$) for these problems by proving $\Omega(n^2)$ bits lower bounds for constant values of k.

Section 3.2 shows a lower bound of $\Omega(n)$ bits for any algorithm that approximates (within a factor $\frac{\beta}{32}$) the size of min dominating set on graphs of arboricity $(\beta + 2)$, i.e., this problem has no $\widetilde{O}(f(\beta) \cdot n^{1-\epsilon})$ bits algorithm (since β is a constant), and hence does not belong to the class SubPS when parameterized by β. In Section 3.3 we obtain unconditional lower bounds on the space complexity of 1-pass parameterized streaming algorithms for a large class of graph problems inspired by some of the recent frameworks to show conditional lower bounds for kernels [3, 16, 17, 22, 27]. In the full version we also show that any parameterized streaming algorithm for the d-SAT problem (for any $d \geq 2$) must (essentially) follow the naive algorithm of storing all the clauses.

Figure 1 provides a pictorial representation of the complexity classes, and the known classification of several graph problems (from this paper and some previous work) into these classes. Table 1 summarizes our results, and clarifies the stream arrival model(s) under which they hold.

1.3 Prior work on Parametrized Streaming Algorithms

Prior work began by considering how to implement kernels in the streaming model. Formally, a streaming kernel [25] for a parameterized problem (I, k) is a streaming algorithm that receives the input I as a stream of elements, stores $f(k) \cdot \log^{O(1)} |I|$ bits and returns an equivalent instance[5]. This is especially important from the practical point of view since several real-world situations can be modeled by the streaming setting, and streaming kernels would help to efficiently preprocess these instances. Fafianie and Kratsch [25] showed that

[4] Which is essentially optimal since the algorithm also returns a VC of size k (if one exists)

[5] [25] required $f(k) = k^{O(1)}$, but we choose to relax this requirement

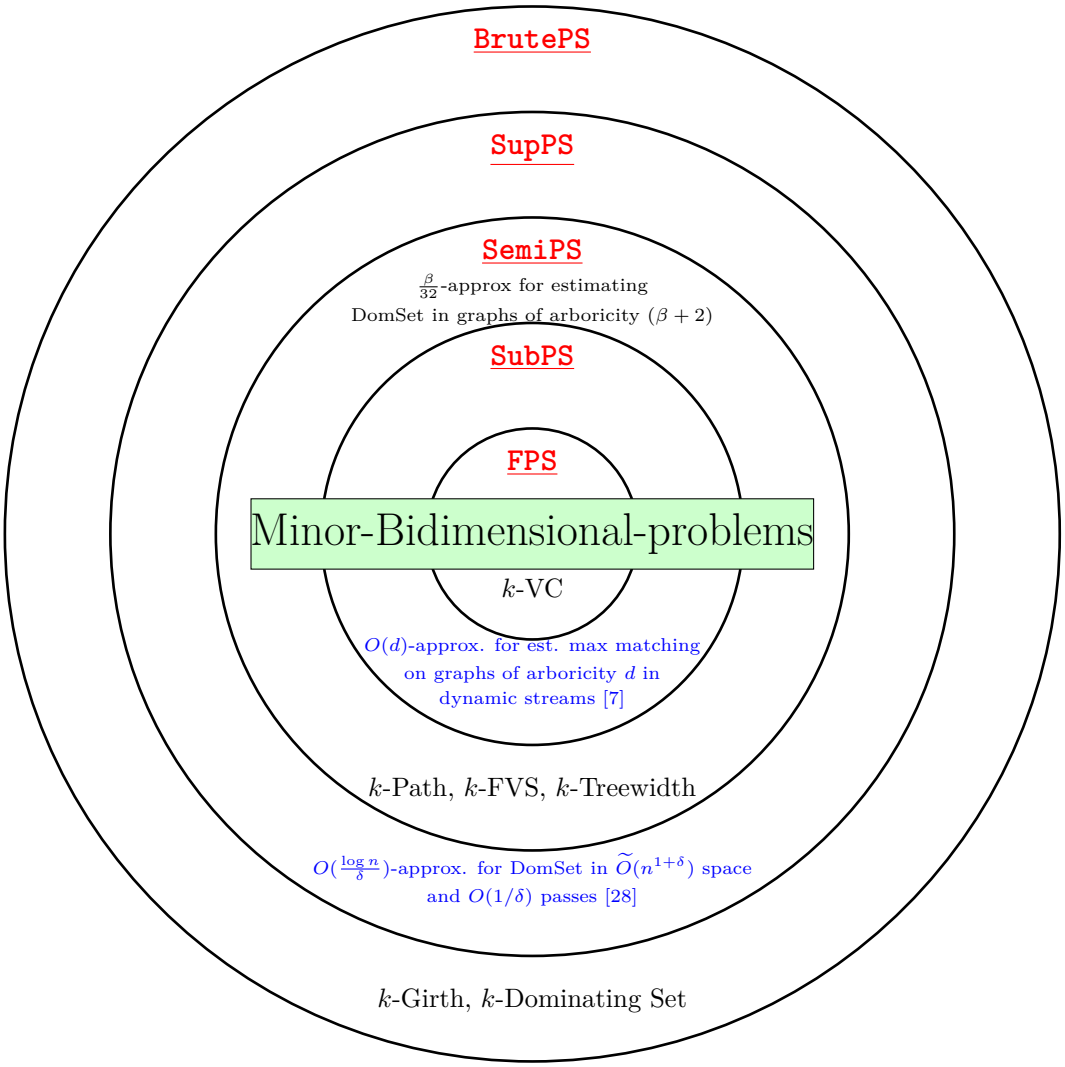


Figure 1 Pictorial representation of classification of some graph problems into complexity classes: our results are in black and previous work is referenced in blue. All results are for 1-pass deterministic algorithms on insertion-only streams unless otherwise specified. It was already known that k-VC $\in$ FPS [9, 7] using only 1-pass, but here we design an algorithm with optimal space storage at the expense of multiple passes.

the kernels for some problems like Hitting Set and Set Matching can be implemented in the streaming setting, but other problems such as Edge Dominating Set, Feedback Vertex Set, etc. do not admit (1-pass) streaming kernels.

Chitnis et al. [9] studied how to circumvent the worst case bound of $\Omega(n^2)$ bits for Vertex Cover by designing a streaming algorithm for the parameterized k-Vertex-Cover (k-VC)[6]. They showed that the k-VC problem can be solved in insertion-only streams using storage of $O(k^2)$ space. They also showed an almost matching lower bound of $\Omega(k^2)$ bits for any streaming algorithm for k-VC. A sequence of papers showed how to solve the k-VC problem in more general streaming models: Chitnis et al. [9, 8] gave an $\widetilde{O}(k^2)$ space algorithm under a particular promise, which was subsequently removed in [7].

[6] That is, determine whether there is a vertex cover of size at most k?

Table 1 Table summarizing our results (in the order in which they appear in the paper). All our algorithms are deterministic. All the lower bounds are unconditional, and hold even for randomized algorithms in insertion-only streams. Proofs of results labeled with [⋆] appear in the full version.

Problem	Number of Passes	Type of Stream	Space Upper Bound	Space Lower Bound
$g(r)$-minor-bidimensional problems [Sec. 2.3]	1	Ins-Del.	$\widetilde{O}((g^{-1}(k+1))^{10}n)$ words	—
k-VC [Sec. 2.2]	$2^{2k} \cdot k$	Ins-only	$O(k)$ words	$\Omega(k)$ words
k-VC [Sec. 2.1]	2^k	Ins-only	$O(k)$ words	$\Omega(k)$ words
k-FVS, k-Path k-Treewidth [Sec. 3.1]	1	Ins-only	$O(k \cdot n)$ words	No $f(k) \cdot n^{1-\epsilon} \log^{O(1)} n$ bits algorithm
k-FVS, k-Path k-Treewidth [Sec. 3.1]	1	Ins-Del.	$\widetilde{O}(k \cdot n)$ words	No $f(k) \cdot n^{1-\epsilon} \log^{O(1)} n$ bits algorithm
k-Girth, k-DomSet, [Sec. 3.1]	1	Ins-Del.	$O(n^2)$ bits	No $f(k) \cdot n^{2-\epsilon} \log^{O(1)} n$ bits algorithm
$\frac{\beta}{32}$-approximation for size of min DomSet on graphs of arboricity β [Sec. 3.2]	1	Ins-only	$\widetilde{O}(n\beta)$ bits	No $f(\beta) \cdot n^{1-\epsilon}$ bits algorithm
AND-compatible problems and OR-compatible problems [Sec. 3.3]	1	Ins-only	$O(n^2)$ bits	No $\widetilde{O}(f(k) \cdot n^{1-\epsilon})$ bits algorithm
d-SAT with N variables [⋆]	1	Clause Arrival	$\widetilde{O}(d \cdot N^d)$ bits	$\Omega((N/d)^d)$ bits

Table 2 Table summarizing some of the results for the k-VC problem in the different settings outlined in Remark 1. Proofs of results labeled with [⋆] appear in the full version.

Problem	# of Passes	Type of Stream	Type of Algorithm	Approx. Ratio	Space Bound
k-VC	1	Ins-only	Det.	1	$O(k^2 \log n)$ bits [9]
k-VC	1	Ins-only	Rand.	1	$\Omega(k^2)$ bits [9]
k-VC	1	Ins-Del.	Rand.	1	$O(k^2 \log^{O(1)} n)$ bits [7]
k-VC	2^k	Ins-only	Det.	1	$O(k \log n)$ bits [Algorithm 1]
k-VC	$k \cdot 2^k$	Ins-only.	Det.	1	$O(k \log n)$ bits [⋆]
Estim. k-VC	$\Omega(k/\log n)$	Ins-only.	Rand.	1	$O(k \log n)$ bits [1, Theorem 16]
Estim. k-VC on Trees	1	Ins-only.	Det. Rand.	$(3/2-\epsilon)$	$\Omega(n)$ bits [24, Theorem 6.1] $\Omega(\sqrt{n})$ bits [24, Theorem 6.1]

Recently, there have been several papers considering the problem of estimating the size of a maximum matching using $o(n)$ space in graphs of bounded arboricity. If the space is required to be sublinear in n, then versions of the problem that involve estimating the size of a maximum matching (rather than demonstrating such a matching) become the focus. Since the work of Esfandiari et al. [24], there have been several sublinear space algorithms [31, 32, 12, 7] which obtain $O(\alpha)$-approximate estimations of the size of maximum matching in graphs of arboricity α. The current best bounds [4, 12] for insertion-only streams is $O(\log^{O(1)} n)$ space and for insertion-deletion streams is $\widetilde{O}(\alpha \cdot n^{4/5})$. All of these results can be viewed as parameterized streaming algorithms (`FPS` or `SubPS`) for approximately estimating the size of maximum matching in graphs parameterized by the arboricity.

2 Parameterized Streaming Algorithms Inspired by FPT techniques

In this section we design parameterized streaming algorithms using three techniques from the world of parameterized algorithms, viz. branching, iterative compression and bidimensionality.

2.1 Multipass FPS algorithm for k-VC using Branching

The streaming algorithm from Section 2.2 already uses optimal storage of $O(k \log n)$ bits but requires $O(2^{2k} \cdot k)$ passes. In this section, we show how to reduce the number of passes to 2^k (while still maintaining the same storage) using the technique of bounded-depth search trees (also known as branching). The method of bounded-depth search trees gives a folklore FPT algorithm for k-VC which runs in $2^{O(k)} \cdot n^{O(1)}$ time. The idea is simple: any vertex cover must contain at least one end-point of each edge. We now build a search tree as follows: choose an arbitrary edge, say $e = u - v$ in the graph. Start with the graph G at the root node of the search tree. Branch into two options, viz. choosing either u or v into the vertex cover[7]. The resulting graphs at the two children of the root node are $G - u$ and $G - v$. Continue the branching process. Note that at each step, we branch into two options and we only need to build the search tree to height k for the k-VC problem. Hence, the binary search tree has $2^{O(k)}$ leaf nodes. If the resulting graph at any leaf node is empty (i.e., has no edges) then G has a vertex cover of size $\leq k$ which can be obtained by following the path from the root node to the leaf node in the search tree. Conversely, if the resulting graphs at none of the leaf nodes of the search tree are empty then G does not have a vertex cover of size $\leq k$: this is because at each step we branched on all the (two) possibilities at each node of the search tree.

Simulating branching-based FPT algorithm using multiple passes: We now simulate the branching-based FPT algorithm described in the previous section using 2^k passes and $O(k \log n)$ bits of storage in the streaming model.

▶ **Definition 3.** *Let $V(G) = \{v_1, v_2, \ldots, v_n\}$. Fix some ordering ϕ on $V(G)$ as follows: $v_1 < v_2 < v_3 < \ldots < v_n$. Let $\texttt{Dict}_k$ be the dictionary ordering on the 2^k binary strings of $\{0,1\}^k$. Given a string $X \subseteq \{0,1\}^k$, let $\texttt{Dict}_k(\texttt{Next}(X))$ denote the string that comes immediately after X in the ordering $\texttt{Dict}_k$. We set $\texttt{Dict}_k(\texttt{Next}(1^k)) = \spadesuit$*

We formally describe our multipass algorithm in Algorithm 1. This algorithm crucially uses the fact that in each pass we see the edges of the stream in the *same* order.

▶ **Theorem 4.** [⋆] [8] *Algorithm 1 correctly solves the k-VC problem using 2^k passes and $O(k \log n)$ bits of storage.*

Note that the total storage of Algorithm 1 is $O(k \log n)$ bits which is essentially optimal since the algorithm also outputs a vertex cover of size at most k (if one exists).

The next natural question is whether one need exponential (in k) number of passes when we want to solve the k-VC problem using only $O(k \log n)$ bits. A lower bound of $(k/\log n)$ passes follows for such algorithms from the following result of Abboud et al.

▶ **Theorem 5.** (rewording of [1, Thm 16]) *Any algorithm for the k-VC problem which uses S bits of space and R passes must satisfy $RS \geq n^2$*

[7] Note that if we choose u in the first branch then that does not imply that we cannot or will not choose v later on in the search tree

[8] Proofs of results labeled with [⋆] appear in the full version.

Algorithm 1 2^k-pass Streaming Algorithm for k-VC using $O(k \log n)$ bits via Branching.

Input: An undirected graph $G = (V, E)$ and an integer k.
Output: A vertex cover S of G of size $\leq k$ (if one exists), and NO otherwise
Storage: i, j, S, X

Let $X = 0^k$, and suppose the edges of the graph are seen in the order $e_1, e_2, \ldots, e_m$
while $X \neq$ ♠ **do**
  $S = \emptyset, i = 1, j = 1$
  while $i \neq k + 1$ **do**
    Let $e_j = u - v$ such that $u < v$ under the ordering ϕ
    if Both $u \notin S$ and $v \notin S$ **then**
      if $X[i] = 0$ **then** $S \leftarrow S \cup \{u\}$
      else $S \leftarrow S \cup \{v\}$
      $i \leftarrow i + 1$
    $j \leftarrow j + 1$
  if $j = m + 1$ **then** Return S and abort
  else $X \leftarrow \texttt{Dict}_k(\texttt{Next}(X))$
if $X =$ ♠ **then** Return NO

2.2 Multipass FPS algorithm for k-VC using Iterative Compression

The technique of *iterative compression* was introduced by Reed et al. [34] to design the first FPT algorithm for the k-OCT problem[9]. Since then, iterative compression has been an important tool in the design of faster parameterized algorithms [6, 10, 5] and kernels [15]. In the full version, using the technique of iterative compression, we design an algorithm for k-VC which uses $O(k \log n)$ bits but requires $O(k \cdot 2^{2k})$ passes. Although this algorithm is strictly worse (same storage, but higher number of passes) compared to Algorithm 1, we mention it here to illustrate that the technique of iterative compression can be used in the streaming setting.

As in the FPT setting, a natural problem to attack using iterative compression in the streaming setting would be the k-OCT problem. It is known that 0-OCT, i..e, checking if a given graph is bipartite, in the 1-pass model has an upper bound of $O(n \log n)$ bits [26] and a lower bound of $\Omega(n \log n)$ bits [35]. For $k \geq 1$, can we design a $g(k)$-pass algorithm for k-OCT which uses $\widetilde{O}(f(k) \cdot n)$ bits for some functions f and g, maybe using iterative compression? To the best of our knowledge, such an algorithm is not known even for 1-OCT.

2.3 Minor-Bidimensional problems belong to SemiPS

The theory of bidimensionality [18, 19] provides a general technique for designing (subexponential) FPT for NP-hard graph problems on various graph classes. In this section, we briefly sketch how we can use this technique to show that a large class of problems belong to the class SemiPS. All the details (including graph-theoretic definitions such as minors, treewidth, etc.) of this section are deferred to the full version.

▶ **Definition 6** (minor-bidimensional). *A graph problem* Π *is* $g(r)$*-minor-bidimensional if*

- *The value of* Π *on the* $r \times r$ *grid is* $\geq g(r)$
- Π *is closed under taking minors, i.e., the value of* Π *does not increase under the operations of vertex deletions, edge deletions, edge contractions.*

[9] Is there a set of size at most k whose deletion makes the graph odd cycle free, i.e. bipartite

Hence, we obtain the following "win-win" approach for designing FPT algorithms for bidimensional problems:

- Either the graph has small treewidth and we can then use dynamic programming algorithms for bounded treewidth graphs; or
- The treewidth is large[10] which implies that the graph contains a large grid as a minor. This implies that the solution size is large, since the parameter is minor-bidimensional.

Several natural graph parameters are known to be minor-bidimensional. For example, treewidth is $\Omega(r)$-minor-dimensional and Feedback Vertex Set, Vertex Cover, Minimum Maximal Matching, Long Path, etc are $\Omega(r^2)$-minor-bidimensional. To design parameterized streaming algorithms, we will replace the dynamic programming step for bounded treewidth graphs by simply storing all the edges of such graphs. The main theorem of this section is that minor-bidimensional problems belong to the class `SemiPS`.

▶ **Theorem 7.** [⋆] (minor-bidimensional problems ∈ `SemiPS`) *Let* Π *be a* $g(r)$*-minor-dimensional problem. Then the* k-Π *problem on graphs with* n *vertices can be solved using*

- $O((g^{-1}(k+1))^{10} \cdot n)$ *space in insertion-only streams*
- $\widetilde{O}((g^{-1}(k+1))^{10} \cdot n)$ *space in insertion-deletion streams*

Theorem 7 implies the following results for specific graph problems[11]:

- Since Treewidth is $\Omega(r)$-minor-bidimensional, it follows that k-Treewidth has an $O(k^{10} \cdot n)$ space algorithm in insertion-only streams and $\widetilde{O}(k^{10} \cdot n)$ space algorithm in insertion-deletion streams.
- Since problems such as Long Path, Vertex Cover, Feedback Vertex Set, Minimum Maximal Matching, etc. are $\Omega(r^2)$-minor-bidimensional, it follows that their parameterized versions have $O(k^5 \cdot n)$ space algorithm in insertion-only streams and $\widetilde{O}(k^5 \cdot n)$ space algorithm in insertion-deletion streams.

In Section 3.1, we design algorithms for some of the aforementioned problems with smaller storage. In particular, we design problem-specific structural lemmas to reduce the dependency of k on the storage from $k^{O(1)}$ to k.

▶ Remark 8. It is tempting to conjecture a lower bound complementing Theorem 7: for example, can we show that the bounds for minor-bidimensional problems are tight for `SemiPS`, i.e., they do not belong to `SubPS` or even `FPS`? Unfortunately, we can rule out such a converse to Theorem 7 via the two examples of Vertex Cover (VC) and Feedback Vertex Set (FVS) which are both $\Omega(r^2)$-minor-bidimensional. Chitnis et al. [9] showed that k-VC can be solved in $O(k^2)$ space and hence belongs to the class `FPS`. However, we show in the full version that k-FVS cannot belong to `SubPS` since it has a $\Omega(n \log n)$ bits lower bound for $k = 0$.

3 Lower Bounds for Parameterized Streaming Algorithms

3.1 Tight Problems for the classes `SemiPS` and `BrutePS`

In this section we show that certain problems are tight for the classes `SemiPS` and `BrutePS`. All of the results hold for 1-pass in the insertion-only model. Our algorithms are deterministic, while the lower bounds also hold for randomized algorithms.

[10] Chuzhoy and Tan [11] showed that treewidth $= O(r^9 \cdot \log^{O(1)} r) \Rightarrow$ there is a $r \times r$ grid minor
[11] We omit the simple proofs of why these problems satisfy the conditions of Definition 6

Tight Problems for the class SemiPS: We now show that some parameterized problems are tight for the class SemiPS, i.e.,

- They belong to SemiPS, i.e., can be solved using $\widetilde{O}(g(k) \cdot n)$ bits for some function g.
- They do not belong to SubPS, i.e., there is no algorithm which uses $\widetilde{O}(f(k) \cdot n^{1-\epsilon})$ bits for any function f and any constant $1 > \epsilon > 0$. We do this by showing $\Omega(n \cdot \log n)$ bits lower bounds for these problems for constant values of k.

For each of the problems considered in this section, a lower bound of $\Omega(n)$ bits (for constant values of k) was shown by Chitnis et al. [7]. To obtain the improved lower bound of $\Omega(n \cdot \log n)$ bits for constant k, we will reduce from the Perm problem defined by Sun and Woodruff [35].

Perm
Input: Alice has a permutation $\delta : [N] \to [N]$ which is represented as a bit string B_δ of length $N \log N$ by concatenating the images of $1, 2, \ldots, N$ under δ. Bob has an index $I \in [N \log N]$.
Goal: Bob wants to find the I-th bit of B_δ

Sun and Woodruff [35] showed that the one-way (randomized) communication complexity of Perm is $\Omega(N \cdot \log N)$. Using the Perm problem, we show $\Omega(n \cdot \log n)$ bit lower bounds for constant values of k for various problem such as k-Path, k-Treewidth, k-Feedback-Vertex-Set, etc. We also show a matching upper bound for these problems: for each k, these problems can be solved using $O(kn \cdot \log n)$ words in insertion-only streams and $\widetilde{O}(kn \cdot \log n)$ words in insertion-deletion streams. The proofs of these results are deferred to the full version. To the best of our knowledge, the only problems known previously to be tight for SemiPS were k-vertex-connectivity and k-edge-connectivity [13, 35, 23].

Tight Problems for the class BrutePS: We now show that some parameterized problems are tight for the class BrutePS, i.e.,

- They belong to BrutePS, i.e., can be solved using $O(n^2)$ bits. Indeed any graph problem can be solved by storing the entire adjacency matrix which requires $O(n^2)$ bits.
- They do not belong to SubPS, i.e., there is no algorithm which uses $\widetilde{O}(f(k) \cdot n^{1+\epsilon})$ bits for any function f and any $\epsilon \in (0, 1)$. We do this by showing $\Omega(n^2)$ bits lower bounds for these problems for constant values of k via reductions from the Index problem.

Index
Input: Alice has a string $B = b_1 b_2 \ldots b_N \in \{0, 1\}^N$. Bob has an index $I \in [N]$
Goal: Bob wants to find the value b_I

There is a $\Omega(N)$ lower bound on the (randomized) one-way communication complexity of Index [30]. Via reduction from the Index problem, we are able to show $\Omega(n^2)$ bits for constant values of k for several problems such as k-Dominating-Set and k-Girth. The proofs of these reductions are deferred to the full version.

▶ Remark 9. We usually only design FPT algorithms for NP-hard problems. However, parameterized streaming algorithms make sense for all graph problems since we are only comparing ourselves against the naive choice of storing all the $O(n^2)$ bits of adjacency matrix. Hence, here we consider the k-Girth problem as an example of a polynomial time solvable problem.

Finally, in the full version, we also show that for any $d \geq 2$, any streaming algorithm for d-SAT (in the clause arrival model) must essentially store all the clauses (and hence fits into the "brute-force" streaming setting). This is the only non-graph-theoretic result in this paper, and may be viewed as a "streaming analogue" of the Exponential Time Hypothesis.

3.2 Lower bound for approximating size of minimum Dominating Set on graphs of bounded arboricity

▶ **Theorem 10.** *Let $\beta \geq 1$ be any constant. Then any algorithm which $\frac{\beta}{32}$-approximates the size of a min dominating set on graphs of arboricity $\beta + 2$ requires $\Omega(n)$ space.*

Note that Theorem 10 shows that the naive algorithm which stores all the $O(n\beta)$ edges of an β-arboriticy graph is essentially optimal. Our lower bound holds even for randomized algorithms (required to have success probability $\geq 3/4$) and also under the vertex arrival model, i.e., we see at once all edges incident on a vertex. We (very) closely follow the outline from [2, Theorem 4] who used this approach for showing that any α-approximation for estimating size of a minimum dominating set in general graphs requires $\tilde{\Omega}(\frac{n^2}{\alpha^2})$ space. Because we are restricted to bounded arboriticy graphs, we cannot just sue their reduction as a black-box but need to adapt it carefully for our purposes.

Let $V(G) = [n+1]$, and $\mathcal{F}_\beta$ be the collection of all subsets of $[n]$ with cardinality β. Consider the following distribution $\mathcal{D}_{\mathsf{est}}$ for $\mathsf{DomSet}_{\mathsf{est}}$.

Distribution $\mathcal{D}_{\mathsf{est}}$: A hard input distribution for $\mathsf{DomSet}_{\mathsf{est}}$.

- **Alice.** The input of Alice is a collection of n sets $\mathcal{S}' = \{S'_1, S'_2, \ldots, S'_n\}$ where for each $i \in [n]$ we have that $S'_i = \{i\} \cup S_i$ with S_i being a set chosen independently and uniformly at random from $\mathcal{F}_\beta$.
- **Bob.** Pick $\theta \in \{0, 1\}$ and $i^* \in [n]$ independently and uniformly at random; the input of Bob is a single set T defined as follows.
  - If $\theta = 0$, then $\overline{T} = [n] \setminus T$ is a set of size $\beta/8$ chosen uniformly at random from S_{i^*}.
  - If $\theta = 1$, then $\overline{T} = [n] \setminus T$ is a set of size $\beta/8$ chosen uniformly at random from $[n] \setminus S_{i^*}$.

Recall that $\mathrm{OPT}(\mathcal{S}', T)$ denotes the size of the minimum *dominating set* of the graph G whose edge set is given by $N[i] = \{i\} \cup S_i$ for each $i \in [n]$ and $N[n+1] = \{n+1\} \cup T$. It is easy to see that G has arboricity $\leq (\beta+2)$ since it has $(n+1)$ vertices and $\leq (\beta+1)n + (1+n-\frac{\beta}{8})$ edges. We first establish the following lemma regarding the parameter θ and $\mathrm{OPT}(\mathcal{S}', T)$ in the distribution $\mathcal{D}_{\mathsf{est}}$.

▶ **Lemma 11.** [$\star$] *Let $\alpha = \frac{\beta}{32}$. Then, for $(\mathcal{S}', T) \sim \mathcal{D}_{est}$ we have*

1. $\Pr\left(OPT(\mathcal{S}', T) = 2 \mid \theta = 0\right) = 1$.
2. $\Pr\left(OPT(\mathcal{S}', T) > 2\alpha \mid \theta = 1\right) = 1 - o(1)$.

The proof of Lemma 11 is deferred to the full version. The first observation is that the distribution $\mathcal{D}_{\mathsf{est}}$ is not a product distribution due to the correlation between the input given to Alice and Bob. However, we can express the distribution $\mathcal{D}_{\mathsf{est}}$ as a convex combination of a relatively small set of product distributions. The proof of Theorem 10 then follows by showing a lower bound on this set of product distributions. This proof is a bit technical, and we defer it to the full version.

3.3 Streaming Lower Bounds Inspired by Kernelization Lower Bounds

Streaming algorithms and kernelization are two (somehwhat related) compression models. In kernelization, we have access to the whole input but our computation power is limited to polynomial time whereas in streaming algorithms we don't have access to the whole graph (have to pay for whatever we store) but have unbounded computation power on whatever part of the input we have stored.

A folklore result states that a (decidable) problem is FPT if and only if it has a kernel. Once the fixed-parameter tractability for a problem is established, the next natural goals are to reduce the running time of the FPT algorithm and reduce the size of the kernel. In the last decade, several frameworks have been developed to show (conditional) lower bounds on the size of kernels [3, 16, 17, 22, 27]. Inspired by these frameworks, we define a class of problems, which we call as AND-compatible and OR-compatible, and show (unconditionally) that none of these problems belong to the class `SubPS`.

▶ **Definition 12.** *We say that a graph problem* Π *is* AND-compatible *if there exists a constant* k *such that*

- *for every* $n \in \mathbb{N}$ *there exists a graph* G_{YES} *of size* n *such* $\Pi(G_{\text{YES}}, k)$ *is a YES instance*
- *for every* $n \in \mathbb{N}$ *there exists a graph* G_{NO} *of size* n *such* $\Pi(G_{\text{NO}}, k)$ *is a NO instance*
- *for every* $t \in \mathbb{N}$ *we have that* $\Pi\Big(\uplus_{i=1}^{t} G_i, k\Big) = \wedge_{i=1}^{t}\Pi(G_i, k)$ *where* $G = \uplus_{i=1}^{t} G_i$ *denotes the union of the vertex-disjoint graphs* $G_1, G_2, \ldots, G_t$

Examples of *AND-compatible* graph problems are k-Treewidth, k-Girth, k-Pathwidth, k-Coloring, etc.

▶ **Definition 13.** *We say that a graph problem* Π *is* OR-compatible *if there exists a constant* k *such that*

- *for every* $n \in \mathbb{N}$ *there exists a graph* G_{YES} *of size* n *such* $\Pi(G_{YES}, k)$ *is a YES instance*
- *for every* $n \in \mathbb{N}$ *there exists a graph* G_{NO} *of size* n *such* $\Pi(G_{NO}, k)$ *is a NO instance*
- *for every* $t \in \mathbb{N}$ *we have that* $\Pi(\uplus_{i=1}^{t} G_i, k) = \vee_{i=1}^{t}\Pi(G_i, k)$ *where* $G = \uplus_{i=1}^{t} G_i$ *denotes the union of the vertex-disjoint graphs* $G_1, G_2, \ldots, G_t$

A general example of an *OR-compatible* graph problem is the subgraph isomorphism problem parameterized by size of smaller graph: given a graph G of size n and a smaller graph H of size k, does G have a subgraph isomorphic to H? Special cases of this problem are k-Path, k-Clique, k-Cycle, etc.

▶ **Theorem 14.** *If* Π *is an* AND-compatible *or an* OR-compatible *graph problem then* $\Pi \notin$ `SubPS`

Proof. Let Π be an *AND-compatible* graph problem, and $G = \uplus_{i=1}^{t} G_i$ for some $t \in \mathbb{N}$. We claim that any streaming algorithm ALG for Π must use t bits. Intuitively, we need at least one bit to check that each of the instances (G_i, k) is a YES instance of Π (for all $1 \leq i \leq t$). Consider a set of t graphs $\mathcal{G} = \{G_1, G_2, \ldots, G_t\}$: note that we don't fix any of these graphs yet. For every subset $X \subseteq [t]$ we define the instance (G_X, k) of Π where $G_X = \uplus_{j \in J} G_j$. Suppose that ALG uses less than t bits. Then by pigeonhole principle, there are two subsets I, I' of $[t]$ such that ALG has the same answer on (G_I, k) and $(G_{I'}, k)$. Since $I \neq I'$ (without loss of generality) there exists i^* such that $i^* \in I \setminus I'$. This is where we now fix each of the graphs in $\mathcal{G}$ to arrive at a contradiction: consider the input where $G_i = G_{\text{YES}}$ for all $(I \cup I') \setminus i^*$ and $G_{i^*} = G_{\text{NO}}$. Then, it follows that (G_I, k) is a NO instance but $(G_{I'}, k)$ is a YES instance.

Suppose that $\Pi \in$ `SubPS`, i.e., there is an algorithm for Π which uses $f(k) \cdot N^{1-\epsilon} \cdot \log^{O(1)} N$ bits (for some $1 > \epsilon > 0$) on a graph G of size N to decide whether (G, k) is a YES or NO instance. Let $G = \uplus_{i=1}^{t} G_i$ where $|G_i| = n$ for each $i \in [t]$. Then $|G| = N = nt$. By the previous paragraph, we have that

$$f(k) \cdot (nt)^{1-\epsilon} \cdot \log^{O(1)}(nt) \geq t \Rightarrow f(k) \cdot n^{1-\epsilon} \cdot \log^{O(1)}(nt) \geq t^{\epsilon}$$

Choosing $t = n^{\frac{2-\epsilon}{\epsilon}}$ we have that $f(k) \cdot \log^{O(1)} n^{1+(\frac{2-\epsilon}{\epsilon})} \geq n$, which is a contradiction for large enough n (since k and ϵ are constants).

We now prove the lower bound for AND-compatible problems. Recall that De Morgan's law states that $\neg(\vee_i P_i) = \wedge_i(\neg P_i)$. Hence, if Π is an *OR-compatible* graph problem then the complement[12] problem $\overline{\Pi}$ is an *AND-compatible* graph problem, and hence the lower bound follows from the previous paragraph. ◀

▶ Remark 15. Note that throughout this paper we have considered the model where we allow unbounded computation at each edge update, and also at the end of the stream. However, if we consider a **restricted** model of allowing only polynomial (in input size n) computation at each edge update and also at end of the stream, then it is easy to see that existing (conditional) lower bounds from the parameterized algorithms and kernelization setting translate easily to this restricted model. For example, the following two lower bounds for parameterized streaming algorithms follow immediately in the restricted (polytime computation) model:

- Let X be a graph problem that is $W[i]$-hard parameterized by k (for some $i \geq 1$). Then (in the polytime computation model) $X \notin \texttt{FPS}$ unless $\texttt{FPT} = W[i]$.
- Let X be a graph problem that is known to not have a polynomial kernel unless $\texttt{NP} \subseteq \text{co}\texttt{NP}/\text{poly}$. Then (in the polytime computation model) X does not have a parameterized streaming algorithm which uses $k^{O(1)} \cdot \log^{O(1)} n$ bits, unless $\texttt{NP} \subseteq \text{co}\texttt{NP}/\text{poly}$.

4 Conclusions & Open Problems

In this paper, we initiate a systematic study of graph problems from the paradigm of parameterized streaming algorithms. We define space complexity classes of FPS, SubPS, SemiPS, SupPS and BrutePS, and then obtain tight classifications for several well-studied graph problems such as Longest Path, Feedback Vertex Set, Girth, Treewidth, etc. into these classes. Our parameterized streaming algorithms use techniques of bidimensionality, iterative compression and branching from the FPT world. In addition to showing lower bounds for some parameterized streaming problems via communication complexity, we also show how (conditional) lower bounds for kernels and W-hard problems translate to lower bounds for parameterized streaming algorithms.

Our work leaves open several concrete questions. We list some of them below:

- The streaming algorithm (Algorithm 1) for k-VC (on insertion-only streams) from Section 2.1 has an optimal storage of $O(k \log n)$ bits but requires 2^k passes. Can we reduce the number of passes to poly(k), or instead show that we need passes which are superpolynomial in k if we restrict space usage to $O(k \log n)$ bits? The only known lower bound for such algorithms is $(k/\log n)$ passes (see Theorem 5).
- For $k \geq 1$ can we design algorithms which use $f(k) \cdot n \cdot \log^{O(1)} n$ bits and $g(k)$ passes for the k-OCT problem (for some functions f, g)? The technique of iterative compression seems like a natural tool to use here.

[12] By complement, we mean that $\overline{\Pi}(G, k)$ is YES if and only if $\Pi(G, k)$ is NO

References

1 Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Ami Paz. Smaller Cuts, Higher Lower Bounds. *CoRR*, abs/1901.01630, 2019. arXiv:1901.01630.

2 Sepehr Assadi, Sanjeev Khanna, and Yang Li. Tight bounds for single-pass streaming complexity of the set cover problem. In *STOC*, pages 698–711, 2016. doi:10.1145/2897518.2897576.

3 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On Problems Without Polynomial Kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

4 Marc Bury, Elena Grigorescu, Andrew McGregor, Morteza Monemizadeh, Chris Schwiegelshohn, Sofya Vorotnikova, and Samson Zhou. Structural Results on Matching Estimation with Applications to Streaming. *Algorithmica*, 81(1):367–392, 2019. doi:10.1007/s00453-018-0449-y.

5 Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. Syst. Sci.*, 74(7):1188–1198, 2008. doi:10.1016/j.jcss.2008.05.002.

6 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008. doi:10.1145/1411509.1411511.

7 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via Sampling with Applications to Finding Matchings and Related Problems in Dynamic Graph Streams. In *SODA*, pages 1326–1344, 2016. doi:10.1137/1.9781611974331.ch92.

8 Rajesh Hemant Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. Brief Announcement: New Streaming Algorithms for Parameterized Maximal Matching & Beyond. In *SPAA*, pages 56–58, 2015. doi:10.1145/2755573.2755618.

9 Rajesh Hemant Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized Streaming: Maximal Matching and Vertex Cover. In *SODA*, pages 1234–1251, 2015. doi:10.1137/1.9781611973730.82.

10 Rajesh Hemant Chitnis, Marek Cygan, Mohammad Taghi Hajiaghayi, and Dániel Marx. Directed Subset Feedback Vertex Set Is Fixed-Parameter Tractable. *ACM Trans. Algorithms*, 11(4):28:1–28:28, 2015. doi:10.1145/2700209.

11 Julia Chuzhoy and Zihan Tan. Towards Tight(er) Bounds for the Excluded Grid Theorem. In *SODA*, pages 1445–1464, 2019. doi:10.1137/1.9781611975482.88.

12 Graham Cormode, Hossein Jowhari, Morteza Monemizadeh, and S. Muthukrishnan. The Sparse Awakens: Streaming Algorithms for Matching Size Estimation in Sparse Graphs. In *ESA*, pages 29:1–29:15, 2017. doi:10.4230/LIPIcs.ESA.2017.29.

13 Michael S. Crouch, Andrew McGregor, and Daniel Stubbs. Dynamic Graphs in the Sliding-Window Model. In *ESA*, pages 337–348, 2013. doi:10.1007/978-3-642-40450-4_29.

14 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.

15 Frank K. H. A. Dehne, Michael R. Fellows, Frances A. Rosamond, and Peter Shaw. Greedy Localization, Iterative Compression, Modeled Crown Reductions: New FPT Techniques, an Improved Algorithm for Set Splitting, and a Novel $2k$ Kernelization for Vertex Cover. In *IWPEC*, pages 271–280, 2004. doi:10.1007/978-3-540-28639-4_24.

16 Holger Dell. AND-Compression of NP-Complete Problems: Streamlined Proof and Minor Observations. *Algorithmica*, 75(2):403–423, 2016. doi:10.1007/s00453-015-0110-y.

17 Holger Dell and Dieter van Melkebeek. Satisfiability Allows no Nontrivial Sparsification Unless the Polynomial-Time Hierarchy Collapses. In *STOC*, pages 251–260, 2010. doi:10.1145/1806689.1806725.

18 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005. doi:10.1145/1101821.1101823.

19 Erik D. Demaine and MohammadTaghi Hajiaghayi. The Bidimensionality Theory and Its Algorithmic Applications. *Comput. J.*, 51(3):292–302, 2008. doi:10.1093/comjnl/bxm033.

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

21 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.

22 Andrew Drucker. New Limits to Classical and Quantum Instance Compression. In *FOCS*, pages 609–618, 2012. doi:10.1109/FOCS.2012.71.

23 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997. doi:10.1145/265910.265914.

24 Hossein Esfandiari, Mohammad Taghi Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming Algorithms for Estimating the Matching Size in Planar Graphs and Beyond. In *SODA*, pages 1217–1233, 2015. doi:10.1137/1.9781611973730.81.

25 Stefan Fafianie and Stefan Kratsch. Streaming Kernelization. In *MFCS*, pages 275–286, 2014. doi:10.1007/978-3-662-44465-8_24.

26 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005. doi:10.1016/j.tcs.2005.09.013.

27 Lance Fortnow and Rahul Santhanam. Infeasibility of Instance Compression and Succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011. doi:10.1016/j.jcss.2010.06.007.

28 Sariel Har-Peled, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. Towards Tight Bounds for the Streaming Set Cover Problem. In *PODS*, pages 371–383, 2016. doi:10.1145/2902251.2902287.

29 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, 50:107–118, 1998.

30 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

31 Andrew McGregor and Sofya Vorotnikova. Planar Matching in Streams Revisited. In *APPROX/RANDOM*, pages 17:1–17:12, 2016. doi:10.4230/LIPIcs.APPROX-RANDOM.2016.17.

32 Andrew McGregor and Sofya Vorotnikova. A Simple, Space-Efficient, Streaming Algorithm for Matchings in Low Arboricity Graphs. In *SOSA*, pages 14:1–14:4, 2018. doi:10.4230/OASIcs.SOSA.2018.14.

33 S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005. doi:10.1561/0400000002.

34 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.

35 Xiaoming Sun and David P. Woodruff. Tight Bounds for Graph Problems in Insertion Streams. In *APPROX-RANDOM*, pages 435–448, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.435.

FPT Inapproximability of Directed Cut and Connectivity Problems

Rajesh Chitnis
School of Computer Science, University of Birmingham, UK
rajeshchitnis@gmail.com

Andreas Emil Feldmann
Charles University, Czechia
feldmann.a.e@gmail.com

Abstract

Cut problems and connectivity problems on digraphs are two well-studied classes of problems from the viewpoint of parameterized complexity. After a series of papers over the last decade, we now have (almost) tight bounds for the running time of several standard variants of these problems parameterized by two parameters: the number k of terminals and the size p of the solution. When there is evidence of FPT intractability, then the next natural alternative is to consider FPT approximations. In this paper, we show two types of results for directed cut and connectivity problems, building on existing results from the literature: first is to circumvent the hardness results for these problems by designing FPT approximation algorithms, or alternatively strengthen the existing hardness results by creating "gap-instances" under stronger hypotheses such as the (Gap-)Exponential Time Hypothesis (ETH). Formally, we show the following results:

Cutting paths between a set of terminal pairs, i.e., Directed Multicut: Pilipczuk and Wahlstrom [TOCT '18] showed that Directed Multicut is W[1]-hard when parameterized by p if $k = 4$. We complement this by showing the following two results:

- Directed Multicut has a $k/2$-approximation in $2^{O(p^2)} \cdot n^{O(1)}$ time (i.e., a 2-approximation if $k = 4$),
- Under Gap-ETH, Directed Multicut does not admit an $(\frac{59}{58} - \epsilon)$-approximation in $f(p) \cdot n^{O(1)}$ time, for any computable function f, even if $k = 4$.

Connecting a set of terminal pairs, i.e., Directed Steiner Network (DSN): The DSN problem on general graphs is known to be W[1]-hard parameterized by $p + k$ due to Guo et al. [SIDMA '11]. Dinur and Manurangsi [ITCS '18] further showed that there is no FPT $k^{1/4-o(1)}$-approximation algorithm parameterized by k, under Gap-ETH. Chitnis et al. [SODA '14] considered the restriction to special graph classes, but unfortunately this does not lead to FPT algorithms either: DSN on planar graphs is W[1]-hard parameterized by k. In this paper we consider the $\text{DSN}_{\text{Planar}}$ problem which is an intermediate version: the graph is general, but we want to find a solution whose cost is at most that of an optimal planar solution (if one exists). We show the following lower bounds for $\text{DSN}_{\text{Planar}}$:

- $\text{DSN}_{\text{Planar}}$ has no $(2 - \epsilon)$-approximation in FPT time parameterized by k, under Gap-ETH. This answers in the negative a question of Chitnis et al. [ESA '18].
- $\text{DSN}_{\text{Planar}}$ is W[1]-hard parameterized by $k + p$. Moreover, under ETH, there is no $(1 + \epsilon)$-approximation for $\text{DSN}_{\text{Planar}}$ in $f(k, p, \epsilon) \cdot n^{o(k+\sqrt{p+1/\epsilon})}$ time for any computable function f.

Pairwise connecting a set of terminals, i.e., Strongly Connected Steiner Subgraph (SCSS): Guo et al. [SIDMA '11] showed that SCSS is W[1]-hard parameterized by $p + k$, while Chitnis et al. [SODA '14] showed that SCSS remains W[1]-hard parameterized by p, even if the input graph is planar. In this paper we consider the $\text{SCSS}_{\text{Planar}}$ problem which is an intermediate version: the graph is general, but we want to find a solution whose cost is at most that of an optimal planar solution (if one exists). We show the following lower bounds for $\text{SCSS}_{\text{Planar}}$:

- $\text{SCSS}_{\text{Planar}}$ is W[1]-hard parameterized by $k + p$. Moreover, under ETH, there is no $(1 + \epsilon)$-approximation for $\text{SCSS}_{\text{Planar}}$ in $f(k, p, \epsilon) \cdot n^{o(\sqrt{k+p+\frac{1}{\epsilon}})}$ time for any computable function f.

Previously, the only known FPT approximation results for SCSS applied to general graphs parameterized by k: a 2-approximation by Chitnis et al. [IPEC '13], and a matching $(2 - \epsilon)$-hardness under Gap-ETH by Chitnis et al. [ESA '18].

th International Symposium on Parameterized and Exact Computation (IPEC 2019).
Editors: Bart M. P. Jansen and Jan Arne Telle; Article No. 8; pp. 8:1–8:20

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

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Directed graphs, cuts, connectivity, Steiner problems, FPT inapproximability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.8

Related Version Full version of the paper is available at `http://arxiv.org/abs/1910.01934`.

Funding *Rajesh Chitnis*: Work done while at University of Warwick, UK and supported by ERC grant 2014-CoG 647557.
Andreas Emil Feldmann: Supported by the Czech Science Foundation GAČR (grant #19-27871X), and by the Center for Foundations of Modern Computer Science (Charles Univ. project UNCE/SCI/004).

Acknowledgements We thank Pasin Manurangsi for helpful discussions.

1 Introduction

Given a weighted directed graph $G = (V, E)$ with two terminal vertices s, t the problems of finding a minimum weight $s \rightsquigarrow t$ cut and a minimum weight $s \rightsquigarrow t$ path can both be famously solved in polynomial time. There are two natural generalizations when we consider more than two terminals: either we look for connectivity/cuts between all terminals of a given set, or we look for connectivity/cuts between a given set of terminal pairs. This leads to the four problems of Directed Multiway Cut, Directed Multicut, Strongly Connected Steiner Subgraph and Directed Steiner Network:

- **Cutting all paths between a set of terminals**: In the Directed Multiway Cut problem, we are given a set of terminals $T = \{t_1, t_2, \ldots, t_k\}$ and the goal is to find a minimum weight subset $X \subseteq V$ such that $G \setminus X$ has no $t_i \rightsquigarrow t_j$ path for any $1 \leq i \neq j \leq k$.
- **Cutting paths between a set of terminal pairs**: In the Directed Multicut problem, we are given a set of terminal pairs $T = \{(s_i, t_i)\}_{i=1}^{k}$ and the goal is to find a minimum weight subset $X \subseteq V$ such that $G \setminus X$ has no $s_i \rightsquigarrow t_i$ path for any $1 \leq i \leq k$.
- **Connecting all terminals of a given set**: In the Strongly Connected Steiner Subgraph (SCSS) problem, we are given a set of terminals $T = \{t_1, t_2, \ldots, t_k\}$ and the goal is to find a minimum weight subset $X \subseteq V$ such that $G[X]$ has a $t_i \rightsquigarrow t_j$ path for every $1 \leq i \neq j \leq k$.
- **Connecting a set of terminal pairs**: In the Directed Steiner Network (DSN) problem, we are given a set of terminal pairs $T = \{(s_i, t_i)\}_{i=1}^{k}$ and the goal is to find a minimum weight subset $X \subseteq V$ such that $G[X]$ has an $s_i \rightsquigarrow t_i$ path for every $1 \leq i \leq k$.

All four of the aforementioned problems are known to be NP-hard, even for small values of k. One way to cope with NP-hardness is to try to design polynomial time approximation algorithms with small approximation ratio. However, apart from Directed Multiway Cut, which admits a 2-approximation in polynomial time [35], all the other three problems are known to have strong lower bounds (functions of n) on the approximation ratio of polynomial time algorithms [16, 19, 25]. Another way to cope with NP-hardness is to try to design FPT algorithms. However, apart from Directed Multiway Cut which has an FPT algorithm parameterized by the size p of the cutset, all the other three problems are known to be W[1]-hard (and hence fixed-parameter intractable) parameterized by size p of the solution X plus the number k of terminals/terminal pairs. When neither of the paradigms of polynomial time approximation algorithms nor (exact) FPT algorithm seem to be successful, the next natural alternative is to try to design FPT approximation algorithms or show hardness of FPT approximation results.

In this paper, we consider the remaining three problems of Directed Multicut, Strongly Connected Steiner Subgraph and Directed Steiner Network, for which strong approximation and parameterized lower bounds exist, from the viewpoint of FPT approximation algorithms. We obtain two types of results for these three problems: the first is to circumvent the W[1]-hardness and polynomial-time inapproximability results for these problems by designing FPT approximation algorithms, and the second is to strengthen the existing W[1]-hardness by creating "gap-instances" under stronger hypotheses than FPT$\neq$ W[1] such as (Gap-) Exponential Time Hypothesis (ETH). Throughout, we use k to denote number of terminals or terminal pairs and p to denote size of the solution. First, in Section 1.1, we give a brief overview of the current state-of-the-art results for each the three problems from the lens of polynomial time approximation algorithms, FPT algorithms, and FPT approximation algorithms followed by the formal statements of our results. Then, in Section 1.2 we describe the recent flux of results which have set up the framework of FPT hardness of approximation under (Gap-)ETH, and how we use it obtain our hardness results in this paper.

1.1 Previous work and our results

The Directed Multicut problem

Garg et al. [23] showed that Directed Multicut is NP-hard even for $k = 2$. The current best approximation ratio in terms of n is $O(n^{11/23} \cdot \log^{O(1)} n)$ due to Agarwal et al. [1], and it is known that Directed Multicut is hard to approximate in polynomial time to within a factor of $2^{\Omega(\log^{1-\epsilon} n)}$ for any constant $\epsilon > 0$, unless NP $\subseteq$ ZPP [16]. There is a simple k-approximation in polynomial time obtained by solving each terminal pair as a separate instance of min $s \rightsquigarrow t$ cut and then taking the union of all the k cuts. Chekuri and Madan [8] and later Lee [30] showed that this is tight: assuming the Unique Games Conjecture of Khot [28], it is not possible to approximate Directed Multicut better than factor k in polynomial time, for any fixed k. On the FPT side, Marx and Razgon [34] showed that Directed Multicut is W[1]-hard paramterized by p. For the case of bounded k, Chitnis et al. [14] showed that Directed Multicut is FPT parameterized by p when $k = 2$, but Pilipczuk and Wahlstrom [36] showed that the problem remains W[1]-hard parameterized by p when $k = 4$. The status of Directed Multicut parameterized by p when $k = 3$ is an outstanding open question. We first obtain the following FPT approximation for Directed Multicut parameterized by p, which beats any approximation obtainable when parameterizing by k (even in XP time) according to [8, 30]:

▶ **Theorem 1.** *The* Directed Multicut *problem admits an* $\lceil k/2 \rceil$*-approximation in* $2^{O(p^2)} \cdot n^{O(1)}$ *time.*

The proof of the above theorem uses the FPT algorithm of Chitnis et al. [14, 12] for Directed Multiway Cut parameterized by p as a subroutine. Note that Theorem 1 gives an FPT 2-approximation for Directed Multicut With 4 Pairs. We complement this upper bound with a constant factor lower bound for approximation ratio of any FPT algorithm for Directed Multicut With 4 Pairs.

▶ **Theorem 2.** *Under Gap-ETH, for any* $\varepsilon > 0$ *and any computable function* f*, there is no* $f(p) \cdot n^{O(1)}$ *time algorithm that computes an* $(\frac{59}{58} - \varepsilon)$*-approximation for* Directed Multicut With 4 Pairs.

We did not optimize the constant 59/58 in order to keep the analysis simple: we believe it can be easily improved, but our techniques would not take it close to the upper bound of 2.

The Directed Steiner Network (DSN) problem

The DSN problem is known to be NP-hard, and furthermore even computing an $O(2^{\log^{1-\varepsilon} n})$-approximation is not possible [19] in polynomial time, unless NP $\subseteq$ DTIME($n^{\text{polylog}(n)}$). The best known approximation factors for polynomial time algorithms are $O(n^{2/3+\varepsilon})$ and $O(k^{1/2+\varepsilon})$ [4, 7, 21]. On the FPT side, Feldman and Ruhl [20] designed an $n^{O(k)}$ algorithm for DSN (cf. [22]). Chitnis et al. [15] showed that the Feldman-Ruhl algorithm is tight: under ETH, there is no $f(k) \cdot n^{o(k)}$ algorithm (for any computable function f) for DSN even if the input graph is a planar directed acyclic graph. Guo et al. [24] showed that DSN remains W[1]-hard even when parameterized by the larger parameter $k + p$. Dinur and Manurangsi [18] further showed that DSN on general graphs has no FPT approximation algorithm with ratio $k^{1/4-o(1)}$ when parameterized by k, under Gap-ETH.

Chitnis et al. [11] considered two relaxations of the Directed Steiner Network problem: the bi-DSN problem where the input graph is bidirected[1], and the $\text{DSN}_{\text{Planar}}$ problem where the input graph is general but the goal is to find a solution whose cost is at most that of an optimal planar solution (if one exists). The main result of Chitnis et al. [11] is that although bi-$\text{DSN}_{\text{Planar}}$ (i.e., the intersection of bi-DSN and $\text{DSN}_{\text{Planar}}$) is W[1]-hard parameterized by $k + p$, it admits a *parameterized approximation scheme*: for any $\varepsilon > 0$, there is a $\max\{2^{k^{2^{O(1/\varepsilon)}}}, n^{2^{O(1/\varepsilon)}}\}$ time algorithm for bi-$\text{DSN}_{\text{Planar}}$ which computes a $(1 + \varepsilon)$-approximation. Such a parameterized approximation is not possible for bi-DSN as Chitnis et al. [11] showed that under Gap-ETH there is a constant $\alpha > 0$ such that there is no FPT α-approximation. They asked whether a parameterized approximation scheme for the remaining variant of DSN, i.e., the $\text{DSN}_{\text{Planar}}$ problem, exists. We answer this question in the negative with the following lower bound

▶ **Theorem 3.** *Under Gap-ETH, for any $\varepsilon > 0$ and any computable function f, there is no $f(k) \cdot n^{O(1)}$ time algorithm that computes a $(2 - \varepsilon)$-approximation for $\text{DSN}_{\text{Planar}}$, even if the input graph is a directed acyclic graph (DAG).*

The W[1]-hardness proof of [15] for DSN on planar graphs parameterized by k does not give hardness parameterized by p since in that reduction the value of p grows with n. Our next result shows that the slightly more general problem of $\text{DSN}_{\text{Planar}}$ (here the input graph is general, but we want to find a solution of cost $\leq p$ if there is a planar solution of size $\leq p$) is indeed W[1]-hard parameterized by $k+p$. Also we obtain a lower bound for approximation schemes for this problem under ETH, i.e., under a weaker assumption than the one used for Theorem 3.[2]

▶ **Theorem 4.** *The $\text{DSN}_{\text{Planar}}$ problem is W[1]-hard parameterized by $p+k$, even if the input graph is a directed acyclic graph (DAG). Moreover, under ETH, for any computable function f*

- *there is no $f(k,p) \cdot n^{o(k+\sqrt{p})}$ time algorithm for $\text{DSN}_{\text{Planar}}$, and*
- *there is no $f(k,\varepsilon,p) \cdot n^{o(k+\sqrt{p+1/\epsilon})}$ time algorithm which computes a $(1+\varepsilon)$-approximation for $\text{DSN}_{\text{Planar}}$ for every $\varepsilon > 0$.*

Note that just the W[1]-hardness of $\text{DSN}_{\text{Planar}}$ parameterized by $k + p$ already follows from [11] who showed that even the special case of bi-$\text{DSN}_{\text{Planar}}$ is W[1]-hard parameterized by $k+p$. However, this reduction from [11] was from ℓ-Clique to an instance of bi-$\text{DSN}_{\text{Planar}}$ with $k = O(\ell^2)$ and $p = O(\ell^5)$, whereas Theorem 4 gives a reduction from ℓ-Clique to $\text{DSN}_{\text{Planar}}$ with $k = O(\ell)$ and $p = O(\ell^2)$. This gives much improved lower bounds on the running times.

---

[1] Bidirected graphs are directed graphs which have the property that for every edge $u \to v$ in G the reverse edge $v \to u$ exists in G as well and moreover has the same weight as $u \to v$.

[2] In the following, $o(f(k,p,\varepsilon))$ means any function $g(f(k,p,\varepsilon))$ such that $g(x) \in o(x)$.

The Strongly Connected Steiner Subgraph (SCSS) problem

The SCSS problem is NP-hard, and the best known approximation ratio in polynomial time for SCSS is k^ϵ for any $\epsilon > 0$ [6]. A result of Halperin and Krauthgamer [25] implies SCSS has no $\Omega(\log^{2-\epsilon} n)$-approximation for any $\epsilon > 0$, unless NP has quasi-polynomial Las Vegas algorithms. On the FPT side, Feldman and Ruhl [20] designed an $n^{O(k)}$ algorithm for SCSS (cf. [22]). Chitnis et al. [15] showed that the Feldman-Ruhl algorithm is almost optimal: under ETH, there is no $f(k) \cdot n^{o(k/\log k)}$ algorithm (for any computable function f) for SCSS. Guo et al. [24] showed that SCSS remains W[1]-hard even when parameterized by the larger parameter $k + p$. Chitnis et al. [11] showed that the SCSS problem restricted to bidirected graphs remains NP-hard, but is FPT parameterized by k. The SCSS problem admits a *square-root phenomenon* on planar graphs: Chitnis et al. [15] showed that SCSS on planar graphs has an $2^{O(k \log k)} \cdot n^{O(\sqrt{k})}$ algorithm, and under ETH there is a tight lower bound of $f(k) \cdot n^{o(\sqrt{k})}$ for any computable function f. The W[1]-hardness proof of [15] for SCSS on planar graphs parameterized by k does not give hardness parameterized by p, since in that reduction the value of p grows with n. Our next result shows that the slightly more general problem of $\text{SCSS}_{\text{PLANAR}}$ (here the input graph is general, but we want to find a solution of cost $\leq p$ if there is a planar solution of size $\leq p$) is indeed W[1]-hard parameterized by $k + p$. We also obtain a lower bound for approximation schemes for this problem under ETH:

▶ **Theorem 5.** *The* $\text{SCSS}_{\text{PLANAR}}$ *problem is* W[1]*-hard parameterized by* $p + k$*. Moreover, under ETH, for any computable function* f

- *there is no* $f(k,p) \cdot n^{o(\sqrt{k+p})}$ *time algorithm for* $\text{SCSS}_{\text{PLANAR}}$*, and*
- *there is no* $f(k,\varepsilon,p) \cdot n^{o(\sqrt{k+p+1/\epsilon})}$ *time algorithm which computes an* $(1+\varepsilon)$*-approximation for* $\text{SCSS}_{\text{PLANAR}}$ *for every* $\varepsilon > 0$*.*

To the best of our knowledge, the only known FPT approximation results for SCSS applied to general graphs parameterized by k: a simple FPT 2-approximation due to Chitnis et al. [13], and a matching $(2 - \epsilon)$-hardness (for any constant $\epsilon > 0$) under Gap-ETH due to Chitnis et al. [11].

1.2 FPT inapproximability results under (Gap-)ETH

A standard hypothesis for showing lower bounds for running times of FPT and exact exponential time algorithms is the Exponential Time Hypothesis (ETH) of Impagliazzo and Paturi [26].

▶ **Hypothesis 6. Exponential Time Hypothesis (ETH)***: There exists a constant* $\delta > 0$ *such that no algorithm can decide whether any given* 3*-CNF formula is satisfiable in time* $O(2^{\delta m})$ *where* m *denotes the number of clauses.*

The original conjecture stated the lower bound as exponential in terms of the number of variables not clauses, but the above statement follows from the Sparsification Lemma of [27]. The Exponential Time Hypothesis has been used extensively to show a variety of lower bounds including those for FPT algorithms, exact exponential time algorithms, hardness of polynomial time approximation, and hardness of FPT approximation. We refer the interested reader to [31] for a survey on lower bounds based on ETH.

To show the W[1]-hardness of $\text{DSN}_{\text{PLANAR}}$ (Theorem 4) and $\text{SCSS}_{\text{PLANAR}}$ (Theorem 5) parameterized by $k + p$ we design parameterized reductions from ℓ-Clique to these problems such that $\max\{k, p\}$ is upper bounded by a function of ℓ. Furthermore, by choosing ϵ to be small enough such that computing an $(1 + \epsilon)$-approximation is the same as computing the

optimal solution, we also obtain runtime lower bounds for $(1+\epsilon)$-approximations for these two problems by translating the $f(\ell) \cdot n^{o(\ell)}$ lower bound for ℓ-Clique [9] under ETH (for any computable function f).

Recently, a *gap version* of the ETH was proposed:

▶ **Hypothesis 7. Gap-ETH** *[17, 32]: There exists a constant $\delta > 0$ such that, given a* 3CNF *formula Φ on n variables, no $2^{o(n)}$-time algorithm can distinguish between the following two cases correctly with probability at least 2/3:*

- *Φ is satisfiable.*
- *Every assignment to the variables violates at least a δ-fraction of the clauses of Φ.*

It is known [5, 2] that Gap-ETH follows from ETH given other standard conjectures, such as the existence of linear sized PCPs or exponentially-hard locally-computable one-way functions. We refer the interested reader to [17, 5] for a discussion on why Gap-ETH is a plausible assumption. In a breakthrough result, Chalermsook et al. [5] used Gap-ETH to show that the two famous parameterized intractable problems of Clique and Set Cover are completely inapproximable in FPT time parameterized by the size of the solution. In this paper, we obtain two hardness of approximation results (Theorem 2 and Theorem 3) based on Gap-ETH. The starting point of our hardness of approximation results are based on the recent results on parameterized inapproximability of the Densest k-Subgraph problem. Recall that, in the Densest k-Subgraph (DkS) problem [29], we are given an undirected graph $G = (V, E)$ and an integer k and the goal is to find a subset $S \subseteq V$ of size ℓ that induces as many edges in G as possible. Chalermsook et al. [5] showed that, under randomized Gap-ETH, there is no FPT approximation (parameterized by k) with ratio $k^{o(1)}$. This was improved recently by Dinur and Manurangsi [18] who showed better hardness and under deterministic Gap-ETH. We state their result formally[3]:

▶ **Theorem 8** ([18, Theorem 2])**.** *Under Gap-ETH, for any function $h(\ell) = o(1)$, there is no $f(\ell) \cdot n^{O(1)}$-time algorithm that, given a graph G on n vertices and an integer k, can distinguish between the following two cases:*

- *(YES) G contains at least one ℓ-clique as a subgraph.*
- *(NO) Every ℓ-subgraph of G contains less than $\ell^{h(\ell)-1} \cdot \binom{\ell}{2}$ edges.*

Note that this result is essentially tight: there is a simple $O(\ell)$ approximation since the number of edges induced by a ℓ-vertex subgraph is at most $\binom{\ell}{2}$ and at least $\lfloor \ell/2 \rfloor$ (without loss of generality, we can assume there are no isolated vertices). Instead of working with DkS, we will reduce from a "colored" version of the problem called Maximum Colored Subgraph Isomorphism, which can be defined as follows.

Maximum Colored Subgraph Isomorphism (MCSI)
Input : An instance Γ of MCSI consists of three components:
- An undirected graph $G = (V_G, E_G)$,
- A partition of vertex set V_G into disjoint subsets $V_1, \ldots, V_\ell$,
- An undirected graph $H = (V_H = \{1, \ldots, \ell\}, E_H)$.

Goal: Find an assignment $\phi : V_H \to V_G$ where $\phi(i) \in V_i$ for every $i \in [\ell]$ that maximizes the number of edges $i - j \in E_H$ such that $\phi(i) - \phi(j) \in E_G$.

[3] Dinur and Manurangsi [18] actually state their result for 2-CSPs

This problem is referred to as Label Cover in the hardness of approximation literature [3]. However, Chitnis et al. [11] used the name Maximum Colored Subgraph Isomorphism to be consistent with the naming conventions in the FPT community: this problem is an optimization version of Colored Subgraph Isomorphism [33]. The graph H is sometimes referred to as the *supergraph* of Γ. Similarly, the vertices and edges of H are called *supernodes* and *superedges* of Γ. Moreover, the size of Γ is defined as $n = |V_G|$, the number of vertices of G. Additionally, for each assignment ϕ, we define its value $\text{val}(\phi)$ to be the fraction of superedges $i - j \in E_H$ such that $\phi(i) - \phi(j) \in E_G$; such superedges are said to be *covered* by ϕ. The objective of MCSI is now to find an assignment ϕ with maximum value. We denote the value of the optimal assignment by $\text{val}(\Gamma)$, i.e., $\text{val}(\Gamma) = \max_\phi \text{val}(\phi)$.

Using Theorem 8 we derive the following two corollaries regarding hardness of approximation for Maximum Colored Subgraph Isomorphism when the supergraph H has special structure. These corollaries follow quite straightforwardly from Theorem 8 using the idea of splitters, but we provide proofs in the full version [10] for completeness.

▶ **Corollary 9.** [⋆][4] *Assuming Gap-ETH, for any function $h(\ell) = o(1)$, there is no $f(\ell)\cdot n^{O(1)}$-time algorithm that, given a* MCSI *instance Γ of size n such that the supergraph $H = K_\ell$, can distinguish between the following two cases:*

- *(YES)* $\text{val}(\Gamma) = 1$.
- *(NO)* $\text{val}(\Gamma) < \ell^{h(\ell)-1}$

▶ **Corollary 10.** [⋆] *Assuming Gap-ETH, for any function $h(\ell) = o(1)$, there is no $f(\ell)\cdot n^{O(1)}$-time algorithm that, given a* MCSI *instance Γ of size n such that the supergraph H is the complete bipartite subgraph $K_{\frac{\ell}{2},\frac{\ell}{2}}$, can distinguish between the following two cases:*

- *(YES)* $\text{val}(\Gamma) = 1$.
- *(NO)* $\text{val}(\Gamma) < \ell^{h(\ell)-1}$.

We prove Theorem 2 and Theorem 3 via reductions from Corollary 9 and Corollary 10 resepctively.

2 FPT (In)Approximability of Directed Multicut

In this section we design an FPT 2-approximation for Directed Multicut With 4 Pairs parameterized by p (Section 2.1) and complement this with a lower bound (Section 2.2) showing that no FPT algorithm (parameterized by p) for Directed Multicut With 4 Pairs can achieve a ratio of $(\frac{59}{58} - \epsilon)$ under Gap-ETH.

2.1 FPT approximation algorithm

It is well-known that a k-approximation can be computed in polynomial time by taking union of min cuts of each of the k terminal pairs. Chekuri and Madan [8] and later Lee [30] showed that this approximation ratio is best-possible for polynomial time algorithms under the Unique Games Conjecture of Khot [28]. The same lower bound also applies for any constant k, i.e., even an XP algorithm parameterized by k cannot compute a better approximation than a polynomial time algorithm. We now design an FPT $\lceil k/2 \rceil$-approximation for Directed Multicut. The idea is borrowed from the proof of Chitnis et al. [14] that Directed Multicut With 2 Pairs is FPT parameterized by p.

[4] All proofs labelled with [⋆] appear in the full version [10]

▶ **Theorem 1.** *The* Directed Multicut *problem admits a* $\lceil k/2 \rceil$*-approximation in* $2^{O(p^2)} \cdot n^{O(1)}$ *time. Formally, the algorithm takes an instance* $(G, \mathcal{T})$ *of* Directed Multicut *and in* $2^{O(p^2)} \cdot n^{O(1)}$ *time either concludes that there is no solution of cost at most* p*, or produces a solution of cost at most* $p\lceil k/2 \rceil$*.*

Proof. Let the pairs be $\mathcal{T} = \{(s_i, t_i) \ : \ 1 \leq i \leq k\}$, and let OPT be the optimum value for the instance $(G, \mathcal{T})$ of Directed Multicut. For now, assume that k is even. Introduce $k/2$ new vertices r_j, q_j, for $1 \leq j \leq k/2$, of weight $p+1$ each, and add the following edges:

- $r_j \to s_{2j-1}$ and $t_{2j-1} \to q_j$
- $q_j \to s_{2j}$ and $t_{2j} \to r_j$

Let the resulting graph be G', and note that G has an $s_i \to t_i$ path for some $1 \leq i \leq k$ if and only if G' has a $q_{i/2} \to r_{i/2}$ or $r_{(i-1)/2} \to q_{(i-1)/2}$ path (depending on whether i is even or odd). Since the vertices r_j, q_j have weight $p+1$ each, it follows that G has a solution of size at most p for the instance $(G, \{(s_{2j-1}, t_{2j-1}), (s_{2j}, t_{2j})\})$ of Directed Multicut if and only if G' has a solution of size at most p for the Directed Multiway Cut instance with input graph G and terminals r_j, q_j. We use the algorithm of Chitnis et al. [14, 12] for Directed Multiway Cut which checks in $2^{O(p^2)} \cdot n^{O(1)}$ time[5] if there is a solution of cost at most p. If there is no solution of cost at most p between r_j and q_j in G' then this implies that G has no cut of size at most p separating (s_{2j-1}, t_{2j-1}) and (s_{2j}, t_{2j}) and hence OPT $> p$. Otherwise, there is a cut C_j in G of cost at most p which separates (s_{2j-1}, t_{2j-1}) and (s_{2j}, t_{2j}).

The output of the algorithm is the cut $C = \bigcup_{j=1}^{k/2} C_j$. Clearly, if k is even then C is a feasible solution for the instance $(G, \mathcal{T})$ of Directed Multicut with cost at most $\sum_{j=1}^{k/2} \texttt{cost}(C_j) \leq pk/2$. In case k is odd we use the above procedure for the terminal pairs $\{(s_i, t_i) : 1 \leq i \leq k-1\}$, and finally add a min cut between the last terminal pair (s_k, t_k). This results in the desired $\lceil k/2 \rceil$-approximation. ◀

2.2 No FPT $(\frac{59}{58} - \epsilon)$-approximation under Gap-ETH

With the parameterized hardness of approximating MCSI ready, we can now prove our hardness results for Directed Multicut with 4 terminal pairs.

▶ **Theorem 2.** *Under Gap-ETH, for any* $\varepsilon > 0$ *and any computable function* f*, there is no* $f(p) \cdot n^{O(1)}$ *time algorithm that computes an* $(\frac{59}{58} - \varepsilon)$*-approximation for* Directed Multicut With 4 Pairs*.*

Our proof of the parameterized inapproximability of Directed Multicut With 4 Pairs is based on a reduction from Maximum Colored Subgraph Isomorphism whose properties are described below.

▶ **Lemma 11.** *There exists a polynomial time reduction that, given an instance* $\Gamma = (G, K_\ell, V_1 \cup \cdots \cup V_\ell)$ *of* MCSI *, produces an instance* $(G', \mathcal{T}')$ *of* Directed Multicut With 4 Pairs *such that*

- *(***Completeness***): If* $\text{val}(\Gamma) = 1$*, then there exists a solution* $N \subseteq V(G')$ *of cost* $29\ell^2$ *for the instance* $(G', \mathcal{T}')$ *of* Directed Multicut With 4 Pairs
- *(***Soundness***): If* $\text{val}(\Gamma) < \frac{1}{10}$*, then every solution* $N \subseteq V(G')$ *for the instance* $(G', \mathcal{T}')$ *of* Directed Multicut With 4 Pairs *has cost more than* $29.5\ell^2$*.*
- *(Parameter Dependency): The size of the solution is* $p = O(\ell^2)$*.*

[5] This is independent of number of the terminals

In the proof of Lemma 11, we actually use the same reduction as from [36], but with different weights. We reduce to the vertex-weighted variant of Directed Multicut With 4 Pairs where we have four different types of weights for the vertices:

- *light* vertices (shown using gray color) which have weight $B = \frac{\ell^2}{\binom{\ell}{2}}$
- *medium* vertices (shown using green color) which have weight $2B$
- *heavy* vertices (shown using orange color) which have weight 20ℓ
- *super-heavy* vertices (shown using white color) which have weight $100\ell^2$

2.2.1 Construction of the Directed Multicut With 4 Pairs instance

Without loss of generality (by adding isolated vertices if necessary) we can assume that $|V_i| = n$ for each $i \in [\ell]$. For each $i \in [\ell]$ let $V_i = \{v_1^i, v_2^i, v_3^i, \ldots, v_n^i\}$. Then $|V(G)| = n\ell$. We now describe the construction of the (vertex-weighted) Directed Multicut With 4 Pairs instance $(G', \mathcal{T}')$.

- Introduce eight terminals, arranged in four terminal pairs as follows:

  $$\mathcal{T}' = \{(s^x_{0\to n}, t^x_{0\to n}), \quad (s^y_{0\to n}, t^y_{0\to n}), \quad (s^<_{n\to 0}, t^<_{n\to 0}), \quad (s^>_{n\to 0}, t^>_{n\to 0})\}$$

  Each of the 8 terminals is super-heavy.
- For every $1 \leq i \leq \ell$, we introduce a bidirected path on $2n+1$ vertices (see Figure 2)

  $$Z_i := z_0^i \leftrightarrow \hat{z}_1^i \leftrightarrow z_1^i \leftrightarrow \hat{z}_2^i \leftrightarrow z_2^i \leftrightarrow \ldots \leftrightarrow \hat{z}_n^i \leftrightarrow z_n^i,$$

  called henceforth the *z-path for color class i*. For each $0 \leq a \leq n$ the vertex z_a^i is super-heavy and for each $1 \leq a \leq n$ the vertex $\hat{z}_a^i$ is heavy.
- For every pair (i,j) where $1 \leq i,j \leq \ell$, $i \neq j$, we introduce two bidirected paths (see Figure 2 and Figure 1) on $2n+1$ vertices

  $$X_{i,j} := x_0^{i,j} \leftrightarrow \hat{x}_1^{i,j} \leftrightarrow x_1^{i,j} \leftrightarrow \hat{x}_2^{i,j} \leftrightarrow x_2^{i,j} \leftrightarrow \ldots \leftrightarrow \hat{x}_n^{i,j} \leftrightarrow x_n^{i,j}$$

  and

  $$Y_{i,j} := y_0^{i,j} \leftrightarrow \hat{y}_1^{i,j} \leftrightarrow y_1^{i,j} \leftrightarrow \hat{y}_2^{i,j} \leftrightarrow y_2^{i,j} \leftrightarrow \ldots \leftrightarrow \hat{y}_n^{i,j} \leftrightarrow y_n^{i,j}$$

  We call these paths *the x-path and the y-path for the pair* (i,j). For each $0 \leq a \leq n$ the vertices $x_a^{i,j}$ and $y_a^{i,j}$ are super-heavy. For each $1 \leq a \leq n$ the vertices $\hat{x}_a^{i,j}$ and $\hat{y}_a^{i,j}$ are medium.
- For every pair (i,j) with $1 \leq i,j \leq \ell$, $i \neq j$, and every $0 \leq a \leq n$, we add arcs $(x_a^{i,j}, z_a^i)$ and $(z_a^i, y_a^{i,j})$. See Figure 2 for an illustration.
- Furthermore, we attach terminals to the paths as follows: (shown using magenta edges in Figure 1 and Figure 2)
  - for every pair (i,j) with $1 \leq i,j \leq \ell$, $i \neq j$, we add arcs $(s^x_{0\to n}, x_0^{i,j})$ and $(y_n^{i,j}, t^y_{0\to n})$;
  - for every $1 \leq i \leq \ell$ we add arcs $(s^y_{0\to n}, z_0^i)$ and $(z_n^i, t^x_{0\to n})$;
  - for every pair (i,j) with $1 \leq i < j \leq \ell$ we add arcs $(s^<_{n\to 0}, x_n^{i,j})$ and $(y_0^{i,j}, t^<_{n\to 0})$;
  - for every pair (i,j) with $\ell \geq i > j \geq 1$ we add arcs $(s^>_{n\to 0}, x_n^{i,j})$ and $(y_0^{i,j}, t^>_{n\to 0})$.
- For every pair (i,j) with $1 \leq i < j \leq \ell$ we introduce an acyclic $n \times n$ grid $P_{i,j}$ with vertices $p_{a,b}^{i,j}$ for $1 \leq a,b \leq n$ and arcs $(p_{a,b}^{i,j}, p_{a+1,b}^{i,j})$ for every $1 \leq a < n$ and $1 \leq b \leq n$, as well as $(p_{a,b}^{i,j}, p_{a,b+1}^{i,j})$ for every $1 \leq a \leq n$ and $1 \leq b < n$. We call this grid $P_{i,j}$ as the *p-grid for the pair* (i,j). We set the vertex $p_{a,b}^{i,j}$ to be a light vertex if $v_a^i v_b^j \in E(G)$, and super-heavy otherwise. Finally, for every $1 \leq a \leq n$ we introduce the following arcs (shown as dotted in Figure 1):

  $$(x_a^{i,j}, p_{a,1}^{i,j}), \quad (p_{a,n}^{i,j}, y_{a-1}^{i,j}), \quad (x_a^{j,i}, p_{1,a}^{i,j}), \quad (p_{n,a}^{i,j}, y_{a-1}^{j,i}).$$

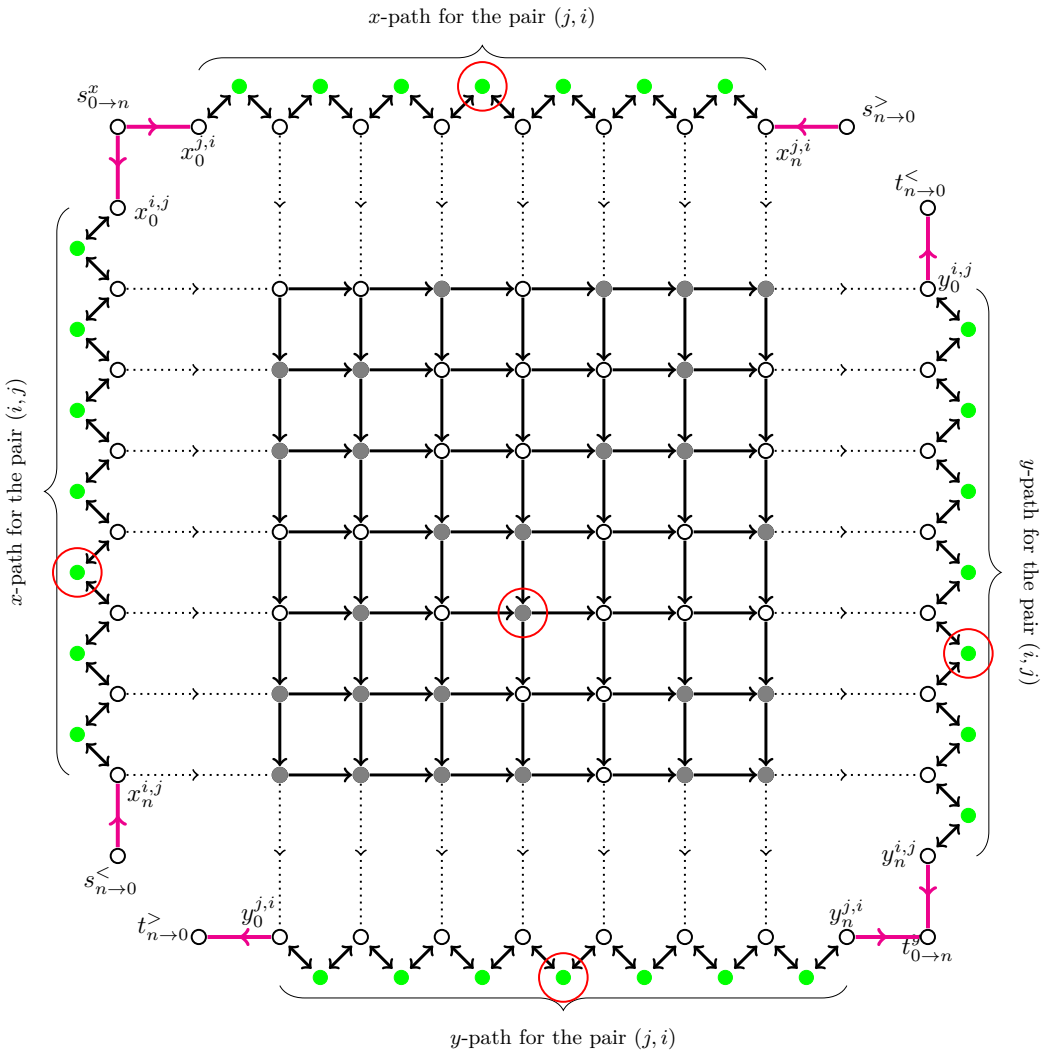


Figure 1 Illustration of the reduction for Directed Multicut With 4 Pairs. For $1 \leq i < j \leq \ell$, the grid $P_{i,j}$ is surrounded by the bidirectional paths $X_{i,j}$ on the left, $X_{j,i}$ on the top, $Y_{i,j}$ on the right and $Y_{j,i}$ on the bottom. Edges incident on terminals are shown in magenta. Green vertices are medium, orange vertices are heavy and white vertices are super-heavy. A desired solution is marked by red circles.

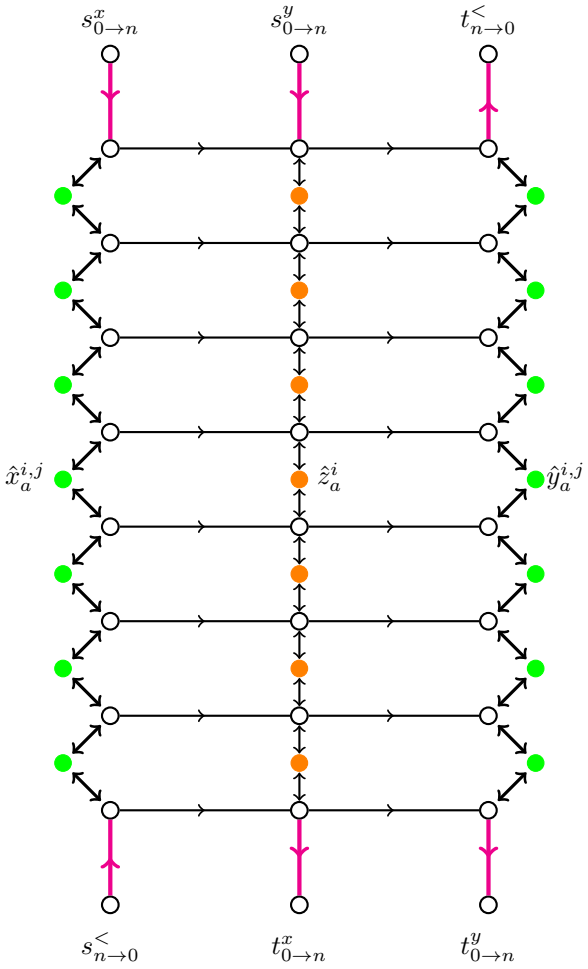


Figure 2 Illustration of the reduction for Directed Multicut With 4 Pairs. For every $1 \leq i < j \leq \ell$, the z-path Z_i corresponding to the color class i is surrounded by the bidirectional paths $X_{i,j}$ on the left and $Y_{i,j}$ on the right. Edges incident on terminals are shown in magenta. Green vertices are medium, orange vertices are heavy and white vertices are super-heavy.

This concludes the construction of the instance $(G', \mathcal{T}')$ of Directed Multicut With 4 Pairs. Note that $|V(G')| = (n+\ell)^{O(1)}$, and also G' can be constructed in $(n+\ell)^{O(1)}$ time.

2.2.2 Completeness of Lemma 11: $\mathbf{val(\Gamma) = 1 \Rightarrow}$ Multicut of cost $\leq 29\ell^2$

Suppose that $\text{val}(\Gamma) = 1$, i.e., G has a ℓ-clique which has exactly one vertex in each V_i for $1 \leq i \leq \ell$. Let this clique be given by $\{v^i_{\alpha(i)} : 1 \leq i \leq \ell\}$. Define

$$X = \{\hat{x}^{i,j}_{\alpha(i)}, \hat{y}^{i,j}_{\alpha(i)} : 1 \leq i, j \leq \ell, i \neq j\} \cup \{\hat{z}^i_{\alpha(i)} : 1 \leq i \leq \ell\} \cup \{p^{i,j}_{\alpha(i),\alpha(j)} : 1 \leq i < j \leq \ell\}.$$

Note that X consists of exactly ℓ heavy $\hat{z}^i_{\alpha(i)}$ vertices, $4\binom{\ell}{2}$ medium $\hat{x}^{i,j}_{\alpha(i)}$ and $\hat{y}^{i,j}_{\alpha(i)}$ vertices, and $\binom{\ell}{2}$ light $p^{i,j}_{\alpha(i),\alpha(j)}$ vertices (the fact that $p^{i,j}_{\alpha(i),\alpha(j)}$ is light for every $1 \leq i < j \leq \ell$ follows from the assumption that the vertices $v^i_{\alpha(i)}$ induce a clique in G). Hence, the weight of X is exactly $\ell \cdot 20\ell + \binom{\ell}{2} \cdot (4 \cdot 2B) + \binom{\ell}{2} \cdot B = 20\ell^2 + \binom{\ell}{2} \cdot 9B = 29\ell^2$. As shown in [36], this set X is a cutset for the instance $(G', \mathcal{T}')$ of Directed Multicut With 4 Pairs. The details are deferred to the full version [10].

2.2.3 Soundness of Lemma 11: Multicut of cost $\leq 29.5\ell^2 \Rightarrow \mathbf{val(\Gamma) \geq \frac{1}{10}}$

Let $\mathcal{X}$ be a solution to the instance $(G', \mathcal{T}')$ of Directed Multicut With 4 Pairs such that weight of $\mathcal{X}$ is $29.5\ell^2$. We now show that $\text{val}(\Gamma) \geq \frac{1}{10}$.

▶ **Observation 12.** *Note that every super-heavy vertex has weight $100\ell^2$ and hence $\mathcal{X}$ cannot contain any super-heavy vertex.*

▶ **Lemma 13.** [⋆] *For each $i \in [\ell]$, the solution $\mathcal{X}$ contains at least one heavy vertex from Z_i.*

▶ **Lemma 14.** [⋆] *For each $1 \leq i \neq j \leq \ell$, the solution $\mathcal{X}$ contains at least one medium vertex from $X_{i,j}$ and at least one medium vertex from $Y_{i,j}$.*

▶ **Definition 15.** *An integer $i \in [\ell]$ is* good *if $\mathcal{X}$ contains exactly one heavy vertex from the z-path for the color class i, i.e., $|\mathcal{X} \cap Z_i| = 1$. In this case, we say that $v^i_{\beta_i}$ be the unique vertex from the z-path for class i in the solution $\mathcal{X}$.*

▶ **Lemma 16.** [⋆] *Let* Good $= \{i \in [\ell] \ : \ i \text{ is good}\}$. *Then* $|\text{Good}| \geq \frac{37\ell}{40}$

▶ **Definition 17.** *Let $1 \leq i < j \leq \ell$. We say that the pair (i, j) is* great *if $\mathcal{X}$ contains*

- *exactly one medium vertex from the x-path for the pair (i, j)*
- *exactly one medium vertex from the y-path for the pair (i, j)*
- *exactly one medium vertex from the x-path for the pair (j, i)*
- *exactly one medium vertex from the y-path for the pair (j, i)*
- *exactly one light vertex from the p-grid for the pair (i, j)*

Let Good-Pairs $= \{(i, j) \ : \ 1 \leq i < j \leq \ell, \ i, j \in \text{Good}\}$

▶ **Lemma 18.** [⋆] *Let $1 \leq i < j \leq \ell$. If both i and j are good, and the pair (i, j) is great then $v^i_{\beta_i} - v^j_{\beta_j} \in E(G)$.*

▶ **Definition 19.** *Let $1 \leq i < j \leq \ell$. We define $\mathcal{X}_{i,j} = \mathcal{X} \cap (X_{i,j} \cup X_{j,i} \cup Y_{i,j} \cup Y_{j,i} \cup P_{i,j})$*

▶ **Lemma 20.** [⋆] *Let $1 \leq i < j \leq \ell$ be such that $i, j \in$ Good. Then either*
- *the pair (i, j) is great and weight of $\mathcal{X}_{i,j}$ is exactly $9B$, or*
- *weight of $\mathcal{X}_{i,j}$ is at least $10B$*

▶ **Lemma 21.** [⋆] *Let $\mathcal{E} = \{1 \leq i < j \leq \ell \ : \ i, j \in$ Good *and* (i, j) *is great*$\}$. Then $|\mathcal{E}| \geq \frac{1}{10} \cdot \binom{\ell}{2}$*

Consider the following ℓ-vertex subgraph C: for each $i \in [\ell]$
- if $i \in [\ell]$ is good then add $v^i_{\beta_i}$ to C,
- otherwise add any vertex from V_i into C.

From Lemma 21 it follows that there are at least $\frac{1}{10} \cdot \binom{\ell}{2}$ edges in G which have both endpoints in C, and hence $\mathrm{val}(\Gamma) \geq \frac{1}{10}$

2.3 Finishing the proof of Theorem 2

We again prove by contrapositive. Suppose that, for some constant $\varepsilon > 0$ and for some computable function $f(p)$ independent of n, there exists an $f(p) \cdot n^{O(1)}$-time $(\frac{59}{58} - \varepsilon)$-approximation algorithm for Directed Multicut. Let us call this algorithm $\mathbb{A}$.

We create an algorithm $\mathbb{B}$ that can distinguish between the two cases of Corollary 9 with $h(\ell) = 1 - \frac{\log(10)}{\log \ell} = o(1)$. Our new algorithm $\mathbb{B}$ works as follows. Given an instance $(G, H, V_1 \cup \cdots \cup V_\ell)$ of MCSI where $H = K_\ell$, the algorithm $\mathbb{B}$ uses the reduction from Lemma 11 to create a Directed Multicut With 4 Pairs instance $(G', \mathcal{T}')$ with 4 terminal pairs. $\mathbb{B}$ then runs $\mathbb{A}$ on this instance with $p = 29\ell^2$; if $\mathbb{A}$ returns a solution N of cost less than $29.5\ell^2$, then $\mathbb{B}$ returns YES. Otherwise, $\mathbb{B}$ returns NO.

To see that algorithm $\mathbb{B}$ can indeed distinguish between the YES and NO cases, first observe that, in the YES case the completeness property of Lemma 11 guarantees that the optimal solution has cost at most $29\ell^2$. Since $\mathbb{A}$ is a $(\frac{59}{58} - \varepsilon)$-approximation algorithm, it returns a solution of cost at most $(\frac{59}{58} - \varepsilon) \cdot 29\ell^2 < 29.5\ell^2$: this means that $\mathbb{B}$ outputs YES. On the other hand, if $(G, H, V_1 \cup \cdots \cup V_\ell)$ is a NO instance, i..e, $\mathrm{val}(\Gamma) < \frac{1}{10} = \ell^{h(\ell)-1}$, then the soundness property of Lemma 22 guarantees that the optimal solution in G' has cost more than $29.5\ell^2$ (which is greater than $(\frac{59}{58} - \varepsilon) \cdot 29\ell^2$) and hence $\mathbb{B}$ correctly outputs NO.

Finally, observe that the running time of $\mathbb{B}$ is $f(p) \cdot |V(G')|^{O(1)}$ plus the $(|V(G)| + \ell)^{O(1)}$ time needed to construct G'. Since $|V(G')| = (|V(G) + \ell|)^{O(1)}$ and $p = O(\ell^2)$ it follows that the total running time is $g(\ell) \cdot |V(G)|$ for some computable function g. Hence, from Corollary 9, Gap-ETH is violated.

3 FPT inapproximability for DSN$_{\text{Planar}}$

3.1 $(2 - \varepsilon)$-hardness for FPT approximation under Gap-ETH

The goal of this section is to show the following theorem:

▶ **Theorem 3.** *Under Gap-ETH, for any $\varepsilon > 0$ and any computable function f, there is no $f(k) \cdot n^{O(1)}$ time algorithm that computes a $(2 - \varepsilon)$-approximation for* DSN$_{\text{Planar}}$.

3.1.1 Reduction from Colored Biclique to DSN$_{\text{Planar}}$

▶ **Lemma 22.** *For every constant $\gamma > 0$, there exists a polynomial time reduction that, given an instance $\Gamma = (G, H, V_1 \cup \cdots \cup V_\ell, W_1, W_2, \ldots, W_\ell)$ of* MCSI *where the supergraph H is $K_{\ell,\ell}$, produces an instance $(G', \mathcal{D}')$ of* DSN$_{\text{Planar}}$*, such that*

- **(Completeness)** *If* $\mathrm{val}(\Gamma) = 1$*, then there exists a planar network* $N \subseteq G'$ *of cost* $2(1+\gamma^{1/5})$ *that satisfies all demands.*
- **(Soundness)** *If* $\mathrm{val}(\Gamma) < \gamma$*, then every network* $N \subseteq G'$ *that satisfies all demands has cost more than* $2(2-4\gamma^{1/5})$*.*
- *(Parameter Dependency) The number of demand pairs* $k = |\mathcal{D}'|$ *is* 2ℓ*.*

Lemma 22 is proven as follows: we construct the $\mathrm{DSN}_{\mathrm{PLANAR}}$ instance in Section 3.1.1.2. The proofs of completeness and soundness of the reduction are deferred to the full version [10]. First, we construct a "path gadget" which we use repeatedly in our construction.

3.1.1.1 Constructing a directed "path" gadget

For every integer n we define the following gadget $\mathcal{P}_n$ which contains $2n$ vertices (see Figure 3). Since we need many of these gadgets later on, we will denote vertices of $\mathcal{P}_n$ by $\mathcal{P}_n(v)$ etc., in order to be able to distinguish vertices of different gadgets. All edges will have the same weight B, which we will fix later during the reductions. The gadget $\mathcal{P}_n$ is constructed as follows: $\mathcal{P}_n$ has a directed path of one edge corresponding to each $i \in [n]$. This is given by $\mathcal{P}_n(0_i) \to \mathcal{P}_n(1_i)$

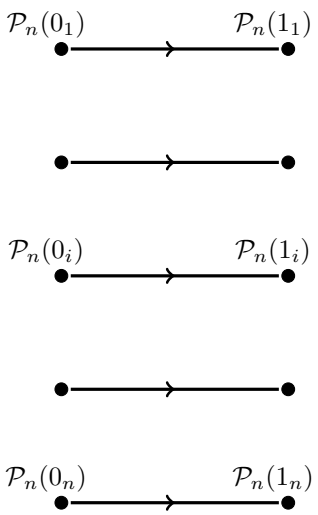


Figure 3 The construction of the path gadget for $\mathcal{P}_n$. Note that the gadget has $2n$ vertices. Each edge of $\mathcal{P}_n$ has the same weight B.

3.1.1.2 Construction of the $\mathrm{DSN}_{\mathrm{Planar}}$ instance

We give a reduction which transforms an instance $G = (V, E)$ of $\mathrm{MCSI}(K_{\ell,\ell})$ into an instance of DSN which has 2ℓ demand pairs and an optimum which is planar. Let the partition of V into color classes be given by $\{V_1, V_2, \ldots, V_\ell, W_1, W_2, \ldots, W_\ell\}$. Without loss of generality (by adding isolated vertices if necessary), we can assume that each color class has the same number of vertices. Let $|V_i| = |W_i| = n'$ for each $1 \le i \le \ell$. Then $n = |V(G)| = 2n'\ell$. For each $1 \le i, j \le \ell$ we denote by $E_{i,j}$ the set of edges with one end-point in V_i and other in W_j.

We design two types of gadgets: the *main gadget* and the *secondary gadget*. The reduction from $\mathrm{MCSI}(K_{\ell,\ell})$ represents each edge set $E_{i,j}$ with a main gadget $M_{i,j}$. This is done as follows: each main gadget is a copy of the path gadget $\mathcal{P}_{|E_{i,j}|}$ from Section 3.1.1.1 with $B = \frac{2}{\ell^2}$, i.e., there is a row in $M_{i,j}$ corresponding to each edge in $E_{i,j}$. Each main gadget

is surrounded by four secondary gadgets: on the top, right, bottom and left. Each of these gadgets are copies of the path gadget from Section 3.1.1.1 with $B = 0$:

- For each $1 \leq i \leq \ell + 1, 1 \leq j \leq \ell$ the *horizontal* gadget $HS_{i,j}$ is a copy of $\mathcal{P}_{|W_j|}$
- For each $1 \leq i \leq \ell, 1 \leq j \leq \ell + 1$ the *vertical* gadget $VS_{i,j}$ is a copy of $\mathcal{P}_{|V_i|}$

We refer to Figure 4 (bird's-eye view) and Figure 5 (zoomed-in view) for an illustration of the reduction. Fix some $1 \leq i, j \leq \ell$. The main gadget $M_{i,j}$ has four secondary gadgets surrounding it:

- Above $M_{i,j}$ is the vertical secondary gadget $VS_{i,j+1}$
- On the right of $M_{i,j}$ is the horizontal secondary gadget $HS_{i+1,j}$
- Below $M_{i,j}$ is the vertical secondary gadget $VS_{i,j}$
- On the left of $M_{i,j}$ is the horizontal secondary gadget $HS_{i,j}$

Hence, there are $\ell(\ell+1)$ horizontal secondary gadgets and $\ell(\ell+1)$ vertical secondary gadgets.

Red intra-gadget edges: Fix (i, j) such that $1 \leq i, j \leq \ell$. Recall that $M_{i,j}$ is a copy of $\mathcal{P}_{|E_{i,j}|}$ with $B = \frac{2}{\ell^2}$ and each of the secondary gadgets are copies of $\mathcal{P}_{n'}$ with $B = 0$. With slight abuse of notation, we assume that the rows of $M_{i,j}$ are indexed by the set $\{(x, y) : (x, y) \in E_{i,j}, x \in W_i, y \in V_j\}$. We add the following edges (in red color) of weight 0: for each $(x, y) \in E_{i,j}$

- Add the edge $VS_{i,j+1}(1_x) \to M_{i,j}(0_{(x,y)})$. These edges are called *top-red* edges incident on $M_{i,j}$.
- Add the edge $HS_{i,j}(1_y) \to M_{i,j}(0_{(x,y)})$. These edges are called *left-red* edges incident on $M_{i,j}$.
- Add the edge $M_{i,j}(1_{(x,y)}) \to HS_{i+1,j}(0_y)$. These edges are called *right-red* edges incident on $M_{i,j}$.
- Add the edge $M_{i,j}(1_{(x,y)}) \to VS_{i,j}(0_x)$. These edges are called *bottom-red* edges incident on $M_{i,j}$.

These are called the *intra-gadget* edges incident on $M_{i,j}$.

Introduce the following 4ℓ vertices (which we call *border* vertices):

- $a_1, a_2, \ldots, a_\ell$
- $b_1, b_2, \ldots, b_\ell$
- $c_1, c_2, \ldots, c_\ell$
- $d_1, d_2, \ldots, d_\ell$

Orange edges: For each $i \in [\ell]$ add the following edges (shown as orange in Figure 4) with weight $\frac{2\gamma^{1/5}}{4\ell}$:

- $a_i \to VS_{i,\ell+1}(0_v)$ for each $v \in V_i$. These are called *top-orange* edges.
- $VS_{i,1}(1_v) \to b_i$ for each $v \in V_i$. These are called *bottom-orange* edges.
- $c_j \to HS_{1,j}(0_w)$ for each $w \in W_j$. These are called *left-orange* edges.
- $HS_{\ell+1,j}(1_w) \to d_j$ for each $w \in W_j$. These are called *right-orange* edges.

Finally, the set of demand pairs $\mathcal{D}'$ is given by:

- Type I: the pairs (a_i, b_i) for each $1 \leq i \leq \ell$.
- Type II: the pairs (c_j, d_j) for each $1 \leq j \leq \ell$.

Clearly, the total number of demand pairs is $k = |\mathcal{D}'| = 2\ell$. Let the final graph constructed be G'. Note that G' has size $N = (n + \ell)^{O(1)}$ and can be constructed in $(n + \ell)^{O(1)}$ time. It is also easy to see that G' is actually a DAG.

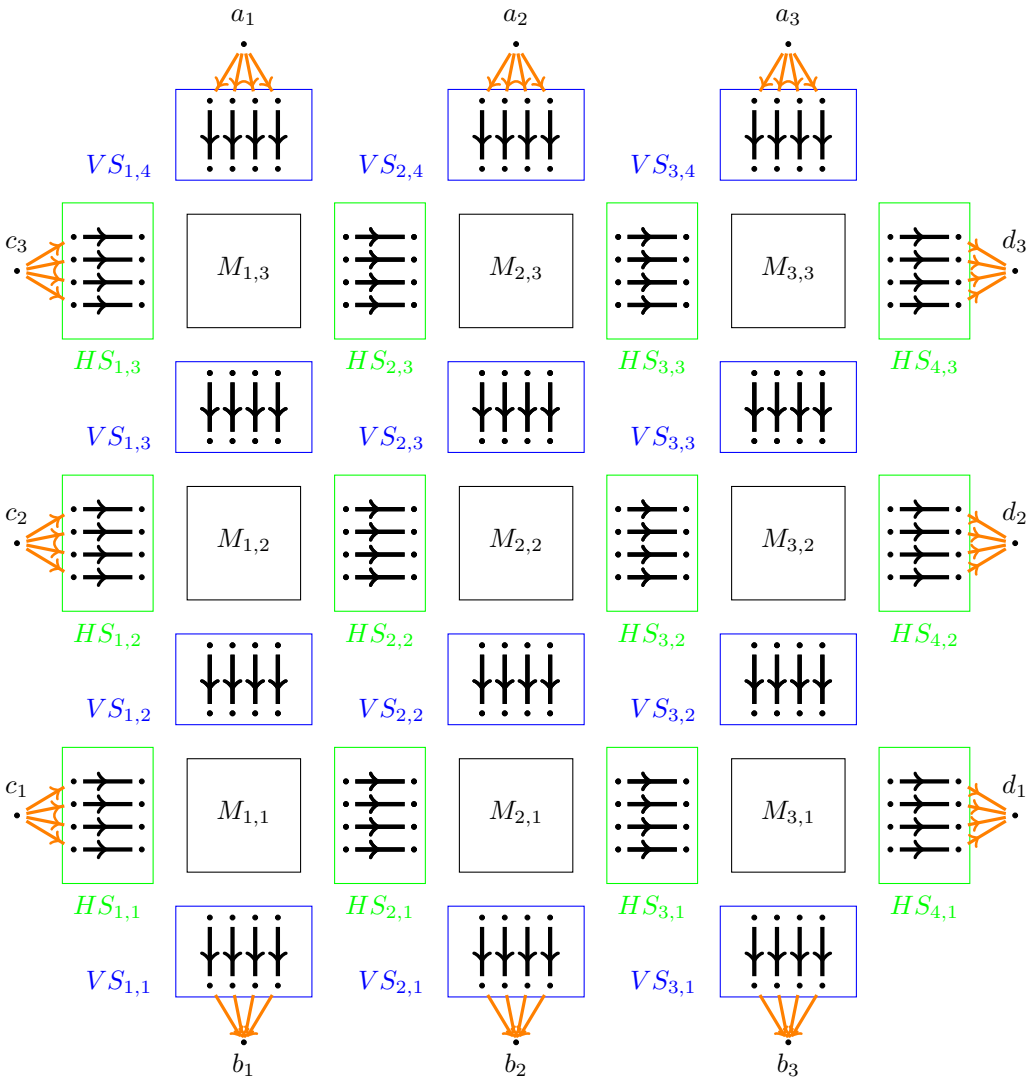


Figure 4 A bird's-eye view of the instance of G' with $\ell = 3$ and $n' = 4$ (see Figure 5 for a zoomed-in view). Additionally we have some red edges between each main gadget and the four secondary gadgets surrounding it which are omitted in this figure for clarity (they are shown in Figure 5 which gives a more zoomed-in view).

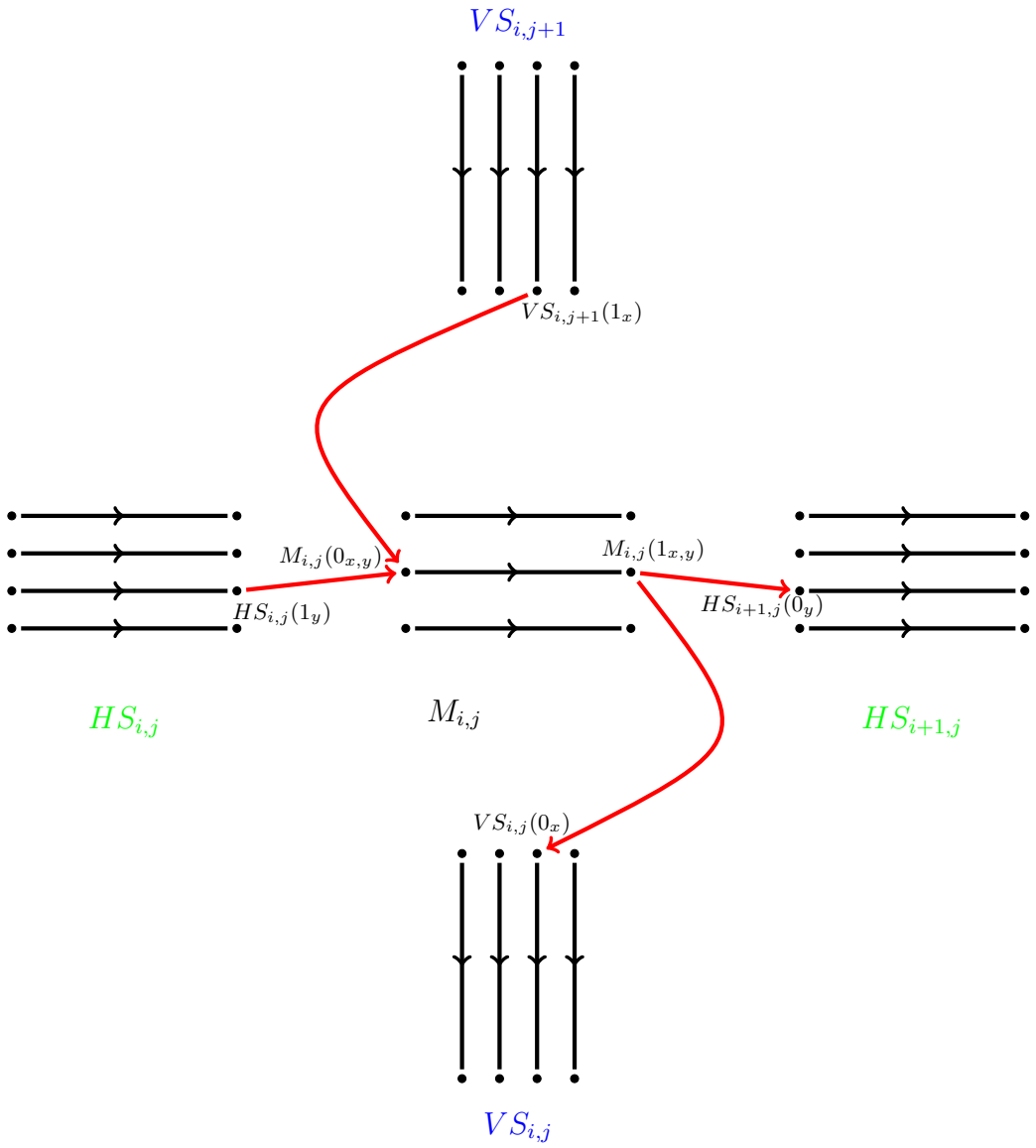


Figure 5 A zoomed-in view of the main gadget $M_{i,j}$ surrounded by four secondary gadgets: vertical gadget $VS_{i,j+1}$ on the top, horizontal gadget $HS_{i,j}$ on the left, vertical gadget $VS_{i,j}$ on the bottom and horizontal gadget $HS_{i+1,j}$ on the right. Each of the secondary gadgets is a copy of the uniqueness gadget U_n (see Section 3.1.1.1) and the main gadget $M_{i,j}$ is a copy of the uniqueness gadget $U_{|S_{i,j}|}$. The only inter-gadget edges are the red edges: they have one end-point in a main gadget and the other end-point in a secondary gadget. We have shown four such red edges which are introduced for every $(x, y) \in E_{i,j}$.

3.1.2 Finishing the proof of Theorem 3

We can now easily prove Theorem 3 by combining Lemma 22 and Corollary 10.

Proof of Theorem 3. We again prove by contrapositive. Suppose that, for some constant $\varepsilon > 0$ and for some function $f(k)$ independent of n, there exists an $f(k) \cdot N^{O(1)}$-time $(2-\varepsilon)$-approximation algorithm for $\text{DSN}_{\text{PLANAR}}$ where k is the number of terminal pairs and N is the size of the instance. Let us call this algorithm $\mathbb{A}$.

Given $\epsilon > 0$, it is easy to see that there exists a sufficiently small $\gamma^* = \gamma^*(\varepsilon)$ such that $\frac{2(2-4\gamma^{*1/5})}{2(1+\gamma^{*1/5})} \geq (2-\varepsilon)$. We create an algorithm $\mathbb{B}$ that can distinguish between the two cases of Corollary 10 with $h(\ell) = 1 - \frac{\log(1/\gamma^*)}{\log \ell} = o(1)$. Our new algorithm $\mathbb{B}$ works as follows. Given an instance $(G, H, V_1 \cup \cdots \cup V_\ell, W_1 \cup \cdots \cup W_\ell)$ of MCSI of size n where $H = K_{\ell,\ell}$, the algorithm $\mathbb{B}$ uses the reduction from Lemma 22 to create in $(n+\ell)^{O(1)}$ time a $\text{DSN}_{\text{PLANAR}}$ instance on the graph G' with $k = 2\ell$ terminal pairs and size $N = (\ell+n)^{O(1)}$. The algorithm $\mathbb{B}$ then runs $\mathbb{A}$ on this instance; if $\mathbb{A}$ returns a solution N of cost at most $2(2-4\gamma^{*1/5})$, then $\mathbb{B}$ returns YES. Otherwise, $\mathbb{B}$ returns NO.

We now show that the algorithm $\mathbb{B}$ can indeed distinguish between the YES and NO cases of Corollary 10. In the YES case, i.e., $\text{val}(\Gamma) = 1$, the completeness property of Lemma 22 guarantees that the optimal planar solution has cost at most $2(1+\gamma^{*1/5})$. Since $\mathbb{A}$ is a $(2-\varepsilon)$-approximation algorithm, it returns a solution of cost at most $2(1+\gamma^{*1/5}) \cdot (2-\varepsilon) \leq 2(2-4\gamma^{*1/5})$ where the inequality comes from our choice of γ^*; this means that $\mathbb{B}$ outputs YES. On the other hand, in the NO case, i.e., $\text{val}(\Gamma) < \gamma$, the soundness property of Lemma 22 guarantees that the optimal solution (and hence the planar optimal solution as well, if it exists) in G' has cost more than $2(2-4\gamma^{*1/5})$, which implies that $\mathbb{B}$ outputs NO.

Finally, observe that the running time of $\mathbb{B}$ is $f(k) \cdot N^{O(1)} + \text{poly}(\ell+n)^{O(1)}$ which is bounded by $f'(\ell) \cdot n^{O(1)}$ for some computable function f' since $k = 2\ell$ and $N = (n+\ell)^{O(1)}$. Hence, from Corollary 10, Gap-ETH is violated. ◀

3.2 Lower Bounds for FPT Approximation Schemes for $\text{DSN}_{\text{Planar}}$

We obtain the following result regarding the parameterized complexity of $\text{DSN}_{\text{PLANAR}}$ parameterized by $k+p$.

▶ **Theorem 4.** [⋆] *The* $\text{DSN}_{\text{PLANAR}}$ *problem is* W[1]*-hard parameterized by* $p+k$*. Moreover, under ETH, for any computable function* f *and any* $\varepsilon > 0$

- *There is no* $f(k,p) \cdot n^{o(k+\sqrt{p})}$ *time algorithm for* $\text{DSN}_{\text{PLANAR}}$*, and*
- *There is no* $f(k,\varepsilon,p) \cdot n^{o(k+\sqrt{p+1/\epsilon})}$ *time algorithm which computes a* $(1+\varepsilon)$*-approximation for* $\text{DSN}_{\text{PLANAR}}$

4 Lower Bounds for FPT Approximation Schemes for $\text{SCSS}_{\text{Planar}}$

We obtain the following result regarding the parameterized complexity of $\text{DSN}_{\text{PLANAR}}$ parameterized by $k+p$.

▶ **Theorem 5.** [⋆] *The* $\text{SCSS}_{\text{PLANAR}}$ *problem is* W[1]*-hard parameterized by* $p+k$*. Moreover, under ETH, for any computable function* f *and any* $\varepsilon > 0$

- *there is no* $f(k,p) \cdot n^{o(\sqrt{k+p})}$ *time algorithm for* $\text{SCSS}_{\text{PLANAR}}$*, and*
- *there is no* $f(k,\varepsilon,p) \cdot n^{o(\sqrt{k+p+1/\epsilon})}$ *time algorithm which computes an* $(1+\varepsilon)$*-approximation for* $\text{SCSS}_{\text{PLANAR}}$*.*

References

1 Amit Agarwal, Noga Alon, and Moses Charikar. Improved approximation for directed cut problems. In *STOC*, pages 671–680, 2007. doi:10.1145/1250790.1250888.

2 Benny Applebaum. Exponentially-Hard Gap-CSP and Local PRG via Local Hardcore Functions. In *FOCS 2017*, pages 836–847, 2017. doi:10.1109/FOCS.2017.82.

3 Sanjeev Arora, László Babai, Jacques Stern, and Z. Sweedyk. The Hardness of Approximate Optima in Lattices, Codes, and Systems of Linear Equations. *J. Comput. Syst. Sci.*, 54(2):317–331, 1997. doi:10.1006/jcss.1997.1472.

4 Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Approximation algorithms for spanner problems and Directed Steiner Forest. *Information and Computation*, 222:93–107, 2013.

5 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 *FOCS*, pages 743–754, 2017. doi:10.1109/FOCS.2017.74.

6 Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation Algorithms for Directed Steiner Problems. *J. Algorithms*, 33(1):73–91, 1999. doi:10.1006/jagm.1999.1042.

7 Chandra Chekuri, Guy Even, Anupam Gupta, and Danny Segev. Set connectivity problems in undirected graphs and the directed Steiner network problem. *ACM Transactions on Algorithms*, 7(2):18, 2011.

8 Chandra Chekuri and Vivek Madan. Approximating Multicut and the Demand Graph. In *SODA*, pages 855–874, 2017. doi:10.1137/1.9781611974782.54.

9 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *J. Comput. Syst. Sci.*, 72(8):1346–1367, 2006.

10 Rajesh Chitnis and Andreas Emil Feldmann and. FPT Inapproximability of Directed Cut and Connectivity Problems. *CoRR*, abs/1910.01934, 2019. arXiv:1910.01934.

11 Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized Approximation Algorithms for Bidirected Steiner Network Problems. In *ESA*, pages 20:1–20:16, 2018. doi:10.4230/LIPIcs.ESA.2018.20.

12 Rajesh Hemant Chitnis, Marek Cygan, Mohammad Taghi Hajiaghayi, and Dániel Marx. Directed Subset Feedback Vertex Set Is Fixed-Parameter Tractable. *ACM Trans. Algorithms*, 11(4):28:1–28:28, 2015. doi:10.1145/2700209.

13 Rajesh Hemant Chitnis, MohammadTaghi Hajiaghayi, and Guy Kortsarz. Fixed-Parameter and Approximation Algorithms: A New Look. In *IPEC 2013*, pages 110–122, 2013. doi:10.1007/978-3-319-03898-8_11.

14 Rajesh Hemant Chitnis, MohammadTaghi Hajiaghayi, and Dániel Marx. Fixed-Parameter Tractability of Directed Multiway Cut Parameterized by the Size of the Cutset. *SIAM J. Comput.*, 42(4):1674–1696, 2013. doi:10.1137/12086217X.

15 Rajesh Hemant Chitnis, MohammadTaghi Hajiaghayi, and Dániel Marx. Tight Bounds for Planar Strongly Connected Steiner Subgraph with Fixed Number of Terminals (and Extensions). In *SODA*, pages 1782–1801, 2014. doi:10.1137/1.9781611973402.129.

16 Julia Chuzhoy and Sanjeev Khanna. Polynomial flow-cut gaps and hardness of directed cut problems. *J. ACM*, 56(2):6:1–6:28, 2009. doi:10.1145/1502793.1502795.

17 Irit Dinur. Mildly exponential reduction from gap 3SAT to polynomial-gap label-cover. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:128, 2016.

18 Irit Dinur and Pasin Manurangsi. ETH-Hardness of Approximating 2-CSPs and Directed Steiner Network. In *ITCS*, pages 36:1–36:20, 2018. doi:10.4230/LIPIcs.ITCS.2018.36.

19 Yevgeniy Dodis and Sanjeev Khanna. Design networks with bounded pairwise distance. In *STOC 1999*, pages 750–759, 1999.

20 Jon Feldman and Matthias Ruhl. The Directed Steiner Network Problem is Tractable for a Constant Number of Terminals. *SIAM J. Comput.*, 36(2):543–561, 2006. doi:10.1137/S0097539704441241.

21 Moran Feldman, Guy Kortsarz, and Zeev Nutov. Improved approximation algorithms for Directed Steiner Forest. *J. Comput. Syst. Sci.*, 78(1):279–292, 2012. doi:10.1016/j.jcss.2011.05.009.

22 Andreas Emil Feldmann and Dániel Marx. The Complexity Landscape of Fixed-Parameter Directed Steiner Network Problems. *CoRR*, abs/1707.06808, 2017. arXiv:1707.06808.

23 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Multiway Cuts in Directed and Node Weighted Graphs. In *ICALP*, pages 487–498, 1994. doi:10.1007/3-540-58201-0_92.

24 Jiong Guo, Rolf Niedermeier, and Ondrej Suchý. Parameterized Complexity of Arc-Weighted Directed Steiner Problems. *SIAM J. Discrete Math.*, 25(2):583–599, 2011.

25 Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. *STOC '03*, pages 585–594, 2003. doi:10.1145/780542.780628.

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

27 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

28 Subhash Khot. On the power of unique 2-prover 1-round games. In *STOC*, pages 767–775, 2002. doi:10.1145/509907.510017.

29 Guy Kortsarz and David Peleg. On Choosing a Dense Subgraph (Extended Abstract). In *FOCS 1993*, pages 692–701, 1993.

30 Euiwoong Lee. Improved Hardness for Cut, Interdiction, and Firefighter Problems. In *ICALP*, pages 92:1–92:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.92.

31 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/92.

32 Pasin Manurangsi and Prasad Raghavendra. A Birthday Repetition Theorem and Complexity of Approximating Dense CSPs. In *ICALP*, pages 78:1–78:15, 2017.

33 Dániel Marx. Can You Beat Treewidth? *Theory of Computing*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.

34 Dániel Marx and Igor Razgon. Fixed-Parameter Tractability of Multicut Parameterized by the Size of the Cutset. *SIAM J. Comput.*, 43(2):355–388, 2014. doi:10.1137/110855247.

35 Joseph Naor and Leonid Zosin. A 2-Approximation Algorithm for the Directed Multiway Cut Problem. *SIAM J. Comput.*, 31(2):477–482, 2001. doi:10.1137/S009753979732147X.

36 Marcin Pilipczuk and Magnus Wahlström. Directed Multicut is W[1]-hard, Even for Four Terminal Pairs. *TOCT*, 10(3):13:1–13:18, 2018. doi:10.1145/3201775.

C-Planarity Testing of Embedded Clustered Graphs with Bounded Dual Carving-Width

Giordano Da Lozzo
Roma Tre University, Rome, Italy
giordano.dalozzo@uniroma3.it

David Eppstein
University of California, Irvine, USA
eppstein@uci.edu

Michael T. Goodrich
University of California, Irvine, USA
goodrich@uci.edu

Siddharth Gupta
Ben-Gurion University of the Negev, Beersheba, Israel
siddhart@post.bgu.ac.il

Abstract

For a *clustered graph*, i.e, a graph whose vertex set is recursively partitioned into clusters, the C-Planarity Testing problem asks whether it is possible to find a planar embedding of the graph and a representation of each cluster as a region homeomorphic to a closed disk such that **1.** the subgraph induced by each cluster is drawn in the interior of the corresponding disk, **2.** each edge intersects any disk at most once, and **3.** the nesting between clusters is reflected by the representation, i.e., child clusters are properly contained in their parent cluster. The computational complexity of this problem, whose study has been central to the theory of graph visualization since its introduction in 1995 [Feng, Cohen, and Eades, *Planarity for clustered graphs*, ESA'95], has only been recently settled [Fulek and Tóth, *Atomic Embeddability, Clustered Planarity, and Thickenability*, to appear at SODA'20]. Before such a breakthrough, the complexity question was still unsolved even when the graph has a prescribed planar embedding, i.e, for *embedded clustered graphs*.

We show that the C-Planarity Testing problem admits a single-exponential single-parameter FPT algorithm for embedded clustered graphs, when parameterized by the carving-width of the dual graph of the input. This is the first FPT algorithm for this long-standing open problem with respect to a single notable graph-width parameter. Moreover, in the general case, the polynomial dependency of our FPT algorithm is smaller than the one of the algorithm by Fulek and Tóth. To further strengthen the relevance of this result, we show that the C-Planarity Testing problem retains its computational complexity when parameterized by several other graph-width parameters, which may potentially lead to faster algorithms.

2012 ACM Subject Classification Human-centered computing → Graph drawings; Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph theory

Keywords and phrases Clustered planarity, carving-width, non-crossing partitions, FPT

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.9

Related Version A full version of the paper [47] is available at `https://arxiv.org/abs/1910.02057`.

Funding *Giordano Da Lozzo*: Supported in part by H2020-MSCA-RISE project 734922 – "CONNECT", by MIUR Project "AHeAD" under PRIN 20174LF3T8, by MIUR Project "MODE" under PRIN 20157EFM5C, and by MIUR-DAAD JMP N° 34120.
David Eppstein: Supported in part by the US NSF under grants CCF-1618301 and CCF-1616248.
Michael T. Goodrich: Supported in part by the US NSF under grant 1815073.
Siddharth Gupta: Supported in part by the Zuckerman STEM Leadership Program and by the Frankel Foundation.

1 Introduction

Many real-word data exhibit an intrinsic hierarchical structure that can be captured in the form of clustered graphs, i.e., graphs equipped with a recursive clustering of their vertices. This graph model has proved very powerful to represent information at different levels of abstraction and drawings of clustered networks appear in a wide variety of application domains, such as software visualization, knowledge representation, visual statistics, and data mining. More formally, a *clustered graph* (for short, *c-graph*) is a pair $\mathcal{C}(G, \mathcal{T})$, where G is the *underlying graph* and $\mathcal{T}$ is the *inclusion tree* of $\mathcal{C}$, i.e., a rooted tree whose leaves are the vertices of G. Each non-leaf node μ of $\mathcal{T}$ corresponds to a cluster containing the subset V_μ of the vertices of G that are the leaves of the subtree of $\mathcal{T}$ rooted at μ. Edges between vertices of the same cluster (resp., of different clusters) are *intra-cluster edges* (resp., *inter-cluster edges*).

A natural and well-established criterion for a readable visualization of a c-graph has been derived from the classical notion of graph planarity. A *c-planar drawing* of a c-graph $\mathcal{C}(G, \mathcal{T})$ (see Fig. 4c) is a planar drawing of G together with a representation of each cluster μ in $\mathcal{T}$ as a region $D(\mu)$ homeomorphic to a closed disc such that: (i) for each cluster μ in $\mathcal{T}$, region $D(\mu)$ contains the drawing of the subgraph $G[V_\mu]$ of G induced by V_μ; (ii) for every two clusters μ and η in $\mathcal{T}$, it holds $D(\eta) \subseteq D(\mu)$ if and only if η is a descendant of μ in $\mathcal{T}$; (iii) each edge crosses the boundary of any cluster disk at most once; and (iv) the boundaries of no two cluster disks intersect. A c-graph is *c-planar* if it admits a c-planar drawing.

The C-Planarity Testing problem, introduced by Feng, Cohen, and Eades more than two decades ago [31], asks for the existence of a c-planar drawing of a c-graph. Despite several algorithms having been presented in the literature to construct c-planar drawings of c-planar c-graphs with nice aesthetic features [9, 28, 42, 48], determining the computational complexity of the C-Planarity Testing problem has been one of the most challenging quests in the graph drawing research area [16, 21, 53]. To shed light on the complexity of the problem, several researchers have tried to highlight its connections with other notoriously difficult problems in the area [3, 53], as well as to consider relaxations [6, 7, 10, 30, 25, 56] and more constrained versions [2, 4, 5, 8, 23, 32, 33] of the classical notion of c-planarity. Algebraic approaches have also been considered [34, 41]. Only recently, Fulek and Tóth settled the question by giving a polynomial-time algorithm for a generalization of the C-Planarity Testing problem called Atomic Embeddability [36].

A cluster μ is *connected* if $G[V_\mu]$ is connected, and it is *disconnected* otherwise. A c-graph is *c-connected* if every cluster is connected. Efficient algorithms for the c-connected case have been known since the early stages of the research on the problem [22, 27, 31]. Afterwards, polynomial-time algorithms have also been conceived for c-graphs satisfying other, weaker, connectivity requirements [20, 38, 40]. A c-graph is *flat* if each leaf-to-root path in $\mathcal{T}$ consists of exactly two edges, that is, the clustering determines a partition of the vertex set; see, e.g., Fig. 4a. For flat c-graphs polynomial-time algorithms are known for several restricted cases [2, 11, 13, 17, 29, 35, 33, 43, 45, 46].

Motivations and contributions. In this paper, we consider the parameterized complexity of the C-Planarity Testing problem for *embedded c-graphs*, i.e., c-graphs with a prescribed combinatorial embedding; see also [13, 18, 26, 46] for previous work in this direction.

In Section 2, we show that the C-Planarity Testing problem retains its complexity when restricted to instances of bounded path-width and to connected instances of bounded tree-width. Such a result implies that the goal of devising an algorithm parameterized by graph-width parameters that are within a constant factor from tree-width (e.g., branch-

width [50]) or that are bounded by path-width (e.g., tree-width, rank-width [49], boolean-width [1], and clique-width [19]) and with a dependency on the input size which improves upon the one in [36] has to be regarded as a major algorithmic challenge.

Remarkably, before the results presented in [36], the computational complexity of the problem was still unsolved even for instances with faces of bounded size, and polynomial-time algorithms were known only for "small" faces and in the flat scenario. Namely, Jelinkova et al. [46] presented a quadratic-time algorithm for 3-connected flat c-graphs with faces of size at most 4. Subsequently, Di Battista and Frati [29] presented a linear-time/linear-space algorithm for embedded flat c-graphs with faces of size at most 5.

Motivated by the discussion above and by the results in Section 2, we focus our attention on embedded c-graphs $\mathcal{C}(G, \mathcal{T})$ whose underlying graph G has *both* bounded tree-width and bounded face size, i.e., instances such that the *carving-width* of the dual $\delta(G)$ of G is bounded. In Section 3, we present an FPT algorithm based on a dynamic-programming approach on a bond-carving decomposition to solve the problem for embedded flat c-graphs, which is ultimately based on maintaining a succinct description of the internal cluster connectivity via non-crossing partitions. We remark that, to the best of the authors' knowledge, this is the first FPT algorithm for the C-Planarity Testing problem, with respect to a *single* graph-width parameter. More formally, we prove the following.

▶ **Theorem 1.** C-Planarity Testing *can be solved in* $O(2^{4\omega+\log\omega}n + n^2)$ *time for any* n*-vertex embedded flat c-graph* $\mathcal{C}(G, \mathcal{T})$*, where* ω *is the carving-width of* $\delta(G)$*, if a carving decomposition of* $\delta(G)$ *of width* ω *is provided, and in* $O(2^{4\omega+\log\omega}n + n^3)$ *time, otherwise.*

It is well know that the carving-width $\mathrm{cw}(\delta(H))$ of the dual graph $\delta(H)$ of a plane graph H with maximum face size $\ell(H)$ and tree-width $\mathrm{tw}(H)$ satisfies the relationship $\mathrm{cw}(\delta(H)) \leq \ell(H)(\mathrm{tw}(H) + 2)$ [12, 15]. Therefore, Theorem 1 provides the first[1] polynomial-time algorithm for instances of bounded face size and bounded tree-width, which answers an open question posed by Di Battista and Frati [29, Open Problem (ii)] for instances of bounded tree-width; also, since any n-vertex planar graph has tree-width in $O(\sqrt{n})$, it provides an $2^{O(\sqrt{n})}$ subexponential-time algorithm for instances of bounded face size, which improves the previous $2^{O(\sqrt{n}\log n)}$ time bound presented in [26] for such instances.

Further implications of Theorem 1 for instances of bounded embedded-width and of bounded dual cut-width are discussed in Section 4. Moreover, we extend Theorem 1 to get an FPT algorithm for general non-flat embedded c-graphs, whose running time is $O(4^{4\omega+\log\omega}n + n^2)$ if a carving decomposition of $\delta(G)$ of width ω is provided, and is $O(4^{4\omega+\log\omega}n + n^3)$, otherwise. The details for such an extension can be found in the full version [47].

2 Definitions and Preliminaries

In this section, we give definitions and preliminaries that will be useful throughout.

Graphs and connectivity. A *graph* $G = (V, E)$ is a pair, where V is the set of *vertices* of G and E is the set of *edges* of G ,i.e., pairs of vertices in V. A *multigraph* is a generalization of a graph that allows the existence of multiple copies of the same edge. The *degree* of a vertex is the number of its incident edges. We denote the *maximum degree* of G by $\Delta(G)$. Also, for any $S \subseteq V$, we denote by $G[S]$ the subgraph of G induced by the vertices in S.

[1] The results in this paper were accepted for publication before the contribution of Fulek and Tóth in [36].

A graph is *connected* if it contains a path between any two vertices. A *cutvertex* is a vertex whose removal disconnects the graph. A connected graph containing no cutvertices is 2-*connected*. The *blocks* of a graph are its maximal 2-connected subgraphs. In this paper, we only deal with connected graphs, unless stated otherwise.

Planar graphs and embeddings. A drawing of a graph is *planar* if it contains no edge crossings. A graph is *planar* if it admits a planar drawing. Two planar drawings of the same graph are *equivalent* if they determine the same *rotation* at each vertex, i.e, the same circular orderings for the edges around each vertex. A *combinatorial embedding* (for short, *embedding*) is an equivalence class of planar drawings. A planar drawing partitions the plane into topologically connected regions, called *faces*. The bounded faces are the *inner faces*, while the unbounded face is the *outer face*. A combinatorial embedding together with a choice for the outer face defines a *planar embedding*. An *embedded graph* (resp. *plane graph*) G is a planar graph with a fixed combinatorial embedding (resp. fixed planar embedding). The *length* of a face f of G is the number of occurrences of the edges of G encountered in a traversal of the boundary of f. The *maximum face size* $\ell(G)$ of G is the maximum length over all faces of G.

C-Planarity. An *embedded c-graph* $\mathcal{C}(G, \mathcal{T})$ is a c-graph whose underlying graph G has a prescribed combinatorial embedding, and it is *c-planar* if it admits a c-planar drawing that preserves the given embedding. Since we only deal with embedded c-graphs, in the remainder of the paper we will refer to them simply as c-graphs. Also, when G and $\mathcal{T}$ are clear from the context, we simply denote $\mathcal{C}(G, \mathcal{T})$ as $\mathcal{C}$. A *candidate saturating edge* of $\mathcal{C}$ is an edge not in G between two vertices of the same cluster in $\mathcal{T}$ that are incident to the same face of G; refer to Fig. 4b. A c-graph $\mathcal{C}'(G', \mathcal{T}')$ with $\mathcal{T}' = \mathcal{T}$ obtained by adding to $\mathcal{C}$ a subset E^+ of its candidate saturating edges is a *super c-graph* of $\mathcal{C}$; also, set E^+ is a *planar saturation* if G' is planar. Further, c-graph $\mathcal{C}$ is *hole-free* if there exists a face f in G such that when f is chosen as the outer face for G no cycle composed of vertices of the same cluster encloses a vertex of a different cluster in its interior. Finally, two c-graphs are *equivalent* if and only if they are both c-planar or they are both not c-planar.

▶ Remark 2. In this paper, we only consider c-graphs whose underlying graph is connected, unless stated otherwise.

We will exploit the following characterization presented by Di Battista and Frati [29], which holds true also for non-flat c-graphs although originally only proved for flat c-graphs.

▶ **Theorem 3** ([29], Theorem 1)**.** *A c-graph $\mathcal{C}(G, \mathcal{T})$ is c-planar if and only if:*

- **(i)** *G is planar,*
- **(ii)** *$\mathcal{C}$ is hole-free, and*
- **(iii)** *there exists a super c-graph $\mathcal{C}^*(G^*, \mathcal{T}^*)$ of $\mathcal{C}$ such that G^* is planar and $\mathcal{C}^*$ is c-connected.*

Condition i of Theorem 3 can be tested using any of the known linear-time planarity-testing algorithms. Condition ii of Theorem 3 can be verified in linear time as described by Di Battista and Frati [29, Lemma 7], by exploiting the linear-time algorithm for checking if an embedded, possibly non-flat, c-graph is hole-free presented by Dahlhaus [27]. Therefore, in the following we will assume that any c-graph satisfies these conditions and thus view the C-Planarity Testing problem as one of testing Condition iii.

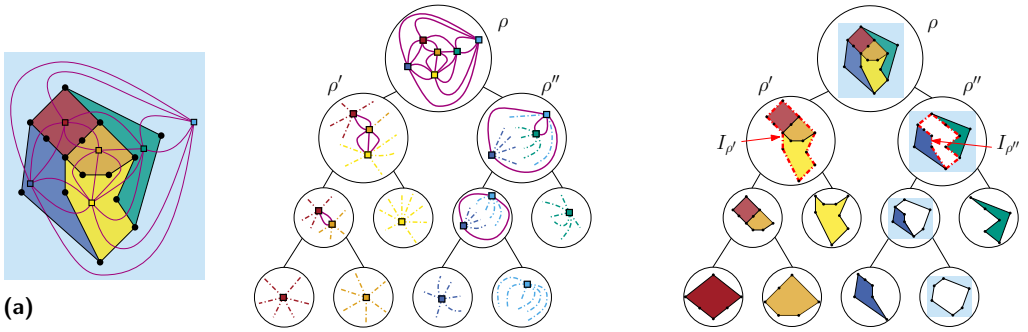


Figure 1 (a) Running example: An embedded graph G and its dual $\delta(G)$. (b) A bond-carving decomposition (D, γ) of the dual $\delta(G)$ of the graph G in Fig. 1a. (c) The decomposition (D, γ) where the vertices of $\delta(G)$ are replaced by the corresponding faces of G.

Tree-width. A *tree decomposition* of a graph G is a tree T whose nodes, called *bags*, are labeled by subsets of vertices of G. For each vertex v the bags containing v must form a nonempty contiguous subtree of T, and for each edge (u, v) of G at least one bag of T must contain both u and v. The *width* of the decomposition is one less than the maximum cardinality of any bag. The *tree-width* $\mathrm{tw}(G)$ of G is the minimum width of any of its tree decompositions.

Cut-sets and duality. Let $G = (V, E)$ be a connected graph and let S be a subset of V. The partition $\{S, V \setminus S\}$ of V is a *cut* of G and the set $(S, V \setminus S)$ of edges with an endpoint in S and an endpoint in $V \setminus S$ is a *cut-set* of G. Also, cut-set $(S, V \setminus S)$ is a *bond* if $G[S]$ and $G[V \setminus S]$ are both non-null and connected.

For an embedded graph, the *dual* $\delta(G)$ of G is the planar multigraph that has a vertex v_f, for each face f of G, and an edge (v_f, v_g), for each edge e shared by faces f and g. The edge e is the *dual edge* of (v_f, v_g), and vice versa. Also, $\delta(G)$ is 2-connected if and only if G is 2-connected. Fig. 1a shows a plane graph G (black edges) and its dual $\delta(G)$ (purple edges); we will use these graphs as running examples throughout the paper. The following duality is well known.

▶ **Lemma 4** ([37], Theorem 14.3.1)**.** *If G is an embedded graph, then a set of edges is a cycle of G if and only if their dual edges form a bond in $\delta(G)$.*

Carving-width. A *carving decomposition* of a graph $G = (V, E)$ is a pair (D, γ), where D is a rooted binary tree whose leaves are the vertices of G, and γ is a function that maps the non-root nodes of D, called *bags*, to cut-sets of G as follows. For any non-root bag ν, let D_ν be the subtree of D rooted at ν and let $\mathcal{L}_\nu$ be the set of leaves of D_ν. Then, $\gamma(\nu) = (\mathcal{L}_\nu, V \setminus \mathcal{L}_\nu)$. The *width* of a carving decomposition (D, γ) is the maximum of $|\gamma(\nu)|$ over all bags ν in D. The *carving-width* $\mathrm{cw}(G)$ of G is the minimum width over all carving decompositions of G. The *dual carving-width* is the carving-width of the dual of G. A *bond-carving decomposition* is a special kind of carving decomposition in which each cut-set is a bond of the graph; i.e., in a bond-carving decomposition every cut-set separates the graph into two connected components [51, 54].

In this paper, we view a bond-carving decomposition of the vertices of the dual $\delta(G)$ of an embedded graph G as a decomposition of the faces of G; see Fig. 1c. A similar approach was followed in [12]. In particular, due to the duality expressed by Lemma 4, the cut-sets $\gamma(\nu)$ of

the bags ν of D correspond to cycles that can be used to *recursively partition* the faces of the primal graph, where these cycles are formed by the edges of the primal that are dual to those in each cut-set.

Partitions. Let $\mathcal{Q} = \{q_1, q_2, \ldots, q_n\}$ be a ground set. A *partition* of $\mathcal{Q}$ is a set $\{Q_1, \ldots, Q_k\}$ of non-empty subsets Q_i's of $\mathcal{Q}$, called *parts*, such that $\mathcal{Q} = \bigcup_{i=1}^{k} Q_i$ and $Q_i \cap Q_j = \emptyset$, with $1 \leq i < j \leq k$. Observe that $k \leq |\mathcal{Q}|$. Let now $\mathcal{S} = (s_1, s_2, \ldots, s_n)$ be a *cyclically-ordered set*, i.e., a set equipped with a circular ordering. Let a, b, and c be three elements of $\mathcal{S}$ such that b appears after a and before c in the circular ordering of $\mathcal{S}$; we write $a \prec_b c$. A partition P of $\mathcal{S}$ is *crossing*, if there exist elements $a, c \in S_i$ and $b, d \in S_j$, with $S_i, S_j \in P$ and $i \neq j$, such that $a \prec_b c$ and $c \prec_d a$; and, it is *non-crossing*, otherwise. We denote the set of all the non-crossing partitions of $\mathcal{S}$ by $\mathcal{NC}(\mathcal{S})$. Note that, $|\mathcal{NC}(\mathcal{S})|$ coincides with the *Catalan number* $\mathcal{CAT}(n)$ of n, which satisfies $\mathcal{CAT}(n) \leq 2^{2n}$.

2.1 Relationship between Graph-Width Parameters and Connectivity

In this section, we present reductions that shed light on the effect that the interplay between some notable graph-width parameters and the connectivity of the underlying graph have on the computational complexity of the C-Planarity Testing problem.

We will exploit recent results by Cortese and Patrignani, who proved the following:

(a) Any n-vertex non-flat c-graph $\mathcal{C}(G, \mathcal{T})$ can be transformed into an equivalent $O(n \cdot h)$-vertex flat c-graph in quadratic time [24, Theorem 1], where h is the height of $\mathcal{T}$.

(b) Any n-vertex flat c-graph can be turned into an equivalent $O(n)$-vertex *independent flat c-graph*, i.e., a flat c-graph such that each non-root cluster induces an independent set, in linear time [24, Theorem 2].

We remark that the reductions from [24] preserve the connectivity of the underlying graph.

▶ **Theorem 5.** *Let $\mathcal{C}(G, \mathcal{T})$ be an n-vertex (flat) c-graph and let h be the height of $\mathcal{T}$. In $O(n^2)$ time (in $O(n)$ time), it is possible to construct an $O(n \cdot h)$-vertex ($O(n)$-vertex) independent flat c-graph $\mathcal{C}'(G', \mathcal{T}')$ that is equivalent to $\mathcal{C}$ such that:*

(i) *G' is a collection of stars or*

(ii) *G' is a tree.*

Sketch. Let $\mathcal{C}(G, \mathcal{T})$ be an n-vertex (flat) c-graph. By the results a and b above, we can construct an $O(n \cdot h)$-vertex ($O(n)$-vertex) independent flat c-graph $\mathcal{C}^+(G^+, \mathcal{T}^+)$ equivalent to $\mathcal{C}$ in $O(n^2)$ time (in $O(n)$ time). Note that, G^+ only contains inter-cluster edges.

Let $e = (u, v)$ be an edge of G^+. Consider a c-graph $\mathcal{C}^1(G^1, \mathcal{T}^1)$ obtained from $\mathcal{C}^+$ as follows. First, subdivide the edge e with two dummy vertices u_e and v_e to create edges (u, u_e), (u_e, v_e), and (v_e, v). Then, delete the edge (u_e, v_e). Note that, the rotation scheme at u and v is same in G' as in G, considering $u = v_e$ and $v = u_e$ in the cyclic ordering of the neighbors of v and of u respectively in G. Finally, assign u_e and v_e to a new cluster μ_e, and add μ_e as a child of the root of the tree $\mathcal{T}^+$. To construct $\mathcal{C}'$ in case (i), we perform the above transformation for all the edges of G^+. To construct $\mathcal{C}'$ in case (ii), as long as the graph contains a cycle, we perform the above transformation for an edge e of such a cycle. Since the construction of C' from C^+ can be done in linear time both in case (i) and (ii), the running time follows. The equivalence can be proved by performing the above transformation, and its reverse, in c-connected super c-graphs of $\mathcal{C}^+$ and of $\mathcal{C}^1$, respectively. ◀

We point out that by applying the reduction in the above proof without enforcing a specific embedding, Theorem 5 also holds for general instances of the C-Planarity Testing problem, i.e., non-embedded c-graphs. Moreover, since the reduction given in [24]

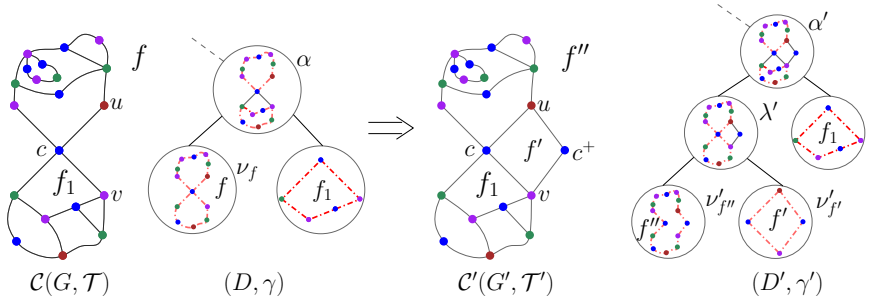


Figure 2 Reduction of Lemma 6 focused on cutvertex c. The transformation of (D, γ) into (D', γ') is shown. The red dashed edges are dual to those in the cut-set of each bag.

also works for disconnected instances, applying the reduction of Theorem 5 for case (i) to a general disconnected instance $\mathcal{C}(G, \mathcal{T})$ would result in an equivalent independent flat c-graph $\mathcal{C}'(G', \mathcal{T}')$ such that G' is a collection of stars. An immediate, yet important, consequence of this discussion is that an algorithm with running time in $O(r(n))$ for flat instances whose underlying graph is a collection of stars would result in an algorithm with running time in $O(r(n))$ for flat instances and in $O(r(n^2) + n^2)$ for general, possibly non-flat, instances.

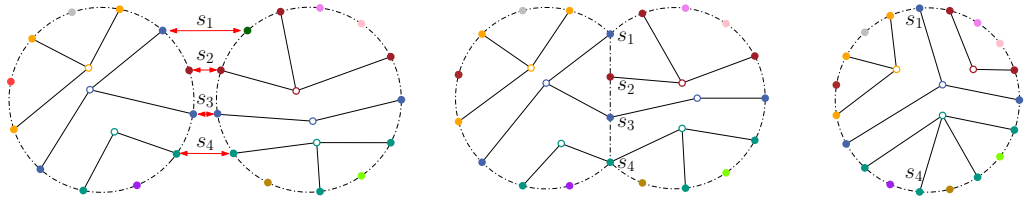
The proof of the following lemma, which will turn useful in the following sections, is based on the duality expressed by Lemma 4.

▶ **Lemma 6.** *Given an n-vertex c-graph $\mathcal{C}(G, \mathcal{T})$ and a carving decomposition (D, γ) of $\delta(G)$ of width ω, in $O(n)$ time, it is possible to construct an $O(n)$-vertex c-graph $\mathcal{C}'(G', \mathcal{T}')$ that is equivalent to $\mathcal{C}$ such that G' is 2-connected, and a carving decomposition (D', γ') of $\delta(G')$ of width $\omega' = \max(\omega, 4)$.*

Proof. We construct $\mathcal{C}'(G', \mathcal{T}')$ as follows. Let $\beta(G) < n$ be the number of blocks of G.

Let c be a cutvertex of G, let μ be the cluster that is the parent of c in $\mathcal{T}$, and let (u, c) and (v, c) be two edges belonging to different blocks of G that are incident to the same face f of G. Consider the c-graph $\mathcal{C}^+(G^+, \mathcal{T}^+)$ obtained from $\mathcal{C}$ by embedding a path (u, c^+, v) inside f, where c^+ is a new vertex that we add as a child of μ; see Fig. 2. We denote by f' the face of G^+ bounded by the cycle (u, c^+, v, c) and by f'' the other face of G^+ incident to c^+. Clearly, this augmentation can be done in $O(1)$ time and G^+ contains $\beta(G) - 1$ blocks. Also, $\mathcal{C}$ and $\mathcal{C}^+$ are equivalent. This is due to the fact that any saturating edge (c, x) incident to c and lying in f (of a c-connected c-planar super c-graph of $\mathcal{C}$) can be replaced by two saturating edges (x, c^+) lying in f'' and (c^+, c) lying in f' (of a c-connected c-planar super c-graph of $\mathcal{C}^+$), and vice versa.

We now show how to modify the carving decomposition (D, γ) of $\delta(G)$ of width ω into a carving decomposition (D^+, γ^+) of $\delta(G^+)$ of width $\omega^+ = \max(\omega, 4)$ in $O(1)$ time. Consider the leaf bag ν_f of D corresponding to face f and let α be the parent of ν_f in D. We construct D^+ from D as follows. First, we initialize $(D^+, \gamma^+) = (D, \gamma)$; in the following, we denote by ν' the bag of D^+ corresponding to the bag ν of D. We remove ν'_f from D^+, add a new non-leaf bag λ' as a child of α' and two leaf bags $\nu'_{f'}$, corresponding to face f', and $\nu'_{f''}$, corresponding to face f'', as children of λ'; refer to Fig. 2. Further, we have $\gamma^+(\nu'_{f'}) = \{(u, c), (u, c^+), (v, c), (v, c^+)\}$, $\gamma^+(\nu'_{f''}) = \gamma(\nu'_f) \setminus \{(u, c), (v, c)\} \cup \{(u, c^+), (v, c^+)\}$, $\gamma^+(\lambda') = \gamma(\nu'_f)$, and $\gamma^+(\nu') = \gamma(\nu)$, for any other bag ν belonging to both D^+ and D. In particular, the size of the edge-cuts defined by all the bags different from $\nu'_{f'}$ stays the same, while the size of the edge-cut of $\nu'_{f'}$ is 4. Therefore, (D^+, γ^+) is a carving decomposition of $\delta(G^+)$ of width $\omega^+ = \max(\omega, 4)$.



(a) $G(P')$ and $G(P'')$ **(b)** H **(c)** $G(P^*)$

Figure 3 Illustrations for the definition of bubble merge.

Repeating the above procedure, eventually yields a 2-connected c-graph $\mathcal{C}'(G', \mathcal{T}')$, with $|V(G')| = n + \beta(G) - 1 = O(n)$, that is equivalent to $\mathcal{C}$ and a carving decomposition (D', γ') of $\delta(G')$ of width $\omega' = \max(\omega, 4)$. Since each execution of the above procedure takes $O(1)$ time and since the cutvertices and the blocks of G can be computed in $O(n)$ time [44], we have that $\mathcal{C}'$ and (D', γ') can be constructed in $O(n)$ time. This concludes the proof. ◀

3 A Dynamic-Programming Algorithm for Flat Instances

In this section, we present an FPT algorithm for the C-Planarity Testing problem of flat c-graphs parameterized by the dual carving-width. We first describe a dynamic-programming algorithm to test whether a 2-connected flat c-graph $\mathcal{C}$ is c-planar, by verifying whether $\mathcal{C}$ satisfies Condition iii of Theorem 3. Then, by combining this result and Lemma 6, we extend the algorithm to simply-connected instances.

Basic operations. Let $\mathcal{C}(G, \mathcal{T})$ be a flat c-graph. A partition $\{S_1, \ldots, S_k\}$ of $V' \subseteq V(G)$ is *good* if, for each part S_i, there exists a non-root cluster μ such that all the vertices in S_i belong to μ; also, we say that the part S_i *belongs* to the cluster μ. Further, a partition of a cyclically-ordered set $\mathcal{S} \subseteq V(G)$ is *admissible* if it is both good and non-crossing. We define the binary operator $\uplus$, called *generalized union*, that given two good partitions P' and P'' of ground sets $\mathcal{Q}'$ and $\mathcal{Q}''$, respectively, returns a good partition $P^* = P' \uplus P''$ of $\mathcal{Q}' \cup \mathcal{Q}''$ obtained as follows. Initialize $P^* = P' \cup P''$. Then, as long as there exist $Q_i, Q_j \in P^*$ such that $Q_i \cap Q_j \neq \emptyset$, replace sets Q_i and Q_j with their union $Q_i \cup Q_j$ in P^*. We have the following.

▶ **Lemma 7.** *$P^* = P' \uplus P''$ can be computed in $O(|\mathcal{Q}'| + |\mathcal{Q}''|)$ time.*

Let P be a good partition of the ground set $\mathcal{Q}$ and let $\mathcal{Q}' \subset \mathcal{Q}$. The *projection* of P *onto* $\mathcal{Q}'$, denoted as $P|_{\mathcal{Q}'}$, is the good partition of $\mathcal{Q}'$ obtained from P by first replacing each part $S_i \in P$ with $S_i \cap \mathcal{Q}'$ and then removing empty parts, if any.

An admissible partition P of a cyclically-ordered set $\mathcal{S}$ can be naturally associated with a 2-connected plane graph $G(P)$ as follows. The outer face of $G(P)$ is a cycle $C(P)$ whose vertices are the elements in $\mathcal{S}$ and the clockwise order in which they appear along $C(P)$ is the same as in $\mathcal{S}$. Also, for each part $S_i \in P$ such that $|S_i| \geq 2$, graph $G(P)$ contains a vertex v_i in the interior of $C(P)$ that is adjacent to all the elements in S_i, i.e., removing all the edges of $C(P)$ yields a collection of stars, whose central vertices are the v_i's, and isolated vertices. We say that $G(P)$ is the *cycle-star* associated with P; see, e.g., Fig. 3c.

We also extend the definitions of *generalized union* and *projection* to admissible partitions by regarding the corresponding cyclically-ordered sets as unordered.

Let P' and P'' be two admissible partitions of cyclically-ordered sets $\mathcal{S}'$ and $\mathcal{S}''$, respectively, with the following properties (where $\mathcal{S}_\cap = \{s_1, s_2, \ldots, s_k\}$ denotes the set of elements that are common to $\mathcal{S}'$ and $\mathcal{S}''$): (i) $|\mathcal{S}_\cap| \geq 2$ and $\mathcal{S}' \cup \mathcal{S}'' \setminus \mathcal{S}_\cap \neq \emptyset$, (ii) the elements of $\mathcal{S}_\cap$

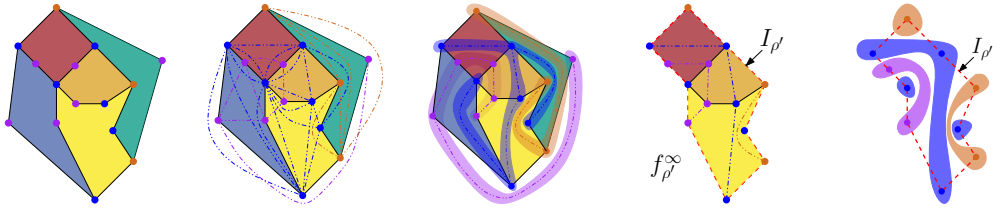


Figure 4 (a) A flat c-graph $\mathcal{C}(G,\mathcal{T})$. (b) A super c-graph of $\mathcal{C}$ containing all the candidate saturating edges. (c) A c-planar drawing of $\mathcal{C}$ and the corresponding planar saturation. (d) A planar saturation of a c-graph, whose underlying graph is the graph $G_{\rho'}$ of the decomposition in Fig. 1, where no saturating edge lies in the interior of $f^{\infty}_{\rho'}$. (e) The admissible partition P determined by the planar saturation in (d); sets of vertices of $I_{\rho'}$ belonging to the same cluster and connected by saturating edges in (d) form distinct parts in P (enclosed by shaded regions).

appear consecutively both in $\mathcal{S}'$ and $\mathcal{S}''$, and (iii) the cyclic ordering of the elements in $\mathcal{S}_\cap$ determined by $\mathcal{S}'$ is the reverse of the cyclic order of these elements determined by $\mathcal{S}''$. We define the binary operator $\text{(}\!\text{)}$, called *bubble merge*, that returns an admissible partition $P^* = P'\text{(}\!\text{)}P''$ obtained as follows. Consider the cycle-stars $G(P')$ and $G(P'')$ associated with P' and P'', respectively. First, we identify the vertices corresponding to the same element of $\mathcal{S}_\cap$ in both $C(P')$ and $C(P'')$ (see Fig. 3a) to obtain a new plane graph H (see Fig. 3b). Observe that H is 2-connected since $G(P')$ and $G(P'')$ are 2-connected and since $|\mathcal{S}_\cap| \geq 2$; therefore, the outer face f_H of H is a simple cycle. Second, we traverse f_H clockwise to construct a cyclically-ordered set $\mathcal{S}^* \subseteq \mathcal{S}' \cup \mathcal{S}''$ on the vertices of f_H. Finally, we set $P^* = (P' \uplus P'')|_{\mathcal{S}^*}$. P^* is good by the definition of generalized union. The fact that P^* is a non-crossing partition of $\mathcal{S}^*$ follows immediately by the planarity of H (see Fig. 3c). Lemma 7 and the fact that the graph H can be easily constructed from P' and P'' in linear time imply the following.

▶ **Lemma 8.** *$P^* = P'\text{(}\!\text{)}P''$ can be computed in $O(|\mathcal{S}'| + |\mathcal{S}''|)$ time.*

Algorithm. Let $\mathcal{C}(G,\mathcal{T})$ be a 2-connected flat c-graph. Let (D,γ) be a bond-carving decomposition of $\delta(G)$ of width at most ω and let ν be a non-root bag of D. We denote by F_ν the set of faces of G that are dual to the vertices of $\delta(G)$ that are leaves of the subtree D_ν of D rooted at ν. Also, let G_ν be the embedded subgraph of G induced by the edges of the faces in F_ν. The *interface graph* I_ν of ν is the subgraph of G_ν induced by the edges that are incident to a face of G_ν not in F_ν. The *boundary* B_ν of ν is the vertex set of I_ν. Note that, the edges of I_ν are dual to those in $\gamma(\nu)$. By Lemma 4 and by the definition of bond-carving decomposition, we derive the next observation about I_ν.

▶ **Observation 1.** *The interface graph I_ν of ν is a cycle of length at most ω.*

Since G is 2-connected, by Observation 1, the vertices in B_ν have a natural (clockwise) circular ordering defined by cycle I_ν, and I_ν bounds the *unique* face f^{∞}_ν of G_ν not in F_ν. Therefore, from now on, we regard B_ν as a cyclically-ordered set.

Let $P \in \mathcal{NP}(B_\nu)$ be an admissible partition and let $\mathcal{C}_\nu(G_\nu,\mathcal{T}_\nu)$ be the flat c-graph obtained by restricting $\mathcal{C}$ to G_ν. Also, let $\mathcal{C}^\diamond_\nu(G^\diamond_\nu,\mathcal{T}^\diamond_\nu)$ be a super c-graph of $\mathcal{C}_\nu$ containing no saturating edges in the interior of f^{∞}_ν and such that $G^\diamond_\nu$ is planar.

▶ **Definition 9.** *The c-graph* $\mathcal{C}_\nu^\diamond$ realizes P *if (refer to Fig. 4):*

(a) *for every two vertices* $u, v \in B_\nu$*, we have that* u *and* v *belong to the same part* $S_i \in P$ if and only if *they are connected in* $G_\nu^\diamond$ *by paths of intra-cluster edges of the cluster the part* S_i *belongs to,*

(b) *for each cluster* μ *in* $\mathcal{T}$ *such that* $V_\mu \cap B_\nu \neq \emptyset$*, all the vertices of* μ *in* G_ν *are connected to some vertex of* μ *in* B_ν *by paths of intra-cluster edges of* μ *in* $G_\nu^\diamond$*, and*

(c) *for each cluster* μ *in* $\mathcal{T}$ *such that* $V_\mu \subseteq V(G_\nu) \setminus B_\nu$*, all the vertices of* μ *are in* G_ν *and are connected by paths of intra-cluster edges of* μ *in* $G_\nu^\diamond$*.*

A vertex v of G_ν is *dominated by* some part S_i of P, if either $v \in S_i$ or v is connected to some vertex of S_i by paths of intra-cluster edges in $\mathcal{C}_\nu^\diamond$. Note that, by Conditions a and b of Definition 9, for each part S_i of P, the set of vertices of G_ν dominated by S_i induces a connected subgraph of $G_\nu^\diamond$. Also, by Condition c, cycle I_ν does not form a *cluster separator*, that is, a cycle of G such that the vertices of some cluster μ appear both in its interior and in its exterior, but not in it; note that, in fact, this is a necessary condition for the existence of a c-planar drawing of $\mathcal{C}$. Thus, partition P *"represents"* the internal-cluster connectivity in $\mathcal{C}_\nu^\diamond$ of the clusters whose vertices appear in B_ν *in a potentially positive instance*. Also, P is *realizable* by $\mathcal{C}_\nu$ if there exists a super c-graph $\mathcal{C}'_\nu(G'_\nu, \mathcal{T}'_\nu)$ of $\mathcal{C}_\nu$ that realizes P containing no saturating edges in the interior of f_ν^∞ and such that G'_ν is planar. From Theorem 3 and the definition of realizable partition we have the following.

▶ **Lemma 10** (Necessity)**.** *Let* ν *be a non-root bag of* D*. Then,* $\mathcal{C}$ *is c-planar only if there exists an admissible partition* P *of* B_ν *that is realizable by* $\mathcal{C}_\nu$*.*

We are going to exploit the next lemma, which holds for any bond-carving decomposition.

▶ **Lemma 11.** *Let* ρ' *and* ρ'' *be the two children of the root* ρ *of* D*. Then,* $I_{\rho'} = I_{\rho''}$*.*

Lemmas 10 and 11 allow us to derive the following useful characterization.

▶ **Theorem 12** (Characterization)**.** *The 2-connected flat c-graph* $\mathcal{C}(G, \mathcal{T})$ *is c-planar if and only if there exist admissible partitions* $P' \in \mathcal{NC}(B_{\rho'})$*,* $P'' \in \mathcal{NC}(B_{\rho''})$ *such that:*

(i) P' *and* P'' *are realizable by* $\mathcal{C}_{\rho'}(G_{\rho'}, \mathcal{T}_{\rho'})$ *and by* $\mathcal{C}_{\rho''}(G_{\rho''}, \mathcal{T}_{\rho''})$*, respectively, and*

(ii) *no two distinct parts* $S_i, S_j \in P^*$*, with* $P^* = P' \uplus P''$*, belong to the same cluster of* $\mathcal{T}$*.*

Proof. We first prove the *only if* part. The necessity of Condition i follows from Lemma 10. For the necessity of Condition ii, suppose for a contradiction, that for any two realizable partitions $P' \in \mathcal{NC}(B_{\rho'})$ and $P'' \in \mathcal{NC}(B_{\rho''})$, it holds that $P^* = P' \uplus P''$ does not satisfy such a condition. Then, there is no set of saturating edges of $\mathcal{C}_{\rho'}$ and $\mathcal{C}_{\rho''}$, where none of these edges lies in the interior of $f_{\rho'}^\infty$ and of $f_{\rho''}^\infty$, respectively, that when added to $\mathcal{C}$ yields a c-connected c-graph $\mathcal{C}^\diamond(G^\diamond, \mathcal{T}^\diamond)$ with $G^\diamond$ planar. Thus, by Theorem 3, $\mathcal{C}$ is not c-planar, a contradiction.

We now prove the *if part*. By Lemma 11, it holds $G = G_{\rho'} \cup G_{\rho''}$ and $I_{\rho'} = I_{\rho''} = G_{\rho'} \cap G_{\rho''}$. Let $\mathcal{C}_{\rho'}^\diamond$ be a super c-graph of $\mathcal{C}_{\rho'}$ realizing P' and let $\mathcal{C}_{\rho''}^\diamond$ be a super c-graph of $\mathcal{C}_{\rho''}$ realizing P''; these c-graphs exist since Condition i holds. Let $\mathcal{C}^\diamond$ be the super c-graph of $\mathcal{C}$ obtained by augmenting $\mathcal{C}$ with the saturating edges of both $\mathcal{C}_{\rho'}^\diamond$ and $\mathcal{C}_{\rho''}^\diamond$. Note that, $G^\diamond$ is planar.

We show that every cluster μ is connected in $\mathcal{C}^\diamond$, provided that Condition ii holds. This proves that $\mathcal{C}^\diamond$ is a c-connected super c-graph of $\mathcal{C}$, thus by Condition iii of Theorem 3, c-graph $\mathcal{C}$ is c-planar. We distinguish two cases, based on whether some vertices of μ appear along cycle $I_{\rho'} = I_{\rho''}$ or not. Let $B = B_{\rho'} = B_{\rho''}$.

Consider first a cluster μ containing vertices in B. By Condition b of Definition 9, we have that every vertex in μ is dominated by at least a part of P' or of P'', i.e., they either belong to B or they are connected by paths of intra-cluster edges in either $\mathcal{C}_{\rho'}^\diamond$ or $\mathcal{C}_{\rho''}^\diamond$ to a

vertex in B. Since, by Condition ii of the statement, there exists only one part $S_\mu \in P^*$ that contains vertices of cluster μ, we have that the different parts of P' and of P'' containing vertices of μ are joined together by the vertices of μ in B. Therefore, the cluster μ is connected in $\mathcal{C}^\diamond$. Finally, consider a cluster μ such that no vertex of μ belongs to B. Then, all the vertices of cluster μ only belong to either $\mathcal{C}_{\rho'}$ or $\mathcal{C}_{\rho''}$, by Condition c of Definition 9. Suppose that μ only belongs to $\mathcal{C}_{\rho'}$, the case when μ only belongs to $\mathcal{C}_{\rho''}$ is analogous. Since $\mathcal{C}^\diamond_{\rho'}$ realizes P', by Condition c of Definition 9, all the vertices of μ in G'_ρ are connected by paths of intra-cluster edges. Thus, cluster μ is connected in $\mathcal{C}^\diamond$, since it is connected in $\mathcal{C}^\diamond_{\rho'}$. This concludes the proof. ◀

We now present our main algorithmic tool.

Algorithm 1. Let (D, γ) be a bond-carving decomposition of $\delta(G)$ of width ω. Let ν be a non-root bag of D, we denote by R_ν the set of all the admissible partitions of B_ν that are realizable by $\mathcal{C}_\nu$. We process the bags of D bottom-up and compute the following *relevant information*, for each non-root bag ν of D: **1.** the set R_ν, and **2.** for each admissible partition $P \in R_\nu$ and for each part $S_i \in P$, the number $count(S_i)$ of vertices of cluster μ belonging to G_ν that are dominated by S_i, where μ is the cluster S_i belongs to.

- If ν is a leaf bag of D, then $G_\nu = I_\nu$ consists of the vertices and edges of a single face of G. Further, by Observation 1, graph G_ν is a cycle of length at most ω. In this case, R_ν simply coincides with the set of all the admissible partitions of B_ν. Therefore, we can construct R_ν by enumerating all the possible at most $\mathcal{CAT}(\omega) \leq 2^{2\omega}$ non-crossing partitions of B_ν and by testing whether each such partition is good in $O(\omega)$ time. Further, for each $P \in R_\nu$, we can compute all counters $count(S_i)$ for every $S_i \in P$, in total $O(\omega)$ time, by visiting cycle I_ν.
- If ν is a non-leaf non-root bag of D, we have already computed the relevant information for the two children ν' and ν'' of ν. In the following way, we either detect that $\mathcal{C}$ does not satisfy Condition iii of Theorem 3 or construct the relevant information for ν:

(1) Initialize $R_\nu = \emptyset$;

(2) For every pair of realizable admissible partitions $P' \in \mathcal{R}_{\nu'}$ and $P'' \in R_{\nu''}$, perform the following operations:

(2a) Compute $P^* = P' \uplus P''$ and compute the counters $count(S_i)$, for each $S_i \in P^*$, from the counters of the parts in $P' \cup P''$ whose union is S_i.

(2b) If there exists some $S_i \in P^*$ such that $S_i \cap B_\nu = \emptyset$ and $count(S_i)$ is smaller than the number of vertices in the cluster S_i belongs to, then **reject** the instance.

(2c) Compute $P = P' \textcircled{\scriptsize \cup} P''$ and add P to R_ν.

▶ Remark 13. Algorithm 1 rejects the instance at step (2b), if I_μ forms a cluster separator. This property is independent of the specific generalized union P^* considered at this step and implies that no P^* (and, thus, no P at step (2c)) can satisfy Condition c of Definition 9.

As the total number of pairs of partitions at step (2) is at most $(\mathcal{CAT}(\omega))^2$ and as P^* and P can be computed in $O(\omega)$ time, by Lemmas 7 and 8, we get the following.

▶ **Lemma 14.** *For each non-root bag ν of D, Algorithm 1 computes the relevant information for ν in $O(2^{4\omega+\log\omega})$ time, given the relevant information for its children.*

Proof. Let ν' and ν'' be the two children of ν in D. We will first show the correctness of the algorithm and then argue about the running time.

Let R^*_ν be the set of all the admissible partitions of B_ν that are realizable by $\mathcal{C}_\nu$ and let R_ν be the set of all the admissible partitions of B_ν computed by Algorithm 1. We show $R_\nu = R^*_\nu$.

We first prove $R^*_\nu \subseteq R_\nu$. Let P_ν be a realizable admissible partition in R^*_ν. Since P_ν is realizable by $\mathcal{C}_\nu$, there exists a super c-graph $\mathcal{C}^*_\nu(G^*_\nu, \mathcal{T}^*_\nu)$ of $\mathcal{C}_\nu$ that realizes P_ν containing no saturating edges in the interior of f^∞_ν and such that G^*_ν is planar. Let $\mathcal{C}^*_{\nu'}(G^*_{\nu'}, \mathcal{T}^*_{\nu'})$ (resp. $\mathcal{C}^*_{\nu''}(G^*_{\nu''}, \mathcal{T}^*_{\nu''})$) be the super c-graph of $\mathcal{C}_{\nu'}(G_{\nu'}, \mathcal{T}_{\nu'})$ (resp. of $\mathcal{C}_{\nu''}(G_{\nu''}, \mathcal{T}_{\nu''})$) obtained by adding to $\mathcal{C}_{\nu'}$ (resp. to $\mathcal{C}_{\nu''}$) all the saturating edges in G^*_ν laying in the interior of the faces of $G_{\nu'}$ (resp. of $G_{\nu''}$) that are also faces of G_ν. Clearly, c-graph $\mathcal{C}^*_{\nu'}(G^*_{\nu'}, \mathcal{T}^*_{\nu'})$ (resp. c-graph $\mathcal{C}^*_{\nu''}(G^*_{\nu''}, \mathcal{T}^*_{\nu''})$) contains no saturating edges in the interior of $f^\infty_{\nu'}$ (resp. in the interior of $f^\infty_{\nu''}$), since such a face does not belong to G_ν. Let P' and P'' be the admissible partitions of $B_{\nu'}$ and of $B_{\nu''}$ realized by $\mathcal{C}^*_{\nu'}(G^*_{\nu'}, \mathcal{T}^*_{\nu'})$ and by $\mathcal{C}^*_{\nu''}(G^*_{\nu''}, \mathcal{T}^*_{\nu''})$, respectively. By hypothesis, we have $P' \in R_{\nu'}$ and $P'' \in R_{\nu''}$. We show that when step (2) of Algorithm 1 considers partitions P' and P'', it successfully adds P_ν to the set R_ν. It is clear by the construction of P' and of P'' that $P_\nu = P' \between P''$. Therefore, we only need to show that when the algorithm considers the pair (P', P''), it does not reject the instance at step (2b), and thus P_ν is added to R_ν at step (2c). Let $P^* = P' \uplus P''$, which is constructed at step (2a) of the algorithm. Suppose, for a contradiction, that $\mathcal{C}$ is rejected at step (2b). Then, there exists a part S_i of P^* such that $S_i \cap B_\nu = \emptyset$ and $count(S_i)$ is smaller than the number of vertices in the cluster μ the part S_i belongs to. Therefore, the cluster μ contains vertices that belong to $G \setminus G_\nu$, which implies that P_ν cannot satisfy Condition c of Definition 9, a contradiction. This concludes the proof of this direction.

We now prove $R_\nu \subseteq R^*_\nu$. Let P be a partition in R_ν obtained from the partitions $P' \in R_{\nu'}$ and $P'' \in R_{\nu''}$ (selected at step (2) of the algorithm). We show that P is realizable by $\mathcal{C}_\nu$.

By the definition of realizable partition, there exists a super c-graph $\mathcal{C}^*_{\nu'}(G^*_{\nu'}, \mathcal{T}^*_{\nu'})$ (resp. $\mathcal{C}^*_{\nu''}(G^*_{\nu''}, \mathcal{T}^*_{\nu''})$) of $\mathcal{C}_{\nu'}(G_{\nu'}, \mathcal{T}_{\nu'})$ (resp. of $\mathcal{C}_{\nu''}(G_{\nu''}, \mathcal{T}_{\nu''})$) that realizes P' (resp. P'') containing no saturating edges in the interior of $f^\infty_{\nu'}$ (resp. of $f^\infty_{\nu''}$) and such that $G^*_{\nu'}$ (resp. $G^*_{\nu''}$) is planar. Let $\mathcal{C}^*_\nu(G^*_\nu, \mathcal{T}^*_\nu)$ be the super c-graph of $\mathcal{C}_\nu(G_\nu, \mathcal{T}_\nu)$ constructed by adding to $\mathcal{C}_\nu$ the saturating edges in $\mathcal{C}^*_{\nu'}$ and $\mathcal{C}^*_{\nu''}$. We show that the c-graph $\mathcal{C}^*_\nu(G^*_\nu, \mathcal{T}^*_\nu)$ realizes P, contains no saturating edges in the interior of f^∞_ν, and G^*_ν is planar.

First, we have that G^*_ν is planar, since $G^*_{\nu'}$ and $G^*_{\nu''}$ are planar and do not contain saturating edges in the interior of $f^\infty_{\nu'}$ and of $f^\infty_{\nu''}$, respectively. By the previous arguments, we also have that f^∞_ν contains no saturating edges.

We show that Condition a of Definition 9 holds. Recall that $P = P' \between P''$. Let S_i be a part of P that also belongs to P' or to P''. Then, since $\mathcal{C}^*_{\nu'}$ and $\mathcal{C}^*_{\nu''}$ realize P' and P'', respectively, the vertices of S_i are connected by paths of intra-cluster edges in C^*_ν as they are connected by paths of intra-cluster edges in either $\mathcal{C}^*_{\nu'}$ or $\mathcal{C}^*_{\nu''}$, by Condition a of Definition 9. Otherwise, let S_i be a part of P that does not belong to either P' or P''. Then, by the definition of bubble merge, the part S_i is obtained by projecting onto B_ν the generalized union $P^* = P' \uplus P''$. Thus, S_i is a subset of a part S^*_i of P^*. Also, the vertices in each of the parts of P' and of P'' contributing to the creation of S^*_i are connected by paths of intra-cluster edges in $\mathcal{C}^*_{\nu'}$ and $\mathcal{C}^*_{\nu''}$, respectively, by Condition a of Definition 9. Therefore, we have that the connectivity of such sets implies the connectivity of the elements of S_i by paths of intra-cluster edges that connect at their shared vertices in $B_{\nu'} \cap B_{\nu''}$. We show that Condition b of Definition 9 holds. Suppose, for a contradiction, that there exists some cluster μ whose vertices appear in B_ν such that there is at least a vertex of μ in G_ν that is not connected by a path of intra-cluster edges to some vertex of μ in B_ν. Then, consider the part $S_i \in P^*$ that dominates this vertex, which exists since P' and P'' are realizable by $\mathcal{C}_{\nu'}$ and by $\mathcal{C}_{\nu''}$, respectively. We have that $S_i \cap B_\nu = \emptyset$ and that $count(S_i)$ is smaller than the

number of vertices of μ. Thus, step (2b) would reject the instance, and thus P would not be added to R_ν, a contradiction. Finally, we show that Condition c of Definition 9 holds. Suppose, for a contradiction, that there exists some cluster μ whose vertices only belong to $V(G_\nu) \setminus B_\nu$ and that there exist two vertices u and v of μ in G_ν that are not connected by a path of intra-cluster edges in G^*_ν. Then, consider the part $S_i \in P^*$ that dominates u. Observe that, S_i does not dominate v. Similarly to the proof of Condition b, we have that $S_i \cap B_\nu = \emptyset$ and that $count(S_i)$ is smaller than the number of vertices of μ. Thus, step (2b) would reject the instance, and thus P would not be added to R_ν, a contradiction.

We conclude by analyzing the running time. Step (2a) can be performed in linear time in the sum of the sizes of $B_{\nu'}$ and $B_{\nu''}$, since the generalized union P^* can be computed in $O(|B_{\nu'}| + |B_{\nu''}|)$ time, by Lemma 7, and since the size of P^*, and thus the number of counters to be updated, is in $O(|B_{\nu'}| + |B_{\nu''}|)$. Step (2b) can also be done in linear time by the previous argument. Step (2c) can be performed in $O(|B_{\nu'}| + |B_{\nu''}|)$ time, by Lemma 8. Further, the number of pairs of realizable partitions considered at step (2) is bounded by $|\mathcal{NC}(B_{\nu'})| \cdot |\mathcal{NC}(B_{\nu''})|$, which is bounded by $2^{2(|B_{\nu'}|+|B_{\nu''}|)}$. Finally, $|B_{\nu'}| \leq \omega$ and $|B_{\nu''}| \leq \omega$. Thus, Algorithm 1 runs in $O(2^{4\omega}\omega) = O(2^{4\omega+\log\omega})$ time. ◀

By Lemma 14 and since D contains $O(n)$ bags, we have the following.

▶ **Lemma 15.** *Sets $R_{\rho'}$ and $R_{\rho''}$ can be computed in $O(2^{4\omega+\log\omega}n)$ time.*

We obtain the next theorem by combining Lemma 15 and Theorem 12, where the additive $O(n^2)$ factor in the running time derives from the time needed to convert a carving decomposition of $\delta(G)$ into a bond-carving decomposition of the same width [54].

▶ **Theorem 16.** C-Planarity Testing *can be solved in $O(2^{4\omega+\log\omega}n + n^2)$ time for any 2-connected n-vertex flat c-graph $\mathcal{C}(G, \mathcal{T})$, if a carving decomposition of $\delta(G)$ of width ω is provided.*

We are finally ready to prove our main result.

Proof of Theorem 1. Let (D, γ) be a carving decomposition of $\delta(G)$ of optimal width $\omega = \mathrm{cw}(\delta(G))$. First, we apply Lemma 6 to $\mathcal{C}$ to obtain, in $O(n)$ time, a 2-connected flat c-graph $\mathcal{C}'(G', \mathcal{T}')$ equivalent to $\mathcal{C}$ and a corresponding carving decomposition (D', γ') of width $\omega' \leq \max(\omega, 4)$. Then, we apply Theorem 16 to test whether $\mathcal{C}'$ (and thus $\mathcal{C}$) is c-planar. The running time follows from the running time of Theorem 16, from the fact that $\omega' = O(\omega)$, $|V(G')| \in O(n)$, and that a carving decomposition of $\delta(G)$ of optimal width can be computed in $O(n^3)$ time [39, 55]. This concludes the proof of the theorem.

We remark that in the recent reduction presented by Patrignani and Cortese to convert any non-flat c-graph $\mathcal{C}(G, \mathcal{T})$ into an equivalent independent flat c-graph $\mathcal{C}'(G', \mathcal{T}')$, the carving-width of $\delta(G')$ is within an $O(h)$ multiplicative factor from the carving-width of $\delta(G)$, where h is the height of $\mathcal{T}$. This is due to the fact that, by [24, Lemma 10], G' is a subdivision of G (which implies that $\mathrm{tw}(G') = \mathrm{tw}(G)$) and that each inter-cluster edge of G is replaced by a path of length at most $4h-4$ in G' (which implies that $\ell(G') = \ell(G)(4h-4)$). Therefore, by [24] and by the results presented in this section, we immediately derive an FPT algorithm for the non-flat case parameterized by h and the dual carving-width of G. In the full version [47], we show how to drop the dependency on h, by suitably adapting the relevant concepts defined for the flat case so that Algorithm 1 can also be applied to the non-flat case.

4 Graph-Width Parameters Related to the Dual Carving-Width

In this section, we discuss implications of our algorithm for instances of bounded embedded-width and of bounded dual cut-width.

Embedded-width. A tree decomposition of an embedded graph G *respects* the embedding of G if, for every face f of G, at least one bag contains all the vertices of f [14]. The *embedded-width* $\mathrm{emw}(G)$ of G is the minimum width of any of its tree decompositions that respect the embedding of G. For consistency with other graph-width parameters, in the original definition of this width measure [14] the vertices of the outer face are not required to be in some bag. Here, we adopt the variant presented in [26], where the tree decomposition must also include a bag containing the outer face. In the full version [47], we prove the following.

▶ **Lemma 17.** *Let G be an embedded graph. Then,* $\mathrm{cw}(\delta(G)) \leq \mathrm{emw}^2(G) + 2\mathrm{emw}(G)$.

Cut-width. Let π be a linear order of the vertex set of a graph $G = (V, E)$. By splitting π into two linear orders π_1 and π_2 such that π is the concatenation of π_1 and π_2, we define a *cut* of π. The *width* of this cut is the number of edges between a vertex in π_1 and a vertex in π_2. The *width* of π is the maximum width over all its possible cuts. Finally, the *cut-width* of G is the minimum width over all the possible linear orders of V. The *dual cut-width* is the cut-width of the dual of G.

The following relationship between cut-width and carving-width has been proved in [52].

▶ **Theorem 18** (Theorem 4.3, [52])**.** *The carving-width of G is at most twice its cut-width.*

By Lemma 17 and Theorem 18, we have that single-parameter FPT algorithms also exist with respect to the embedded-width and to the dual cut-width of the underlying graph.

5 Conclusions

In this paper, we studied the C-Planarity Testing problem for c-graphs with a prescribed combinatorial embedding. We showed that the problem is polynomial-time solvable when the dual carving-width of the underlying graph of the input c-graph is bounded. In particular, this addresses a question we posed in [26], regarding the existence of notable graph-width parameters such that the C-Planarity Testing problem is fixed-parameter tractable with respect to a single one of them. Namely, we answer this question in the affirmative when the parameters are the embedded-width of the underlying graph, and the carving-width and cut-width of its planar dual.

References

1 Isolde Adler, Binh-Minh Bui-Xuan, Yuri Rabinovich, Gabriel Renault, Jan Arne Telle, and Martin Vatshelle. On the Boolean-Width of a Graph: Structure and Applications. In Dimitrios M. Thilikos, editor, *WG 2010*, volume 6410 of *LNCS*, pages 159–170, 2010. doi:10.1007/978-3-642-16926-7_16.

2 Hugo A. Akitaya, Radoslav Fulek, and Csaba D. Tóth. Recognizing Weak Embeddings of Graphs. In Artur Czumaj, editor, *SODA '18*, pages 274–292. SIAM, 2018. doi:10.1137/1.9781611975031.20.

3 Patrizio Angelini and Giordano Da Lozzo. SEFE = C-Planarity? *Comput. J.*, 59(12):1831–1838, 2016. doi:10.1093/comjnl/bxw035.

4 Patrizio Angelini and Giordano Da Lozzo. Clustered Planarity with Pipes. *Algorithmica*, 81(6):2484–2526, 2019. doi:10.1007/s00453-018-00541-w.

5 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, and Fabrizio Frati. Strip Planarity Testing for Embedded Planar Graphs. *Algorithmica*, 77(4):1022–1059, 2017. doi:10.1007/s00453-016-0128-9.

6 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Vincenzo Roselli. Relaxing the constraints of clustered planarity. *Comput. Geom.*, 48(2):42–75, 2015. doi:10.1016/j.comgeo.2014.08.001.

7 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Ignaz Rutter. Intersection-Link Representations of Graphs. *J. Graph Algorithms Appl.*, 21(4):731–755, 2017. doi:10.7155/jgaa.00437.

8 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, and Vincenzo Roselli. The importance of being proper: (In clustered-level planarity and T-level planarity). *Theor. Comput. Sci.*, 571:1–9, 2015. doi:10.1016/j.tcs.2014.12.019.

9 Patrizio Angelini, Fabrizio Frati, and Michael Kaufmann. Straight-Line Rectangular Drawings of Clustered Graphs. *Discrete & Computational Geometry*, 45(1):88–140, 2011. doi:10.1007/s00454-010-9302-z.

10 Jan Christoph Athenstädt and Sabine Cornelsen. Planarity of Overlapping Clusterings Including Unions of Two Partitions. *J. Graph Algorithms Appl.*, 21(6):1057–1089, 2017. doi:10.7155/jgaa.00450.

11 T. Biedl. Drawing Planar Partitions III: Two Constrained Embedding Problems. Tech. Report RRR 13-98, Rutcor Research Report, 1998.

12 Therese C. Biedl and Martin Vatshelle. The Point-Set Embeddability Problem for Plane graphs. *Int. J. Comput. Geometry Appl.*, 23(4-5):357–396, 2013. doi:10.1142/S0218195913600091.

13 Thomas Bläsius and Ignaz Rutter. A new perspective on clustered planarity as a combinatorial embedding problem. *Theor. Comput. Sci.*, 609:306–315, 2016. doi:10.1016/j.tcs.2015.10.011.

14 Glencora Borradaile, Jeff Erickson, Hung Le, and Robbie Weber. Embedded-width: A variant of treewidth for plane graphs, 2017. arXiv:1703.07532.

15 Vincent Bouchitté, Frédéric Mazoit, and Ioan Todinca. Treewidth of planar graphs: connections with duality. *ENDM*, 10:34–38, 2001. doi:10.1016/S1571-0653(04)00353-1.

16 Franz-Josef Brandenburg, David Eppstein, Michael T. Goodrich, Stephen G. Kobourov, Giuseppe Liotta, and Petra Mutzel. Selected Open Problems in Graph Drawing. In Giuseppe Liotta, editor, *GD '03*, volume 2912 of *LNCS*, pages 515–539. Springer, 2003. doi:10.1007/978-3-540-24595-7_55.

17 Markus Chimani, Giuseppe Di Battista, Fabrizio Frati, and Karsten Klein. Advances on Testing C-Planarity of Embedded Flat Clustered Graphs. In Christian A. Duncan and Antonios Symvonis, editors, *GD '14*, volume 8871 of *LNCS*, pages 416–427. Springer, 2014. doi:10.1007/978-3-662-45803-7_35.

18 Markus Chimani and Karsten Klein. Shrinking the Search Space for Clustered Planarity. In Walter Didimo and Maurizio Patrignani, editors, *GD '12*, volume 7704 of *LNCS*, pages 90–101. Springer, 2012. doi:10.1007/978-3-642-36763-2_9.

19 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.

20 Sabine Cornelsen and Dorothea Wagner. Completely connected clustered graphs. *J. Discrete Algorithms*, 4(2):313–323, 2006. doi:10.1016/j.jda.2005.06.002.

21 Pier Francesco Cortese and Giuseppe Di Battista. Clustered planarity. In Joseph S. B. Mitchell and Günter Rote, editors, *SoCG '05*, pages 32–34. ACM, 2005. doi:10.1145/1064092.1064093.

22 Pier Francesco Cortese, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Maurizio Pizzonia. C-Planarity of C-Connected Clustered Graphs. *J. Graph Algorithms Appl.*,

12(2):225–262, 2008. URL: http://jgaa.info/accepted/2008/Cortese+2008.12.2.pdf, doi:10.7155/jgaa.00165.

23 Pier Francesco Cortese, Giuseppe Di Battista, Maurizio Patrignani, and Maurizio Pizzonia. On embedding a cycle in a plane graph. *Discrete Mathematics*, 309(7):1856–1869, 2009. doi:10.1016/j.disc.2007.12.090.

24 Pier Francesco Cortese and Maurizio Patrignani. Clustered Planarity = Flat Clustered Planarity. In Therese C. Biedl and Andreas Kerren, editors, *GD 2018*, volume 11282 of *LNCS*, pages 23–38. Springer, 2018. doi:10.1007/978-3-030-04414-5_2.

25 Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, and Maurizio Patrignani. Computing NodeTrix Representations of Clustered Graphs. *J. Graph Algorithms Appl.*, 22(2):139–176, 2018. doi:10.7155/jgaa.00461.

26 Giordano Da Lozzo, David Eppstein, Michael T. Goodrich, and Siddharth Gupta. Subexponential-Time and FPT Algorithms for Embedded Flat Clustered Planarity. In Andreas Brandstädt, Ekkehard Köhler, and Klaus Meer, editors, *WG 2018*, volume 11159 of *LNCS*, pages 111–124. Springer, 2018. doi:10.1007/978-3-030-00256-5_10.

27 Elias Dahlhaus. A Linear Time Algorithm to Recognize Clustered Graphs and Its Parallelization. In Claudio L. Lucchesi and Arnaldo V. Moura, editors, *LATIN '98*, volume 1380 of *LNCS*, pages 239–248. Springer, 1998. doi:10.1007/BFb0054325.

28 Giuseppe Di Battista, Walter Didimo, and A. Marcandalli. Planarization of Clustered Graphs. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *GD '01*, volume 2265 of *LNCS*, pages 60–74. Springer, 2001. doi:10.1007/3-540-45848-4_5.

29 Giuseppe Di Battista and Fabrizio Frati. Efficient C-Planarity Testing for Embedded Flat Clustered Graphs with Small Faces. *J. Graph Algorithms Appl.*, 13(3):349–378, 2009. URL: http://jgaa.info/accepted/2009/DiBattistaFrati2009.13.3.pdf, doi:10.7155/jgaa.00191.

30 Walter Didimo, Francesco Giordano, and Giuseppe Liotta. Overlapping Cluster Planarity. *J. Graph Algorithms Appl.*, 12(3):267–291, 2008. URL: http://jgaa.info/accepted/2008/DidimoGiordanoLiotta2008.12.3.pdf, doi:10.7155/jgaa.00167.

31 Qing-Wen Feng, Robert F. Cohen, and Peter Eades. Planarity for Clustered Graphs. In Paul G. Spirakis, editor, *ESA'95*, volume 979 of *LNCS*, pages 213–226. Springer, 1995. doi:10.1007/3-540-60313-1_145.

32 Michael Forster and Christian Bachmaier. Clustered Level Planarity. In Peter van Emde Boas, Jaroslav Pokorný, Mária Bieliková, and Julius Stuller, editors, *SOFSEM '04*, volume 2932 of *LNCS*, pages 218–228. Springer, 2004. doi:10.1007/978-3-540-24618-3_18.

33 Radoslav Fulek and Jan Kyncl. Hanani-Tutte for Approximating Maps of Graphs. In Bettina Speckmann and Csaba D. Tóth, editors, *SoCG '18*, volume 99 of *LIPIcs*, pages 39:1–39:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.SoCG.2018.39.

34 Radoslav Fulek, Jan Kyncl, Igor Malinovic, and Dömötör Pálvölgyi. Efficient c-planarity testing algebraically. *CoRR*, abs/1305.4519, 2013. arXiv:1305.4519.

35 Radoslav Fulek, Jan Kyncl, Igor Malinovic, and Dömötör Pálvölgyi. Clustered Planarity Testing Revisited. *Electr. J. Comb.*, 22(4):P4.24, 2015. URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/v22i4p24.

36 Radoslav Fulek and Csaba D. Tóth. Atomic Embeddability, Clustered Planarity, and Thickenability. *CoRR*, abs/1907.13086, 2019. arXiv:1907.13086.

37 Christopher D. Godsil and Gordon F. Royle. *Algebraic Graph Theory*. Graduate texts in mathematics. Springer, 2001. doi:10.1007/978-1-4613-0163-9.

38 Michael T. Goodrich, George S. Lueker, and Jonathan Z. Sun. C-Planarity of Extrovert Clustered Graphs. In Patrick Healy and Nikola S. Nikolov, editors, *GD '05*, volume 3843 of *LNCS*, pages 211–222. Springer, 2005. doi:10.1007/11618058_20.

39 Qian-Ping Gu and Hisao Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ Time. *ACM Trans. Algorithms*, 4(3):30:1–30:13, 2008. doi:10.1145/1367064.1367070.

40 Carsten Gutwenger, Michael Jünger, Sebastian Leipert, Petra Mutzel, Merijam Percan, and René Weiskircher. Advances in C-Planarity Testing of Clustered Graphs. In Stephen G. Kobourov and Michael T. Goodrich, editors, *GD '02*, volume 2528 of *LNCS*, pages 220–235. Springer, 2002. doi:10.1007/3-540-36151-0_21.

41 Carsten Gutwenger, Petra Mutzel, and Marcus Schaefer. Practical Experience with Hanani-Tutte for Testing c-Planarity. In Catherine C. McGeoch and Ulrich Meyer, editors, *ALENEX '14*, pages 86–97. SIAM, 2014. doi:10.1137/1.9781611973198.9.

42 Seok-Hee Hong and Hiroshi Nagamochi. Convex drawings of hierarchical planar graphs and clustered planar graphs. *J. Discrete Algorithms*, 8(3):282–295, 2010. doi:10.1016/j.jda.2009.05.003.

43 Seok-Hee Hong and Hiroshi Nagamochi. Simpler algorithms for testing two-page book embedding of partitioned graphs. *Theoretical Computer Science*, 2016. doi:10.1016/j.tcs.2015.12.039.

44 John E. Hopcroft and Robert Endre Tarjan. Efficient Algorithms for Graph Manipulation [H] (Algorithm 447). *Commun. ACM*, 16(6):372–378, 1973. doi:10.1145/362248.362272.

45 Vít Jelínek, Eva Jelínková, Jan Kratochvíl, and Bernard Lidický. Clustered Planarity: Embedded Clustered Graphs with Two-Component Clusters. In Ioannis G. Tollis and Maurizio Patrignani, editors, *GD '08*, volume 5417 of *LNCS*, pages 121–132. Springer, 2008. doi:10.1007/978-3-642-00219-9_13.

46 Eva Jelínková, Jan Kára, Jan Kratochvíl, Martin Pergel, Ondrej Suchý, and Tomás Vyskocil. Clustered Planarity: Small Clusters in Cycles and Eulerian Graphs. *J. Graph Algorithms Appl.*, 13(3):379–422, 2009. URL: http://jgaa.info/accepted/2009/Jelinkova+2009.13.3.pdf, doi:10.7155/jgaa.00192.

47 Giordano Da Lozzo, David Eppstein, Michael T. Goodrich, and Siddharth Gupta. C-Planarity Testing of Embedded Clustered Graphs with Bounded Dual Carving-Width, 2019. arXiv:1910.02057.

48 Hiroshi Nagamochi and Katsutoshi Kuroya. Drawing c-planar biconnected clustered graphs. *Discrete Applied Mathematics*, 155(9):1155–1174, 2007. doi:10.1016/j.dam.2006.04.044.

49 Sang-il Oum. Rank-width is less than or equal to branch-width. *Journal of Graph Theory*, 57(3):239–244, 2008. doi:10.1002/jgt.20280.

50 Neil Robertson and Paul D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991. doi:10.1016/0095-8956(91)90061-N.

51 Juanjo Rué, Ignasi Sau, and Dimitrios M. Thilikos. Dynamic programming for graphs on surfaces. *ACM Trans. Algorithms*, 10(2):8:1–8:26, 2014. doi:10.1145/2556952.

52 Róbert Sasák. *Comparing 17 graph parameters*. Master's thesis, Department of Informatics, University of Bergen, Bergen, Norway, 2010.

53 Marcus Schaefer. Toward a Theory of Planarity: Hanani-Tutte and Planarity Variants. *J. Graph Algorithms Appl.*, 17(4):367–440, 2013. doi:10.7155/jgaa.00298.

54 Paul D. Seymour and Robin Thomas. Call Routing and the Ratcatcher. *Combinatorica*, 14(2):217–241, 1994. doi:10.1007/BF01215352.

55 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Constructive Linear Time Algorithms for Small Cutwidth and Carving-Width. In D. T. Lee and Shang-Hua Teng, editors, *ISAAC '00*, volume 1969 of *LNCS*, pages 192–203. Springer, 2000. doi:10.1007/3-540-40996-3_17.

56 Juan José Besa Vial, Giordano Da Lozzo, and Michael T. Goodrich. Computing k-Modal Embeddings of Planar Digraphs. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *ESA 2019*, volume 144 of *LIPIcs*, pages 17:1–17:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.17.

The Complexity of Packing Edge-Disjoint Paths

Jan Dreier
Dept. of Computer Science, RWTH Aachen University, Germany
dreier@cs.rwth-aachen.de

Janosch Fuchs
Dept. of Computer Science, RWTH Aachen University, Germany
fuchs@algo.rwth-aachen.de

Tim A. Hartmann
Dept. of Computer Science, RWTH Aachen University, Germany
hartmann@algo.rwth-aachen.de

Philipp Kuinke
Dept. of Computer Science, RWTH Aachen University, Germany
kuinke@cs.rwth-aachen.de

Peter Rossmanith
Dept. of Computer Science, RWTH Aachen University, Germany
rossmani@cs.rwth-aachen.de

Bjoern Tauer
Dept. of Computer Science, RWTH Aachen University, Germany
tauer@algo.rwth-aachen.de

Hung-Lung Wang
Computer Science and Information Engineering, National Taiwan Normal University, Taiwan
hlwang@gapps.ntnu.edu.tw

Abstract

We introduce and study the complexity of Path Packing. Given a graph G and a list of paths, the task is to embed the paths edge-disjoint in G. This generalizes the well known Hamiltonian-Path problem.

Since Hamiltonian Path is efficiently solvable for graphs of small treewidth, we study how this result translates to the much more general Path Packing. On the positive side, we give an FPT-algorithm on trees for the number of paths as parameter. Further, we give an XP-algorithm with the combined parameters maximal degree, number of connected components and number of nodes of degree at least three. Surprisingly the latter is an almost tight result by runtime and parameterization. We show an ETH lower bound almost matching our runtime. Moreover, if two of the three values are constant and one is unbounded the problem becomes NP-hard.

Further, we study restrictions to the given list of paths. On the positive side, we present an FPT-algorithm parameterized by the sum of the lengths of the paths. Packing paths of length two is polynomial time solvable, while packing paths of length three is NP-hard. Finally, even the spacial case Exact Path Packing where the paths have to cover every edge in G exactly once is already NP-hard for two paths on 4-regular graphs.

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

Keywords and phrases parameterized complexity, embedding, packing, covering, Hamiltonian path, unary binpacking, path-perfect graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.10

Related Version A full version of the paper is available at `https://arxiv.org/abs/1910.00440`.

14th International Symposium on Parameterized and Exact Computation (IPEC 2019).
Editors: Bart M. P. Jansen and Jan Arne Telle; Article No. 10; pp. 10:1–10:16

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

1 Introduction

Packing, covering and partitioning are well researched fields in graph theory. In general, the task is to cover a given graph $G = (V, E)$ with or partition it into smaller substructures, or to pack given structures into the graph. Besides that these terms are often used, they are not well defined throughout the literature. Thus, it is important to define problems in this field carefully and in detail.

For example, the path partition problem is a well studied problem [2, 4, 6, 12, 18, 21, 26] which is also known as path cover problem. The task is to cover all vertices of a graph with vertex-disjoint paths. This is equivalent to partitioning the graph into vertex-disjoint paths. The smallest number of paths to achieve this is called the path partition number or path cover number. Observe that G has a Hamiltonian path iff the path-partition number is one, thus the problem is NP-complete.

An NP-complete variant of this problem is the k-path partition problem [22, 27, 19]. Here the task is to partition a graph G into paths, such that none of the path lengths exceeds k. Observe that the 1-path-partition problem corresponds to finding a maximum matching.

Another related problem is the recognition of path-perfect graphs [5, 11, 13, 23, 29], which we denote in this work as Path-Perfect Packing. Instead of partitioning the graph into vertex-disjoint paths, the complete edge set must be partitioned into edge-disjoint paths of ascending length, starting by one. This can also be understood as packing k paths of length 1 to k into G without using an edge twice or leaving one edge uncovered.

This approach of packing smaller subgraphs into a given graph is also well researched [28]. For example, packing edge-disjoint trees into a clique is considered [24]. Since packing edge-disjoint and vertex-disjoint triangles is NP-hard for planar graphs, the parameterized complexity is studied [7].

We generalize the path-perfect graph problem and ask for a given graph G and a list of k paths $P = \{p_1, \ldots, p_k\}$ if they can be embedded into G without using the same edge twice. Note that we define the length of a path equals its number of edges. This problem arises naturally when restricting the path partition problem to edge-disjoint paths instead of vertex-disjoint paths. We denote this problem as Path Packing. Let us formalize what we mean by embedding. An embedding of a graph H into a graph G is an injective mapping $f : V(H) \to V(G)$ such that for every original edge $(u, v) \in E(H)$ also $(f(u), f(v)) \in E(G)$. An embedding of a list of graphs $\mathcal{H}$ into G is an embedding of each graph H into G. Note, that we do not ask to embed the graphs pairwise vertex-disjointly. The embeddings we consider in this work are pairwise edge-disjoint embeddings of paths.

Path Packing
Input: A list of paths $P = \{p_1, \ldots, p_k\}$ of length $l_1, \ldots, l_k$. A graph $G = (V, E)$.
Question: Is there an edge-disjoint embedding of P into G?

The Exact Path Packing problem additionally requires that every edge is covered *exactly* once.

Exact Path Packing
Input: A list of paths $P = \{p_1, \ldots, p_k\}$ of length $l_1, \ldots, l_k$. A graph $G = (V, E)$.
Question: Is there an edge-disjoint embedding of P into G such that each edge $e \in E$ is covered exactly once?

Path Packing is clearly more general than Exact Path Packing, since one can reduce from one to the other by additionally requiring the sum of the path lengths to be equal to the number of edges in the graph. Most of our hardness results are for Exact Path Packing, and therefore translate to Path Packing. Our upper bounds are always regarding more the general Path Packing.

Our Results

The Hamiltonian path problem is a special case of Path Packing. An even though the Hamiltonian path problem is tractable on graphs of bounded treewidth, Path Packing is already NP-complete on subdivided stars. Therefore, we focus on the parameterized complexity to classify this problem on a finer scale. We will analyze the impact of various parameters.

In Section 3, we analyze the parameterized complexity of our packing problems with respect to the number of paths (denoted by k). On the one hand, we give an FPT algorithm for Path Packing that solves the problem in time $2^k n^{O(1)}$ on subcubic (i.e. degree at most three) forests (Theorem 3). On the other hand, we show that Exact Path Packing on graphs with treewidth two is $W[1]$-hard and cannot be solved in time $f(k)n^{o(k/\log k)}$ under ETH (Theorem 11).

In Section 4 we introduce path dependent restrictions. We show that Exact Path Packing is NP-complete even for two paths on 4-regular graphs (Theorem 14). length i is easy for $i = 2$ and NP-complete for $i = 3$ (Theorem 15). If we however parameterize by the summed length of all paths Path Packing is in FPT (Theorem 16).

After parameterizing by the number of paths and their lengths, we further analyze graph dependent parameters in Section 5. We introduce the *bcd-number* of a graph, which is the maximum of the number of components, the maximal degree, and the number of vertices with degree larger than two. We show that Path Packing can be solved in time $k!^k(n+k^2)^{O(k^2)}$, where k is the bcd-number (Theorem 20). This is complemented by showing that the problem cannot be solved in $f(k)n^{o(k^2/\log k)}$ under ETH (Theorem 21). We further show that all three bcd parameters are necessary: If two values are constant and one is unbounded the problem becomes NP-hard (Theorem 1, Corollary 18, Theorem 19).

Note that, one can embed paths $p_1, \ldots, p_k$ as edge-disjoint subgraphs into a graph G if and only if one can embed these paths as vertex-disjoint induced subgraphs into the linegraph of G. Therefore, our results yield new insights for the problem of covering a graph with a list of vertex-disjoint induced paths [17]. Especially, our hardness results for certain graph classes transfer to hardness results on the linegraphs of these graph classes. Due to space limitations, we omit some proofs, and refer to the full version.

2 Preliminaries

All graphs are simple (i.e. without multi-edges or self-loops). The length of a path equals its number of edges.

3 Path Packing on Forests

Our packing problem is a generalization of the Hamiltonian path problem and therefore NP-complete. The Hamiltonian path problem is solvable in polynomial time if the treewidth of the input graph is bounded [9]. We show that (unlike Hamiltonian path) Exact Path Packing is NP-complete on trees. This is done by reducing it to the following NP-complete partitioning problem.

Multi-Way Number Partition

Input: A list of weights $w_1, \ldots w_n \in \mathbf{N}$ encoded in unary, and an integer $k \in \mathbf{N}$.

Question: Is there a partition of $w_1, \ldots w_n$ into k multi-sets $S_1, \ldots, S_k$ such that $\sum_{w_i \in S_j} w_i = \frac{1}{k} \sum_{i=1}^{n} w_i$, for every $1 \leq j \leq k$?

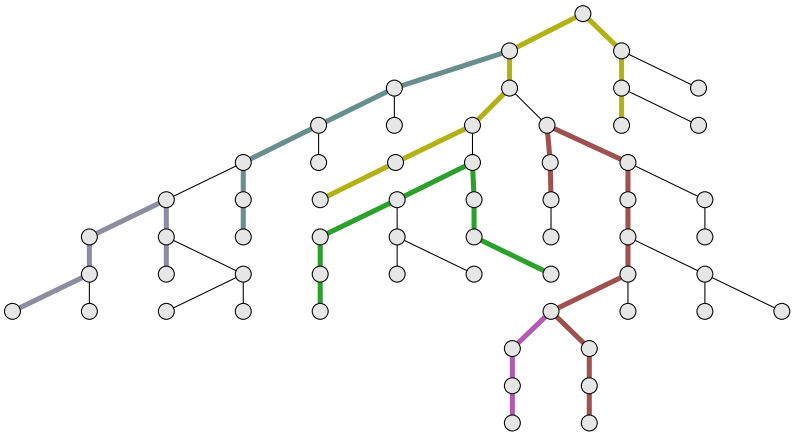


Figure 1 Packing paths of lengths 10, 8, 7, 5, 5, 3 into a subcubic tree. Although the packing looks very loose there is no solution if we replace 3 by 4.

We reduce from Multi-Way Number Partition to prove that Exact Path Packing is NP-hard on very simple trees.

▶ **Theorem 1.** Exact Path Packing *is* NP*-complete on subdivided stars.*

The previous reduction required a large number of paths. Therefore, in the following, we analyze the Path Packing problem parameterized by the number of paths.

Fast subset convolution

We develop dynamic programming algorithms on subcubic trees whose running time will be $O^*(2^k)$, where k is the number of paths that we want to pack. First we develop a naive and not too complicated algorithm with running time $O^*(3^k)$, whose longer running time is due to some very simple operation that occurs when we combine two dynamic programming tables. Björklund, Husfeldt, Kaski, and Koivisto introduced a technique called *fast subset convolution* that was used to speed up the computation of Steiner trees with small integer weights [3] and also to speed up some algorithms that do dynamic programming on tree decompositions [25]. We can use this technique to our advantage to significantly speed up the path packing algorithm on trees. The result that we will be using is:

▶ **Proposition 2.** [3] *Let N be a set of n natural numbers and $f, g\colon \mathbf{N} \to \mathbf{N}$ two functions. Then we can compute $(f * g)(S)$ for all $S \subseteq N$ in time $O(2^n n^2)$ if f and g can be evaluated in constant time and where $(f * g)(S) = \sum_{T \subset S} f(T)g(S - T)$. Here $f(S) = \sum_{i \in S} f(i)$.*

▶ **Theorem 3.** *We can solve* Path Packing *for k paths in time $O^*(2^k)$ on subcubic forests.*

Proof. Let us assume that the graph is a subcubic tree T, but the proof easily generalizes to subcubic forests. Let $l_1, \ldots, l_k$ be length of the paths that we want to pack into T. We can further assume that T is a rooted tree by designating an arbitrary vertex as its root. If v is a vertex of T then let $T(v)$ be the subtree rooted at v.

We solve the path packing problem by dynamic programming computing a table for each vertex in a bottom-up order. Such a table is a mapping $L\colon V \times 2^{[k]} \to [n] \cup \{-\infty\}$. The size of this table is $O(2^k n)$. We interpret the content of the table as follows:

$L(v, P) = r$ with $r \geq 0$ means that it is possible to pack all paths with indices in P (in short all P-paths) into the subtree $T(v)$ and additionally a path of length r that ends in v.

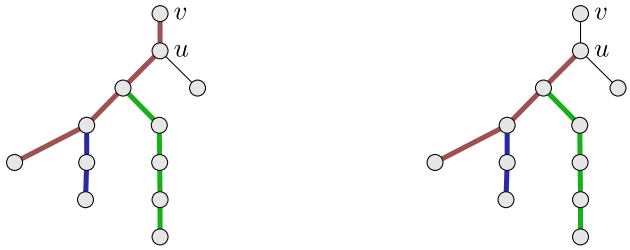


Figure 2 Left side: Packing paths of lengths $l_1 = 4$, $l_2 = 4$, $l_3 = 2$ into $T(v)$. $L(u, \{1, 2, 3\}) = -\infty$, but $L(u, \{1, 2, 3\} - \{1\}) = l_1 - 1$, so $L(v, \{1, 2, 3\}) = 0$.
Right side: Now $l_1 = 4$, $l_2 = 3$, $l_3 = 2$. $L(u, \{1, 2, 3\}) = 1$, so $L(v, \{1, 2, 3\}) = 1 + 1 = 2$. An additional path of length 2 can be packed into $T(v)$, because an additional path of length 1 can be packed into $T(u)$.

The special case $L(v, P) = 0$ means that we can pack all P-paths into $T(v)$, but however we pack them there is no space left to pack another path that ends in v.

If it is not possible to pack all P-paths into $T(v)$ at all then let $L(v, P) = -\infty$.

It is quite clear that having computed all tables enables us to find out whether (T, P) is a yes-instance of the path packing-problem. Simply check whether $L(r, \{1, \dots, k\}) \neq -\infty$.

To compute the tables for all v we distinguish three cases how to compose trees into bigger trees: **1** v is a leaf, **2** v has one child, **3** v has at least two children.

Leaf. If v is a leaf then $L(v, \emptyset) = 0$ and $L(v, P) = -\infty$ if $P \neq \emptyset$ because we cannot pack any path into an empty tree (that has no edges).

One child. If v has one child u then it is also quite easy to compute $L(v, P)$: If $L(u, P) = r$ with $r \geq 0$, then clearly $L(v, P) = r + 1$. The right hand side of Figure 2 shows an example. The more complicated possibility is $L(u, P) = -\infty$, which means that it is completely impossible to pack all P-paths into $T(u)$. It might become possible to pack all P-paths into $T(v)$ by using the additional edge uv. If this is possible, then one path, say the ith one with length l_i, uses the edge uv. Then all paths in $P - \{i\}$ are packed into $T(u)$ and one additional path of length $l_i - 1$ that ends in u. We can check this by verifying that $L(u, P - \{i\}) \geq l_i - 1$ for some $1 \leq i \leq k$ (actually, $L(u, P - \{i\}) \geq l_i$ is impossible, because then all P-paths could be packed into $T(u)$ and $L(u, P) \neq -\infty$). If we find such an i, then all P-paths can be packed into $T(v)$, but only by using the edge uv. This means that no other path can be packed into $T(v)$ that ends in v and therefore $L(v, P) = 0$. See the left hand side of Figure 2 for an example.

$$L(v, P) = \begin{cases} L(u, P) + 1 & \text{if } L(u, P) \geq 0 \\ 0 & \text{if } L(u, P) = -\infty \text{ and } L(u, P - \{i\}) = l_i - 1 \text{ for some } i \in P \\ -\infty & \text{otherwise} \end{cases}$$

Two children. Finally, we assume that v has exactly two children u_1, u_2. In that case we can construct for each of them a new tree by attaching new roots v_1, v_2 to $T(u_1)$ and $T(u_2)$ and computing the L-tables for both of them. To compute the table of v it is sufficient to compute a table for a tree that we get by glueing two trees together by identifying their roots. We just have to glue v_1 to v_2.

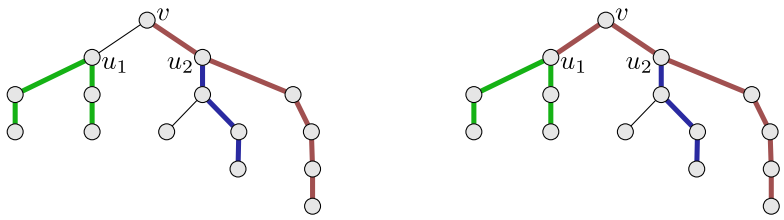


Figure 3 Left side: Packing paths of lengths $l_1 = 5$, $l_2 = 4$, $l_3 = 3$ into $T(v)$. $L(v_1, \{2\}) = 1$ and $L(v_2, \{1, 3\}) = 0$ imply $L(v, \{1, 2, 3\}) \geq 1$.
Right side: Now $l_1 = 6$, $l_2 = 4$, $l_3 = 3$. $L(v_1, \{2\}) + L(v_2, \{3\}) = 1 + 5 = l_1$ implies $L(v, \{1, 2, 3\}) \geq 0$. This time we partition $\{1, 2, 3\}$ into three parts. One goes to the left, one to the right, and one path uses both subtrees.

So we can assume that we have two trees with roots v_1 and v_2 and a tree with root v that we get by identifying v_1 and v_2 and renaming it to v. This is often called a join operation. We have the tables for v_1 and v_2 and want to compute the table for v.

Clearly, $L(v, P) = r$ with $r > 0$ iff some of the P-paths can be packed into $T(v_1)$ and the others into $T(v_2)$ and the additional path with length r that ends in v can be packed into $T(v_1)$ or $T(v_2)$. The additional path of length r that ends in v prevents any P-path from being packed partially into $T(v_1)$ and $T(v_2)$. That is the case iff there is a bipartition of P into P_1 and P_2 such that $L(v_1, P_1), L(v_2, P_2) \geq 0$ and $\max\{L(v_1, P_1), L(v_2, P_2)\} = r$. There are $2^{|P|}$ many subsets of P. To check all bipartitions for all these subsets $P \subseteq [k]$ means looking at $\sum_{i=0}^{k} \binom{k}{i} 2^i = 3^k$ many cases. Using fast subset convolution lets us speed up the computation to $2^k k^{O(1)}$ steps: Let

$$f_i(S) = \begin{cases} 1 & \text{if } L(v_i, S) \geq 0 \\ 0 & \text{otherwise} \end{cases} \qquad g_i(S) = \begin{cases} 1 & \text{if } L(v_i, S) \geq r \\ 0 & \text{otherwise.} \end{cases}$$

Then $L(v, P) \geq r$ iff $(f_1 * g_2)(P) + (g_1 * f_2)(P) \geq 1$.

The situation is different if $L(v, P) = 0$. In that case both edges u_1v and u_2v have to be used when packing all P-paths into $T(v)$ because otherwise at least a path of length one that ends in v could additionally be packed into $T(v)$.

In such a packing one path, say the ith one with length l_i, uses u_1v and u_2v. That is possible iff there is a bipartition of $P - \{i\}$ into P_1 and P_2 such that $L(v_1, P_1) + L(v_2, P_2) \geq l_i$. Again, by using fast subset convolution we can check this in $2^k k^{O(1)}$ steps. ◀

The above proof does not work any more if we glue together two trees whose roots have degree higher than one. For general trees the dynamic programming is much more complicated and we will need more complicated tables.

Let $T(v)$ be a rooted tree with root v and no restrictions on the degree of vertices (and thus on the number of children). Let us again fix length $l_1, \ldots, l_k \in \mathbf{N}$ of paths that we are going to pack into a tree. We are going to identify a set of paths by a set $P \subseteq [k]$. We speak of P-paths as the paths with length l_i for every $i \in P$.

▶ **Definition 4.** Let us fix $l_1, \ldots, l_k \in \mathbf{N}$, $P \subseteq [k]$, and T be a rooted tree. $T(v)$ is the subtree of T with root v.

1. Let $M, M' \subseteq \mathbf{N}$ be two multisets. We say that $M' \succcurlyeq M$ if we can construct M' from M by adding numbers and increasing numbers that are already in M'.
   Let $M' \succ M$ iff $M' \neq M$ and $M' \succcurlyeq M$.
   Example: $\{3, 3, 5, 5, 7\} \succcurlyeq \{2, 3, 4, 6\}$, but $\{3, 3, 5, 5, 7\} \not\succcurlyeq \{2, 3, 4, 8\}$.

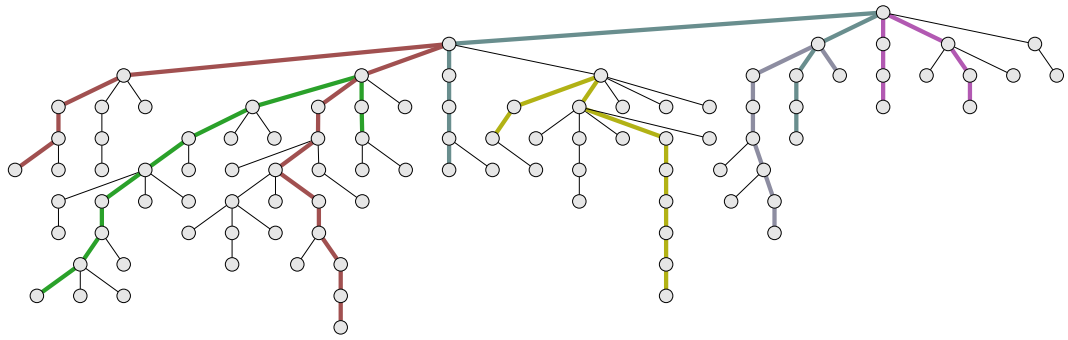


Figure 4 In this tree there are nodes with more than two children and paths can "cross." We pack paths with lengths $13, 9, 9, 9, 7, 6$. There is no solution if we replace 6 by 7.

2. Let $\mathcal{S}$ be a set of multisets of natural numbers. Then

$$K(\mathcal{S}) = \{\, M \in \mathcal{S} \mid \text{ there is no } M' \in \mathcal{S} \text{ with } M' \succ M \,\}.$$

3. Then we define $\mathcal{L}(v, P)$ as a set of multisets of natural numbers as follows:
Let $M \subseteq \mathbf{N}$ be a multiset of natural numbers. Then $M \in \mathcal{L}(v, P)$ iff it is possible to pack all P-paths into $T(v)$ such that we can pack additionally all non-empty paths into $T(v)$ that start at v and have lengths given in M and if there is no $M' \in \mathcal{L}(v, P)$ with $M' \succcurlyeq M$.
Particularly, $\mathcal{L}(v, P) = \emptyset$ iff it is impossible to pack all P-paths into $T(v)$ and $\mathcal{L}(v, P) = \{\emptyset\}$ iff it is possible to pack all P-paths into $T(v)$, but there is no possibility to additionally pack a non-empty path that starts at v.
4. If $M \subseteq \mathbf{N}$ then $\max_q(M)$ is the multiset that consists of the q biggest elements in M or of all of them if M contains less than q numbers, e.g., $\max_3(\{5, 5, 4, 4, 3, 2, 1\}) = \{5, 5, 4\}$.

In the following let $l_1, \ldots, l_k$ be fixed.

▶ **Lemma 5.** *Let $T(v)$ be a rooted tree with root v such that v has one child u.*

1. *Assume that $\mathcal{L}(u, P) = \{M_1, \ldots, M_m\}$ with $m \geq 1$. Define $\mathcal{L}_{\max}(u, P) = \max(M_1 \cup \cdots \cup M_m)$ (where $\max \emptyset = 0$).*
Then $\mathcal{L}(v, P) = \{\{\mathcal{L}_{\max}(u, P) + 1\}\}$.
2. *Assume that $\mathcal{L}(u, P) = \emptyset$ and there is an $i \in \{1, \ldots, k\}$ with $\mathcal{L}_{\max}(u, P - \{i\}) = l_i - 1$.*
Then $\mathcal{L}(v, P) = \{\emptyset\}$.
3. *Otherwise $\mathcal{L}(v, P) = \emptyset$.*

Proof. We have to consider exactly two cases. The first case is that it is possible to pack all P-paths into $T(u)$. If this is the case, then an additional path of length $r + 1$ can be packed into $T(v)$ starting at v iff an additional path of length r can be packed into $T(u)$ starting at u. The latter is the case iff $\mathcal{L}_{\max}(u, P) = r$.

The second case is that it is impossible to pack all P-paths into $T(u)$ alone. It might still be possible to pack them into $T(v)$, but only if the edge uv is used. This means that there is only space for an additional path of length zero that starts at v.

In fact, exactly one path, say the ith one, uses the edge uv. This is possible iff we can pack all $(P - \{i\})$-paths into $T(u)$ and being able to additionally pack a path of length at least $l_i - 1$ into $T(u)$ starting at u. Actually, this path cannot be longer than $l_i - 1$ because then we would be able to pack all P-paths, which is a contradiction. ◀

▶ **Lemma 6.** *Let $T(v_1)$ and $T(v_2)$ be two rooted trees with no common vertices, such that v_2 has exactly one child. Let $T(v)$ be the tree that we get by identifying v_1 with v_2 and renaming it to v. Then $\mathcal{L}(v,P) = K(\mathcal{L}_1 \cup \mathcal{L}_2)$ where*

$$\mathcal{L}_1 = \bigcup_{\substack{P_1 \subseteq P \\ P_2 = P - P_1}} \bigcup_{\substack{M_1 \in \mathcal{L}(v_1, P_1) \\ M_2 \in \mathcal{L}(v_2, P_2)}} \{M_1 \cup M_2\}$$

$$\mathcal{L}_2 = \bigcup_{\substack{P_1 \subseteq P \\ P_2 = P - P_1}} \bigcup_{i \in P} \bigcup_{\substack{M_1 \in \mathcal{L}(v_1, P_1 - \{i\}) \\ M_2 \in \mathcal{L}(v_2, P_2 - \{i\})}} \bigcup_{\substack{r_1 \in M_1 \\ r_2 \in M_2 \\ r_1 + r_2 \geq l_i}} \{(M_1 - \{r_1\}) \cup (M_2 - \{r_2\})\}$$

Proof. "$\mathcal{L}(v,P) \supseteq K(\mathcal{L}_1 \cup \mathcal{L}_2)$": If $M \in \mathcal{L}_1 \cup \mathcal{L}_2$ then $M \in \mathcal{L}_1$ or $M \in \mathcal{L}_2$. Let us first consider the case $M \in \mathcal{L}_1$. By the definition of $\mathcal{L}_1$ there are $P_1 \subseteq P$, $P_2 = P - P_1$, $M_1 \in \mathcal{L}(v_1, P_1)$, and $M_2 \in \mathcal{L}(v_2, P_2)$ such that $M = M_1 \cup M_2$. By induction we know that P_1 can be packed into $T(v_1)$ as well as additional paths of lengths M_1 starting at v_1. The same holds for P_2, v_2, and M_2. Using this packing we actually packed P into $T(v)$ and additional paths of lengths $M_1 \cup M_2 = M$ starting at v. By definition then $M \in \mathcal{L}(v,P)$.

The other possibility is $M \in \mathcal{L}_2$, which is a bit more complicated. If $M \in \mathcal{L}_2$, then $M = (M_1 - \{r_1\}) \cup (M_2 - \{r_2\})$, where $r_1 \in M_1$, $r_2 \in M_2$, $r_1 + r_2 \geq l_i$, $M_1 \in \mathcal{L}(v_1, P_1 - \{i\})$, $M_2 \in \mathcal{L}(v_2, P_2 - \{i\})$, $P_1 \subseteq P$, $P_2 = P - P_1$, and $i \in P$.

We have to show that it is possible to pack P into $T(v)$ and additionally paths with lengths from M starting at v. By induction we know that we can pack all $(P_1 - \{i\})$-paths into $T(v_1)$ and all $(P_2 - \{i\})$-paths into $T(v_2)$. Simultaneously, we can pack additional paths with lengths from M_1 into $T(v_1)$ starting at v_1 and paths with lengths from M_2 into $T(v_2)$ starting at v_2. Hence, we can pack paths with lengths $M = (M_1 - \{r_1\}) \cup (M_2 - \{r_2\})$ into $T(v)$ leaving space for a path of length r_1 in $T(v_1)$ and a path of length r_2 in $T(v_2)$. We can combine these two paths into one path of length $r_1 + r_2 \geq l_i$ and pack one additional path of length l_i into $T(v)$. Altogether we packed P_1, P_2, $\{i\}$ and therefore all P-paths into $T(v)$.

"$\mathcal{L}(v,P) \subseteq K(\mathcal{L}_1 \cup \mathcal{L}_2)$": Let $M \in \mathcal{L}(v,P)$. Then P can be packed into $T(v)$. There are two possibilities:

1. No path corresponding to $i \in P$ lies partially in $T(v_1)$ and partially in $T(v_2)$. Then we can split $P = P_1 \cup P_2$ such that P_1-paths are packed into $T(v_1)$ and P_2-paths into $T(v_2)$. The additional path with lengths from M are also packed into $T(v_1)$ and $T(v_2)$. Let us say $M = M_1 \cup M_2$, where M_1 is in $T(v_1)$ and M_2 in $T(v_2)$. Then it is easy to see that $M \in \mathcal{L}_1$.

2. There is an $i \in P$ such that all $(P - \{i\})$-paths are packed into $T(v_1)$ and $T(v_2)$, but exactly one path with length l_i is packed into $T(v)$ using edges from both $T(v_1)$ and $T(v_2)$. Note that there can be at most one such path because v_2 has only one child in $T(v_2)$. Then all additional paths with lengths in M that start at v have to be packed into $T(v_1)$ alone because the edge in $T(v_2)$ is not available any more. Let r_1 be the length of the part of the bridging path of length l_i that lies in $T(v_1)$ and r_2 the length of the part in $T(v_2)$. Clearly, $r_1 + r_2 = l_i$. With all these facts we can again easily verify that $M \in \mathcal{L}_2$. ◀

The following lemma shows that the size of the tables is bounded by a function in k and the maximal degree. The estimate is quite pessimistic, but we are not trying to optimize the runtime of the dynamic programming algorithm at the moment and are content with proving fixed parameter tractability.

▶ **Lemma 7.** *Let $T(v)$ be a rooted tree and assume that vertex v has d children. Then*

$$|\mathcal{L}(v,P)| \leq d2^{kd}.$$

Proof. If v has only one child, then $|\mathcal{L}(v,P)| = 1$ and the statement is true. Assume next that $T(v)$ has d children. Each subtree can receive at most 2^k different sets of packed paths yielding at most 2^k different length of the longest path that can be additionally packed. Therefore a set $M \in \mathcal{L}(v,P)$ can have size at most d and contain up to d numbers each chosen from a set of size at most 2^k. In total that are at most $d2^{kd}$ possibilities for a set M. ◀

▶ **Theorem 8.** *Let T be a rooted tree and P a multiset of paths. In polynomial time a rooted tree T' can be computed that has the following properties:*
(1) *P can be packed into T iff it can be packed into T',*
(2) *each node in T' has at most $3|P|$ children, and (3) T' is a subtree of T.*

Proof. Let $l_1(u)$ be the length of the longest path in $T(u)$ that starts in u and $l_2(u)$ be the length of the longest path in $T(u)$. Assume that P can be packed into T and v be an arbitrary vertex in T. Let us fix an edge-disjoint packing of P.

Let v be an arbitrary node in T and $v_1, \ldots, v_m$ the children of v. Let us further assume that $v_1, \ldots, v_{3|P|}$ contain the $|P|$ children with biggest $l_1(v_i)$ and $2|P|$ children with biggest $l_2(v_i)$. Ties can be arbitrarily ordered.

If $m \leq 3|P|$ we do nothing. Otherwise assume that P is packed into T and some path $p \in P$ uses $T(v_i)$ with $i > 3|P|$. There are two possibilities:

(i) p contains v_i. Then p is possibly packed partially inside $T(v_i)$ and partially outside. Let p' be the part of p inside $T(v_i)$. Clearly, p' starts at v_i. By the pigeonhole principle there must be some $T(v_k)$ that has not been used in the packing of P, $l_1(v_k) \geq l_1(v_i)$, and $k \leq 3|P|$. Then we can repack p such that it uses $T(v_k)$ instead of $T(v_i)$.

(ii) p does not contain v_i and is therefore completely packed into $T(v_i)$. Again by the pigeonhole principle we can find an appropriate $T(v_k)$ with $k \leq 3|P|$ and $l_2(v_k) \geq l_2(v_i)$. We can repack p from $T(v_i)$ into $T(v_k)$.

Repeated repacking in these two ways leads to a packing that uses only the subtrees $T(v_1), \ldots, T(v_{3|P|})$. We can therefore remove all other subtrees without changing a yes-instance into a no-instance. Applying this pruning to all vertices in T leads to a new tree T' that has all properties stated in the theorem. It is also clear that T' can be computed in polynomial time as it is easy to find longest paths in trees. ◀

Combining the above results (with the base case for a leaf v: $\mathcal{L}(v,P) = \{\{0\}\}$ if P contains only empty paths and $\mathcal{L}(v,P) = \emptyset$ otherwise) we can prove the following:

▶ **Theorem 9.** Path Packing *into forests parameterized by the number of paths is in* FPT.

Proof. Given a tree T compute a rooted tree T' where each node has at most $3k$ children and every P (with $|P| = k$) can be packed into T iff it can be packed into T' (Theorem 8). Then use dynamic programming to find out whether the paths can be packed into T'. By Lemma 7 and 6 this only takes time $f(k)|T|^{O(1)}$ for some function f. ◀

Lower bound

While Path Packing on graphs with treewidth one is in FPT when parameterized by the number of paths, we now show that the problem becomes hard on graphs with treewidth two. As an intermediate step, we reduce from Unary Bin Packing [15] to show hardness of Multi-Way Number Partition. This then leads to hardness results for Exact Path Packing. Remember that for Multi-Way Number Partition the numbers are unary encoded.

▶ **Lemma 10.** Multi-Way Number Partition *parameterized by the number of sets k is $W[1]$-hard. Moreover, unless ETH fails there is no algorithm that solves the problem in $f(k)N^{o(k/\log k)}$ time for some function f where N is the input size.*

▶ **Theorem 11.** *The* Exact Path Packing *problem parameterized by the number of paths on graphs with treewidth two is $W[1]$-hard. Moreover, unless ETH fails there is no algorithm that solves the problem in $f(k)\ n^{o(k/\log k)}$ time for some function f where k is the number of paths and n the number of vertices in the input graph.*

4 Path Packing Parametrized by Path Dependent Attributes

In the previous section we solved Path Packing on forests. Since Path Packing is NP-hard even for graphs with treewidth 2, we try to find some path dependent parameters to cope with its difficulty. At first, we will restrict the number of paths, then we will bound the length of each path and finally we consider the sum of the lengths of all paths.

Number of Paths

We denote the number of paths of an instance by k. We start with $k = 1$. Consider an instance where the length of the single path corresponds to the number of vertices in a complete graph G.

▶ **Observation 12.** *Since Hamiltonian Path is* NP*-hard, also path packing for $k = 1$ is* NP*-hard*

On the other side, for $k = 1$ the special case of Exact Path Packing becomes easy.

▶ **Observation 13.** Exact Path Packing *is solvable in polynomial time for $k = 1$ by deciding if the input graph is a path of length l_1.*

Unfortunately, for fixed $k \geq 2$ restricting the number of paths is not enough to gain a polynomial time algorithm. This holds for Exact Path Packing and therefore also for Path Packing.

▶ **Theorem 14.** *Let $k \geq 2$.* Exact Path Packing *with k paths is* NP*-complete on 4-regular graphs.*

Paths with bounded length

Observe that all hardness proofs that we have seen so far somehow involve paths of a certain length. Thus, we analyze the complexity of Path Packing based on the length of the paths we want to pack.

Length-i Exact Edge Packing

Input: A set of paths $P = \{p_1, \ldots, p_k\}$ of length $l_1 = \ldots = l_k = i$, a graph $G = (V, E)$.

Question: Is there an edge-disjoint embedding of P into G such that each edge $e \in E$ is covered exactly once?

Length-2 Exact Edge Packing is solvable in polynomial time by reformulating it as a matching problem on the line graph. We show that Length-3 Exact Edge Packing is already NP-hard via a reduction from the 3-dimensional matching problem, one of Karp's original 21 NP-complete problems. The 3-dimensional matching problem takes as input sets X, Y, Z of size n and $T \subseteq X \times Y \times Z$. The question is whether there exists a set $M \subseteq T$ of size n such that for all $(x_1, y_1, z_1), (x_2, y_2, z_2) \in M$, $x_1 \neq x_2$, $y_1 \neq y_2$, $z_1 \neq z_2$. The reduction is similar to the P_2-packing reduction in [20].

▶ **Theorem 15.** Length-3 Exact Edge Packing *is* NP*-hard.*

Bounded sum of path lengths

The two previous results show that Path Packing is NP-hard even if the number of paths or the maximal length of the paths is bounded by a constant. At last, we the problem is in FPT when parameterized by the number of paths *and* their length. We give a randomized FPT algorithm using color coding [8, 1] that can easily be derandomized using perfect hash families [8].

▶ **Theorem 16.** Path Packing *parameterized by the summed length of all paths is in* FPT.

For packing vertex-disjoint paths similar results are known: The P_2-packing problem takes a graph G and an integer k and asks if there is a set of k vertex-disjoint P_2 in G. This problem is NP-complete [16, 20]. Fernau and Raible given FPT algorithm parameterized by the number of paths [10].

5 Path Packing Parametrized by Graph Dependent Attributes

Earlier (Theorem 1), we showed that Exact Path Packing is NP-hard even on a single subdivided star. So even for trees where there is only one node of degree higher than two the problem becomes NP-hard. In this section we study further restrictions to forests and finally identify a polynomial time solvable case. We do so by considering restrictions to the following three parameters: number of vertices of degree at least three, the maximal degree, and the number of connected components. For an easier notation we define this combined parameter as the "bcd" of graph G. It is a bound on **b**ranching nodes, connected **c**omponents and maximum **d**egree.

▶ **Definition 17.** *Let G be a graph. Then bcd(G) is the minimal $k \in \mathbf{N}$ such that G has at most k nodes of degree larger than two, at most k connected components, and a maximal degree of at most k.*

The above mentioned reduction showing NP-hardness for a subdivided star constructs a graph with unbounded degree. What is the complexity if we limit the vertex degree to a constant, but in turn allow multiple components? Unfortunately even for a forest of paths, thus a maximum degree of two, the problem remains NP-hard. NP-hardness follows by an easy adaption of the proof of Theorem 1. The constructed subdivided star has an even number $2m$ of legs of length ℓ. Instead one could also use m disjoint paths of length 2ℓ. Thus we can follow NP-hardness of Exact Path Packing even for forests of paths.

▶ **Corollary 18.** Exact Path Packing *is* NP*-hard even on forests of paths.*

Thus, if we drop either the degree or the number of components as parameters, the problem becomes NP-complete, even if the remaining parameters are bounded by a constant. Thus the remaining question is: What is the complexity if we limit the vertex degree to a constant, limit the number of connected components to a constant, but in turn allow arbitrary many vertices of degree at least three? We show hardness in this scenario even for the more restricted problem of packing paths of ascending length, denoted by Path-Perfect Packing.

Path-Perfect Packing

Input:	A graph $G = (V, E)$ where $\lvert E(G)\rvert = 1 + 2 + \cdots + n$ for some $n \in \mathbf{N}$.
Question:	Does there exist an edge-disjoint embedding of paths $p_1, \ldots, p_n$ with lengths $\ell_1 = 1, \ldots, \ell_n = n$ into G such that each edge $e \in E$ is covered exactly once?

We show NP-hardness of this restricted problem on subdivisions of a caterpillar with vertex degree at most eight. We reduce from the following unary version of 3-partition. This version is slightly non-standard since we require that no numbers occur twice and relax the condition to put exactly three elements into each partition.

Unary 3-Partition
Input: A set of integers $A = \{a_1, \ldots, a_{3s}\} \subseteq \mathbf{N}$ in unary encoding.
Question: Is there a partition of A into s sets of equal sum?

Hulett et al. show that the above problem is NP-hard if we require each of the s partitions to contain exactly three elements [14]. We get NP-hardness without the extra condition by increasing each number in A by adding a big number (for example $\sum_{i=}^{3s} a_i$).

We sketch how to reduce from Path-Perfect Packing on caterpillars with vertex degree at most eight to Unary 3-Partition. Note that, each partition of a Unary 3-Partition instance must have size $m = \frac{1}{s}\sum_{i=1}^{3s} a_i$. Consider the paths $\{p_i \mid 1 \leq i \leq m,\ i \in A\}$ whose length occurs in A. We translate a partition of A into an exact packing of these paths. However, we have to account for the paths $\{p_i \mid 1 \leq i \leq m,\ i \notin A\}$ whose length occurs *not* as an integer in set A. For each of these we introduce a path where it fits in precisely, and by an exchange argument we may assume it is packed there. Now, roughly what remains to do is to construct a large caterpillar of low maximum degree where all these paths can be packed in.

▶ **Theorem 19.** Path-Perfect Packing *is* NP*-hard even for subdivided caterpillars with vertex degree at most eight.*

The maximum degree eight in the theorem above was chosen to simplify the proof. One can show NP-hardness even for smaller maximum vertex degree.

XP-algorithm for parameter bcd(G)

We saw that if two bcd-parameters are constant and one bcd-parameter is unbounded then Exact Path Packing is NP-complete. We further study the complexity when parameterized by all three parameters. We give an XP-algorithm for Path Packing parametrized by $\mathrm{bcd}(G)$. This means for every fixed k, there is a polynomial time algorithm for graphs with $\mathrm{bcd}(G) \leq k$.

▶ **Theorem 20.** *There is a* $k!^k(n+k^2)^{O(k^2)}$*-time algorithm for* Path Packing *with* $k = bcd(G)$.

Proof. We give an algorithm, that given a graph G and a list P of paths $p_1, \ldots, p_k$, decides if there is an edge-disjoint embedding of $p_1, \ldots, p_k$ into G. To do so, we guess a partition of G into eventually a set of vertex-disjoint paths $\mathcal{X}$. Then it suffices to find an embedding of P into such a set of vertex-disjoint paths $\mathcal{X}$. The remaining problem then is just a generalized bin-packing problem with $O(k^2)$ bins, but encoded in unary; thus solvable in time $n^{O(k^2)}$. Most technicality lies in guessing the vertex-disjoint paths $\mathcal{X}$. First we guess a partition into a bounded number of walks $\mathcal{W}$. Later we need to partition $\mathcal{W}$ further resulting in vertex-disjoint paths $\mathcal{X}$.

Let V_1 be the set of vertices of degree two. Let $V_2^\star$ consist of a vertex of every connected component that is a circle. Let V_2 be the set of vertices of degree two that are not in $V_2^\star$, and let $V_{\geq 3}$ be the vertices of degree at least three including $V_2^\star$. This seemingly odd definition allows us to work with walks starting and ending in $V_1 \cup V_{\geq 3}$ that cover every edge, in

particular those in a circle. Because there are at most k connected components, $|V_2^\star| \leq k$. Then since there are at most k vertices of degree at least three, we have $|V_{\geq 3}| \leq 2k$.

Assuming a yes-instance, there is an edge-disjoint embedding of paths P into graph G. At every vertex $v \in V_{\geq 3}$ every path of P contains at most two of the incident edges of v. Thus at every vertex $v \in V_{\geq 3}$ there is a maximal matching M_v of v's incident edges such that no path in P contains two unmatched edges.

We consider "direct" walks between "neighboring" vertices $V_1 \cup V_{\geq 3}$: Let $\mathcal{Q}$ be the set of walks between $u, v \in V_1 \cup V_{\geq 3}$ with inner vertices from V_2, and further where no vertex among V_2 is repeated (though possibly $u = v$). We join these walks $\mathcal{Q}$ to a set of walks $\mathcal{W}$ according to matchings M_v for $v \in V(G)$. Whenever two walks w_1, w_2 end at some edges uv respectively $u'v$, and uv is matched to $u'v$ by M_v, then join walks w_1 and w_2 at edges uv, vu'. This procedure terminates and yields a well defined set of walks $\mathcal{W}$.

Note that every edge is covered by a walk $\mathcal{Q}$ and thus also every edge is covered by a walk $\mathcal{W}$. We further claim that every path p of P is a subsequence of edges of some walk $w \in \mathcal{W}$. Assuming otherwise, there are walks w_1, w_2 ending at edges uv and $u'v$. Then v is not a leaf, and thus $v \in V_{\geq 3}$. Then matching M_v matches edges uv and $u'v$, and thus w_1, w_2 had to be joined to a single walk.

Thus for a yes-instance there is at least one set of matchings $M_v, v \in V_{\geq 3}$ which determines walks $\mathcal{W}$ such that P may be embedded into $\mathcal{W}$. An algorithm may try the possible partition of edges into such a set of walks $\mathcal{W}$ as follows. Guess for each vertex $v \in V_{\geq 3}$ a maximal matching M_v of its incident edges. There are at most k high degree vertices $V_{\geq 3} \setminus V_2^\star$, each with at most k incident edges. (Also we have a matching for $V_2^\star$, though since there are only two incident edges, there is only one possible matching.) Thus the algorithm tries at most $k!^k$ possibilities. Then combine the paths $\mathcal{Q}$ to walks $\mathcal{W}$ according to the matchings, which is possible in polynomial time.

We claim that $\mathcal{W}$ has at most k^2 walks. Every walk in $\mathcal{W}$ has two endpoints, and the endpoints are among $V_1 \cup V_{\geq 3}$. Clearly, at every leaf $v \in V_1$ at most one path ends. Further, there are at most k^2 leaves in the input graph of $\leq k$ vertices of degree ≥ 3 and maximal degree of k. If at a vertex $v \in V_{\geq 3}$ two walks w_1, w_2 end, there are edges uv of w_1 and $u'v$ of w_2 unmatched by M_v, in contradiction to a maximal matching M_v. Thus also at every vertex $v \in V_{\geq 3}$ at most one walk ends. Then there are at most $k^2 + 2k$ endpoints of walks, and thus there are at most $\lfloor (k^2 + 2k)/2 \rfloor \leq k^2$ walks in $\mathcal{W}$.

Consider a walk $w \in \mathcal{W}$ where a vertex v occurs more than once. Recall that the embedding of a path $p \in P$ of a yes-instance is injective, thus no vertex $v \in V(G)$ occurs twice in the same path. A naive approach would be to now solve the bin-packing problem of "weights" P and "bins" $\mathcal{W}$. Then, however, a solution to the bin-packing would may potentially translate to an embedding of a path where a vertex occurs twice. Therefore let us guess a partition into paths without multiple occurrence of vertices, as follows.

Between two occurrences of v on walk w there must be vertex u (possibly an occurrence of v itself) which is the endpoint of two different paths. Therefore there is a partition of the walks $\mathcal{W}$ into vertex-disjoint paths $\mathcal{X}$, where still paths P have an embedding into $\mathcal{X}$. We may describe this partition by "cuts" of $\mathcal{W}$ specified by a vertex v in the union of walks from $\mathcal{W}$. Note, that in the union of walks $\mathcal{W}$, each high degree vertex $v \in V_{\geq 3} \setminus V_2^\star$ occurs $\deg(v) \leq k$ times. Thus there are to up to $n + k^2$ potential cut vertices.

We claim that at most k^2 cuts C are necessary to cut the walks $\mathcal{W}$ into vertex-disjoint paths $\mathcal{X}$. Assume, that there is a cut vertex $v \in C$ which is on an inner vertex of a path between V_1 and $V_{\geq 3}$. Then joining its incident vertex-disjoint paths results in a vertex-disjoint path. Thus we may assume that the cuts C are at vertices from walks of $\mathcal{Q}$ between vertices

among $V_{\geq 3}$. Let $\mathcal{Q}_{\geq 3}$ be the set of paths $\mathcal{Q}$ with endpoints in $V_{\geq 3}$. Consider the multi-graph with loops on vertex set $V_{\geq 3}$ with an edge between $u, v \in V_{\geq 3}$ for every path $\mathcal{Q}_{\geq 3}$ with endpoints u and v. Since $|Q_{\geq 3}| \leq k$ and the degree of every vertex $v \in Q_{\geq 3}$ is at most k, this multi-graph has at most k^2 edges. Then also there are at most k^2 paths $\mathcal{Q}_{\geq 3}$. Consider a set of more than k^2 vertices $C \subseteq V$ that cut $\mathcal{Q}$ into vertex-disjoint paths $\mathcal{X}$. Then there is a path of $Q_{\geq 3}$ containing distinct cut vertices $u, v \in C$. Let $x \in \mathcal{X}$ be the path between u and v. Joining them with the incident path at, say u, results in a vertex-disjoint path. Thus cutting $\mathcal{W}$ at vertices $C \setminus \{u\}$ still results in a set of vertex-disjoint paths. Therefore at most k^2 cuts of walks $\mathcal{W}$ are necessary to yield vertex-disjoint paths $\mathcal{X}$.

Let us utilize this observation in the design of our algorithm. Guess up to k^2 cuts C from the $n + k^2$ potential cuts. Cut the previously guessed walks $\mathcal{W}$ into subpaths according to cuts C. We may force exactly k^2 cut vertices by allowing C to be a multi-set containing also leaves, whose cut has no effect. This way, we try another $(n+k^2)^{k^2}$ possibilities of cut vertices C. Then cut the previously guessed $\leq k^2$ walks $\mathcal{X}$ at the k^2 cut positions. If the resulting set of walks is not vertex-disjoint, discard this guess. Otherwise we obtain $\leq 2k^2$ vertex disjoint paths $\mathcal{X}$, since every cut increases the number of paths by one. This resembles a bin-packing problem in unary encoding with k^2 bins of different sizes and total capacity n. We may apply standard dynamic programming technique to test in $n^{O(k^2)}$ time whether the sizes of the paths P fit into the bins in the sizes of $\mathcal{X}$. If the paths P fit in some guessed paths $\mathcal{X}$, then corresponding partition of the edges in G yields paths P. Thus there is an edge-disjoint embedding of P into G. For the other direction, if the edges of G can be partitioned into paths P, then as argued before there is a set $\mathcal{X}$ according to this partition and the there is a solution to the dynamic problem. The runtime is $k!^k(n + k^2)^{O(k^2)} \cdot \text{poly}(n) = k!^k n^{O(k^2)} \cdot \text{poly}(n)$ where poly is a polynomial. ◀

Can we achieve a better runtime than $k!^k n^{k^2+O(1)}$, in particular decrease the dependence on k in the exponent of n? Not significantly unless ETH fails, as the following reduction from Multi-Way Number Partition shows.

▶ **Theorem 21.** *There is no algorithm that decides* Path Packing *in time* $n^{o(k^2/\log k)}$ *with* $k = bcd(G)$ *unless ETH fails.*

6 Conclusion

We showed that edge-disjoint packing of paths into a graph is a very hard problem. Even if the input graph is a subdivided star or a linear forest the problem is hard. If we parameterize the problem by the number of paths, the problem remains hard even for input graphs with treewidth two. However, it becomes fixed parameter tractable on forests. A natural open problem is to not embed paths, but more general graphs such as trees or cycles.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995. doi:10.1145/210332.210337.

2 S. Rao Arikati and C. Pandu Rangan. Linear Algorithm for Optimal Path Cover Problem on Interval Graphs. *Inf. Process. Lett.*, 35(3):149–153, July 1990. doi:10.1016/0020-0190(90)90064-5.

3 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, San Diego, California, USA, June 11-13, 2007*, pages 67–74, 2007. doi:10.1145/1250790.1250801.

4 Maurizio A. Bonuccelli and Daniel P. Bovet. Minimum Node Disjoint Path Covering for Circular-Arc Graphs. *Inf. Process. Lett.*, 8(4):159–161, 1979. doi:10.1016/0020-0190(79)90011-5.

5 Weiting Cao and Peter Hamburger. Solution of fink & straight conjecture on path-perfect complete bipartite graphs. *Journal of Graph Theory*, 55(2):91–111, 2007.

6 Gerard J. Chang and David Kuo. The $L(2,1)$-Labeling Problem on Graphs. *SIAM Journal on Discrete Mathematics*, 9(2):309–316, 1996. URL: https://epubs.siam.org/doi/ref/10.1137/S0895480193245339.

7 Gerard Cornuéjols, David Hartvigsen, and W Pulleyblank. Packing subgraphs in a graph. *Operations Research Letters*, 1(4):139–143, 1982.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

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

10 Henning Fernau and Daniel Raible. A parameterized perspective on packing paths of length two. In *International Conference on Combinatorial Optimization and Applications*, pages 54–63. Springer, 2008.

11 John Frederick Fink and H. Joseph Straight. A note on path-perfect graphs. *Discrete Mathematics*, 33(1):95–98, 1981.

12 S. E. Goodman, Stephen T. Hedetniemi, and Peter J. Slater. Advances on the Hamiltonian Completion Problem. *J. ACM*, 22(3):352–360, 1975. doi:10.1145/321892.321897.

13 Peter Hamburger and Weiting Cao. Edge Disjoint Paths of Increasing Order in Complete Bipartite Graphs. *Electronic Notes in Discrete Mathematics*, 22:61–67, 2005.

14 Heather Hulett, Todd G. Will, and Gerhard J. Woeginger. Multigraph realizations of degree sequences: Maximization is easy, minimization is hard. *Operations Research Letters*, 36(5):594–596, 2008. doi:10.1016/j.orl.2008.05.004.

15 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1):39–49, 2013.

16 David G. Kirkpatrick and Pavol Hell. On the Completeness of a Generalized Matching Problem. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 240–245, 1978. doi:10.1145/800133.804353.

17 Hoàng-Oanh Le, Van Bang Le, and Haiko Müller. Splitting a graph into disjoint induced paths or cycles. *Discrete Applied Mathematics*, 131(1):199–212, 2003. doi:10.1016/S0166-218X(02)00425-0.

18 Jayadev Misra and Robert Endre Tarjan. Optimal Chain Partitions of Trees. *Inf. Process. Lett.*, 4(1):24–26, 1975. doi:10.1016/0020-0190(75)90057-5.

19 Jerome Monnot and Sophie Toulouse. The path partition problem and related problems in bipartite graphs. *Operations Research Letters*, 35(5):677–684, 2007. doi:10.1016/j.orl.2006.12.004.

20 Sarah Pantel. Graph packing problems. Master's thesis, Simon Fraser University, Canada, 1999.

21 R. Srikant, Ravi Sundaram, Karan Sher Singh, and C. Pandu Rangan. Optimal Path Cover Problem on Block Graphs and Bipartite Permutation Graphs. *Theor. Comput. Sci.*, 115(2):351–357, 1993. doi:10.1016/0304-3975(93)90123-B.

22 George Steiner. On the k-path partition of graphs. *Theoretical Computer Science*, 290(3):2147–2155, 2003. doi:10.1016/S0304-3975(02)00577-7.

23 H. Joseph Straight. *Partitions of the vertex set or edge set of a graph*. dissertation, Western Michigan University, 1977.

24 H. Joseph Straight. Packing trees of different size into the complete graph. *Annals of the New York Academy of Sciences*, 328(1):190–192, 1979.

25 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution. In *Proc. of the 17th*

Annual European Symposium on Algorithms (ESA), number 5757 in Lecture Notes in Computer Science, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

26 Jing-Ho Yan and Gerard J. Chang. The path-partition problem in block graphs. *Information Processing Letters*, 52(6):317–322, 1994. doi:10.1016/0020-0190(94)00158-8.

27 Jing-Ho Yan, Gerard J. Chang, Sandra Mitchell Hedetniemi, and Stephen T. Hedetniemi. k-Path Partitions in Trees. *Discrete Applied Mathematics*, 78(1-3):227–233, 1997. doi:10.1016/S0166-218X(97)00012-7.

28 H. P. Yap. Packing of graphs-a survey. In *Annals of Discrete Mathematics*, volume 38, pages 395–404. Elsevier, 1988.

29 Shmuel Zaks and Chung Laung Liu. Decomposition of graphs into trees. Technical Report 860, Department of Computer Science, University of Illinois at Urbana-Champaign, 1977.

Hardness of FO Model-Checking on Random Graphs

Jan Dreier
Department of Computer Science, RWTH Aachen University, Germany
https://tcs.rwth-aachen.de/~dreier
dreier@cs.rwth-aachen.de

Peter Rossmanith
Department of Computer Science, RWTH Aachen University, Germany
https://tcs.rwth-aachen.de
rossmani@cs.rwth-aachen.de

Abstract

It is known that FO model-checking is fixed-parameter tractable on Erdős–Rényi graphs $G(n, p(n))$ if the edge-probability $p(n)$ is sufficiently small [23] ($p(n) = O(n^\epsilon/n)$ for every $\epsilon > 0$). A natural question to ask is whether this result can be extended to bigger probabilities. We show that for Erdős–Rényi graphs *with vertex colors* the above stated upper bound by Grohe is the best possible.

More specifically, we show that there is no FO model-checking algorithm with average FPT run time on vertex-colored Erdős–Rényi graphs $G(n, n^\delta/n)$ $(0 < \delta < 1)$ unless $\mathrm{AW}[*] \subseteq \mathrm{FPT/poly}$. This might be the first result where parameterized average-case intractability of a natural problem with a natural probability distribution is linked to worst-case complexity assumptions.

We further provide hardness results for FO model-checking on other random graph models, including $G(n, 1/2)$ and Chung-Lu graphs, where our intractability results tightly match known tractability results [13]. We also provide lower bounds on the size of shallow clique minors in certain Erdős–Rényi and Chung–Lu graphs.

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

Keywords and phrases random graphs, FO model-checking

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.11

1 Introduction

Model-checking is an important and well-investigated problem with various applications in database theory, verification, artificial intelligence and many other areas. The input to the model-checking problem is a structure and a logical sentence and the question is whether the structure is a model for the sentence, i.e., if the sentence is true in the model. We consider the FO model-checking problem on colored graphs. This means that sentences can express (besides the common rules of first-order logic) adjacency between vertices and whether a vertex has a given color. This problem is known to be PSPACE-complete [39]. Let $\mathcal{G}$ be a graph class and L be a logic. We are interested in model-checking as a parameterized problem, which is defined as follows:

p-MC$(L, \mathcal{G})$	
Input:	A graph $G \in \mathcal{G}$ and a logical sentence $\varphi \in L$
Parameter:	$\lvert\varphi\rvert$
Problem:	$G \models \varphi$?

We denote the class of all graphs by $\mathfrak{G}$. Downey, Fellows, and Taylor showed that p-MC(FO, $\mathfrak{G}$) is AW[$*$]-complete and thus very hard on general graphs [15]. However, model-checking becomes tractable when restricted to special graph classes. Courcelle's theorem states that if $\mathcal{G}$ is a graph class with bounded treewidth then p-MC(MSO, $\mathcal{G}$) is in FPT [11]. The FO model-checking problem p-MC(FO, $\mathcal{G}$) can be solved in FPT time if $\mathcal{G}$ has bounded expansion [17, 38] or is nowhere-dense [25].

For a graph class $\mathcal{G}$, we define $\mathcal{G}_{col}$ to be the class of all vertex colorings of $\mathcal{G}$. For most graph classes $\mathcal{G}$ it makes no difference if we consider the model-checking problem on $\mathcal{G}$ or $\mathcal{G}_{col}$ because colors can be encoded by small gadgets. Given an instance (G, φ) with $G \in \mathcal{G}_{col}$, for most graph classes $\mathcal{G}$ it is easy to construct a new instance (G', φ') of similar size with $G' \in \mathcal{G}$, and $G \models \varphi$ iff $G' \models \varphi'$. In particular, if G comes from a nowhere-dense class, then so does G'.

Average-Case Model-Checking

Average-case complexity analyzes the typical run time of algorithms on random instances (such as random graphs) and is a well-established field of complexity theory [3]. While the worst-case complexity of the parameterized model-checking problem is well analyzed, much less is known about its average-case complexity, or parameterized average-case complexity in general.

We fix a sequence of vertices $(u_i)_{i\in\mathbf{N}}$ and say a *random graph model* is a sequence $\mathcal{G} = (\mathcal{G}_n)_{n\in\mathbf{N}}$, where $\mathcal{G}_n$ is a probability distribution over all graphs G with $V(G) = \{u_1, \dots, u_n\}$. The most well-known random graph model are so called Erdős–Rényi graphs [6]. The Erdős–Rényi graph $G(n, p(n))$ consists of n vertices where each edge is added independently with probability $p(n)$. This very natural model is of great theoretical interest and has been investigated in many ways. Other models such as the preferential attachment model [2, 5], the Chung–Lu model [10, 9, 7], or the configuration model [32, 33] were defined to mimic complex networks that are observed in the real world.

There exist some tractability results for the model-checking problem on random graph models, including sparse Erdős–Rényi graphs, Chung–Lu graphs and the configuration model, as we will see later on. We distinguish between the average-case complexity of model-checking on uncolored graphs (p-MC(FO, $\mathfrak{G}$)) and colored graphs (p-MC(FO, $\mathfrak{G}_{col}$)):

▶ **Definition 1.** We say p-MC(FO, $\mathfrak{G}$) can be decided on a random graph model $\mathcal{G}$ in *expected time* $f(|\varphi|, n)$ if there exists a deterministic algorithm $\mathcal{A}$ which decides p-MC(FO, $\mathfrak{G}$) on input G, φ in time $t_{\mathcal{A}}(G, \varphi)$ and for all $n \in \mathbf{N}$, all FO-sentences φ, $\mathrm{E}_{G\sim\mathcal{G}_n}\left[t_{\mathcal{A}}(G, \varphi)\right] \leq f(|\varphi|, n)$.

A function $C : \mathfrak{G} \to \mathfrak{G}_{col}$ is called a *c-coloring function* for $c \in \mathbf{N}$ if for every $G \in \mathfrak{G}$, $C(G)$ is a coloring of G with up to c colors. The colorings do not need to be proper.

▶ **Definition 2.** We say p-MC(FO, $\mathfrak{G}_{col}$) can be decided on a random graph model $\mathcal{G}$ in *expected time* $f(|\varphi|, n)$ if there exists a deterministic algorithm $\mathcal{A}$ which decides p-MC(FO, $\mathfrak{G}_{col}$) on input G, φ in time $t_{\mathcal{A}}(G, \varphi)$ and if for all $n \in \mathbf{N}$, all FO-sentences φ and all $|\varphi|$-coloring functions C, $\mathrm{E}_{G\sim\mathcal{G}_n}\left[t_{\mathcal{A}}(C(G), \varphi)\right] \leq f(|\varphi|, n)$.

We say p-MC(FO, $\mathfrak{G}_{col}$) or p-MC(FO, $\mathfrak{G}$) can be decided in *expected FPT time* on a random graph model if it can be decided in expected time $f(|\varphi|)n^{O(1)}$ for some function f. Clearly, p-MC(FO, $\mathfrak{G}_{col}$) is harder than p-MC(FO, $\mathfrak{G}$) on random graph models because the model-checking algorithm has to be efficient for every possible coloring. However, all efficient average-case model-checking algorithms so far work by placing the random graph model

with high probability in a tractable graph class (for example low degree [23], or bounded expansion [13]) and then using well-known model-checking algorithms for such graph classes. Thus, the algorithms work no matter how the vertices of the random graphs are colored.

The following tractability results are known: Let $p(n)$ be a function with $p(n) = O(n^\varepsilon/n)$ for all $\varepsilon > 0$. Grohe showed that one can solve p-MC(FO, $\mathfrak{G}_{col}$) on $G(n, p(n))$ in expected time $f(|\varphi|, \delta)n^{1+\delta}$ for every $\delta > 0$ [23]. Later Demaine et al. proved that p-MC(FO, $\mathfrak{G}_{col}$) can be solved in expected FPT linear time on Chung–Lu and configuration graphs whose degrees follow a power law distribution with exponent $\alpha > 3$ and maximal degree at most $n^{1/\alpha}$ [13].

Average-Case Intractability

A natural question to ask next is: For which graphs does model-checking become intractable? For the worst-case complexity of monotone graph classes (classes which are closed under subgraphs) this question is settled: FO model-checking is in FPT if and only if the graph class is nowhere-dense [25]. For non-monotone classes and especially random graph models the question is more difficult.

Ideally, positive algorithmic results should be accompanied by lower bounds that show that the result is the best possible. Very often such matching lower bounds are the triumphant last step in completely answering a question that has had a long history of incremental partial results. The best example is the theory of NP-completeness, but there are many other nice examples from parameterized complexity: ETH-based lower bounds, non-existence of polynomial kernels, and in the area of FO model-checking the fact that p-MC(FO, $\mathcal{G}$) $\in$ FPT for some monotone somewhere-dense graph class $\mathcal{G}$ implies AW[$*$] = FPT.

On the other hand, the situation does not look as good in the area of average-case complexity. While we know that FO model-checking on $G(n, d/n)$ can be decided in expected FPT time if d is constant or grows slower than n^ϵ for every $\epsilon > 0$, we do not have any lower bounds for a larger d.

The lack of lower bounds of average-case runtimes is not restricted to random graphs and the model-checking problem. At this point of time we still do not know techniques to prove good lower bounds in that area. While it is certainly possible to reduce between problems preserving bounds on the expected running time, there are virtually no results linking average- to worst-case complexity. What we are missing in particular are results of this kind: If we can solve parameterized problem X fast on average then some unexpected consequence holds in the world of worst-case complexity (such as P = NP).

There has been some work on average-case complexity of parameterized problems. Fountoulakis, Friedrich, and Hermelin showed that parameterized clique can be decided efficiently on Erdős–Rényi graphs with arbitrary density [18] and Friedrich and Krohmer proved the same result for certain scale free random graphs where the degree sequence follows a power law with exponent $\alpha > 2$ [19]. However, there exist some artificial, computable distributions for which the problem is distW[1]-complete [18], where distW[1] is an average-complexity class, which is assumed to be hard. This means that parameterized clique cannot be decided efficiently on average on that distribution unless every problem in distW[1] can be decided efficiently on average. This promising result identifies distW[1] as a possible corner stone for an average-case parameterized complexity theory. There is, however, no link between average-case and worst-case complexity and no lower bounds for more natural probability distributions.

Hardness results on less artificial distributions are known if one considers counting problems instead of decision problems. Müller presented a counting problem on matrices that is hard on average on certain uniform distributions unless W[1] $\subseteq$ paraNP-BPFPT [35], thereby linking parameterized average-case complexity and classical parameterized complexity.

Our main result is a lower bound for FO model-checking on certain vertex-colored random graph models, in particular certain Erdős–Rényi and Chung–Lu graphs. If for every coloring function we can solve FO model-checking on such a random graph in expected FPT time, then AW[$*$] $\subseteq$ FPT/poly, which is quite unexpected because i.a. every level of the W-hierarchy is contained in AW[$*$]. While the hardness results hold for many random graph models (see Theorem 14) the following theorem shows three examples:

▶ **Theorem 3.** *If* p-MC(FO, $\mathfrak{G}_{col}$) *can be decided in expected* FPT *time on any of the following random graph models, then* AW[$*$] $\subseteq$ FPT/poly*:*

- $G(n, 1/2)$*,*
- $G(n, p(n))$ *with* $p(n) = n^\delta/n$ *for some* $0 < \delta < 1$*,* $\delta \in \mathbf{Q}$*,*
- *Chung–Lu graphs with exponent* $2.5 \leq \alpha < 3$*,* $\alpha \in \mathbf{Q}$*.*

This might be the first result where parameterized average-case hardness of a natural problem with a natural probability distribution is linked to classical complexity assumptions. For a more complete list of hard random graph models see Section 3.3. Our intractability results tightly match known tractability results:

▶ **Corollary 4.** *Assume that* AW[$*$] $\not\subseteq$ FPT/poly *and let* $p\colon \mathbf{N} \to [0, \frac{1}{2}]$ *be monotone and computable in polynomial time. Then* p-MC(FO, $\mathfrak{G}_{col}$) *can be decided on Erdős–Rényi graphs* $G(n, p(n))$ *in expected* FPT *time if and only if* $p(n) = O(n^\varepsilon/n)$ *for every* $\varepsilon > 0$*.*

In Theorem 3 and Corollary 4 we mention expected FPT time which is not closed under invoking polynomial subroutines. Our hardness results also hold for more permissive measures of parameterized tractability, similar to *average polynomial run time* [31], as introduced by Levin (see Theorem 8 and 9).

The drawback of this result is the usage of colored graphs, which turns out to be instrumental in the proof and it seems that colorings introduce a bit of worst-case behavior into the otherwise random sampling of a graph. In the past every positive result about model-checking on graphs worked for colored and uncolored graphs alike, but it is not clear whether something similar is true in the world of average-case complexity. The big open question that remains is whether we can prove a similar lower bound on uncolored random graphs and this paper is merely trying to contribute in the building of tools to reach this goal. A next step could be to show lower bounds under more restricted colorings.

There are examples for other lower bounds, which were shown first on colored graphs. One example is Kreutzer's proof that a graph class whose tree-width is not bounded, but grows at least moderately, has no efficient MSO model-checking algorithm. One of the conditions on the graph class was closure under colorings [29]. This lower bound complements Courcelle's theorem [11], which states that MSO model-checking can be done in linear time on graph classes with bounded tree-width (with or without colors). Later Kreutzer and Tazari could replace closure under colorings by closure under subgraphs, which is less restrictive [30]. Ganian et al. reintroduced closure under (vertex-)colorings to prove a similar result for MSO_1 model-checking [22]. These examples show that lower bounds are sometimes easier to prove in the presence of colorings.

Structural Sparsity and Average-Case Model-Checking

For monotone graph classes, model-checking is tractable if the graph class is nowhere-dense and intractable if it is somewhere-dense [25]. For random graph models, even for sparse random graph models, the situation is more complicated. We say a random graph model $\mathcal{G}$ is *asymptotically almost surely (a.a.s.) somewhere-dense* if there exists a somewhere-dense

graph class $\mathcal{H}$ such that for $n \to \infty$, a graph sampled from $\mathcal{G}_n$ belongs with probability one to $\mathcal{H}$. We similarly define *a.a.s. nowhere-dense*. Erdős–Rényi graphs, Chung–Lu graphs, the configuration model, and preferential attachment graphs have been classified with respect to a.a.s. somewhere- and nowhere-density [36, 13]. However, it is important to note that *a.a.s. somewhere-density is neither necessary nor sufficient for intractability*:

- There exist a.a.s. somewhere-dense random graph models for which p-MC(FO, $\mathfrak{G}_{col}$) is fixed parameter tractable.
- There exist a.a.s. nowhere-dense random graph models for which p-MC(FO, $\mathfrak{G}$) is not fixed parameter tractable (under some assumptions).

We state why: On the one hand, there exist trivial dense random graph models for which p-MC(FO, $\mathfrak{G}_{col}$) is tractable, such as the complete graph. In an upcoming work, we want to show that p-MC(FO, $\mathfrak{G}_{col}$) even is decidable on the a.a.s. somewhere-dense preferential attachment model in expected FPT time. On the other hand, Fountoulakis, Friedrich, and Hermelin showed that under the hypothesis distW[1] $\not\subseteq$ avgFPT $\cup$ typFPT there exists a random graph model $\mathcal{H}$ for which the parameterized clique problem is not tractable [18]. We construct a random graph model $\mathcal{G}$ as follows: With probability $1/\log(n)$, $\mathcal{G}_n$ is distributed according to $\mathcal{H}_n$ and with probability $1 - 1/\log(n)$, $\mathcal{G}_n$ is an independent set of size n. Now $\mathcal{G}$ is a.a.s. sparse. If p-MC(FO, $\mathfrak{G}$) could be solved on $\mathcal{G}$ in expected FPT time then it could also be solved on $\mathcal{H}$ in expected FPT time, which contradicts our hypothesis.

Connections to Shallow Topological Clique Minors

A byproduct of our work is a polynomial lower bound on the size of subdivided cliques in sufficiently dense random graphs, and polynomial time algorithms to find them. Let $\varepsilon > 0$. Dvořák [16] and Jiang [26] showed independently (using a slightly different formulation) that there exists a sequence $(\ell_{n,\varepsilon})_{n\in\mathbf{N}}$ with $\lim_{n\to\infty} \ell_{n,\varepsilon} = \infty$ and an integer c_ε such that every graph G with n vertices and at least $n^{1+\varepsilon}$ edges contains a c_ε-subdivision of a clique of size $\ell_{n,\varepsilon}$. Jiang further showed that one can choose $c_\varepsilon = 10/\varepsilon$. Both authors did not give a lower bound on $\ell_{n,\varepsilon}$.

A simple application of Chernoff bounds together with this result yields that $G(n, n^\varepsilon/n)$ contains a.a.s. a $c_{\varepsilon/2}$-subdivided clique of size $\ell_{n,\varepsilon/2}$ with $\lim_{n\to\infty} \ell_{n,\varepsilon} = \infty$. To obtain good lower bounds for the model-checking problem we need the clique size $\ell_{n,\varepsilon}$ to be polynomial in n. Therefore we show that $G(n, n^\varepsilon/n)$ contains a.a.s. a $6/\varepsilon$-subdivided clique of size $n^{\varepsilon/5}$ (Lemma 15). We also show that the Chung–Lu random graph model with exponent $2.5 \le \alpha < 3$ contains a.a.s. a one-subdivided clique of polynomial size (Lemma 18). We further show that these shallow clique minors can be found in polynomial time.

It is left as an open question whether $l_{n,\varepsilon}$ grows polynomially in n for general graphs. If affirmative then the result is probably not easy to prove: Kostochka and Pyber showed that a graph with n vertices and at least $4^{t^2} n^{1+\varepsilon}$ edges contains a subdivision of K_t with at most $7t^2 \ln(t)/\varepsilon$ (principal and non-principal) vertices [28]. Even with such weaker requirements the bound on t is smaller than $\log n$ and leaves an exponential gap to be filled.

Our Techniques

We use colorings and so called *FO-interpretations* on random graphs. An FO-interpretation a translates a graph G into another graph G' with the help of an FO-formula $\psi(x, y)$: The graph G' contains an edge (x, y) iff $G \models \psi(x, y)$. FO-interpretations are a standard tool in logic and have lately been used in the context of model-checking (for example [20, 21]).

We use FO-interpretations to establish a reduction framework. Let X and Y be two random graph models. We say Y *polynomially FO-interprets to* X ($X \preccurlyeq Y$) if an interpretation of a coloring of a graph G is nearly distributed like X assuming that G is distributed like Y (Definition 11). We further say that X *polynomially FO-interprets to* $\mathfrak{G}$ ($\mathfrak{G} \preccurlyeq X$) if an arbitrary graph $H \in \mathfrak{G}$ can be encoded into X using colorings and interpretations (Definition 7). In both cases, we require the interpretations to change the size of the graph at most polynomially. We show that p-MC(FO, $\mathfrak{G}_{col}$) is not tractable on any random graph model X with $\mathfrak{G} \preccurlyeq X$ (Theorem 9) and that $\preccurlyeq$ propagates hardness (Lemma 12, 13), yielding a way to prove intractability of p-MC(FO, $\mathfrak{G}_{col}$) on random graphs by means of reductions.

Let $\mathcal{G}$ be a random graph model with $\mathfrak{G} \preccurlyeq \mathcal{G}$. Let H be a graph and φ an FO-sentence. We sample a graph G from $\mathcal{G}_n$ for some $n = |H|^{O(1)}$ and use colorings and FO-interpretations to encode H with sufficiently high probability into G. Then one can solve the model-checking problem on H by solving it on the coloring of G instead. If p-MC(FO, $\mathfrak{G}_{col}$) can be decided on $\mathcal{G}$ in expected FPT time then one can also solve the model-checking problem on *every graph* H in expected FPT time. The result is a link between worst-case and average-case complexity.

2 Preliminaries

2.1 Graph Notation

We use common graph theory notation [14]. In this work we obtain results for undirected *colored graphs* [24]. A colored graph is a tuple $G = (V(G), E(G), C_1(G), \dots, C_l(G))$ with $C_i \subseteq V(G)$. We call $C_1(G), \dots, C_l(G)$ the *colors* of G. Vertices may have multiple colors. We say a vertex v is colored with color C_i if $v \in C_i$. All notion for graphs extends to colored graphs as expected. We define $\mathfrak{G}$ to be the class of all graphs and $\mathfrak{G}_{col}$ to be the class of all colored graphs. A *coloring* of an uncolored graph G is a colored graph G' with $(V(G'), E(G')) = G$. We define the *order* of a graph G to be $|G| = |V(G)|$.

A *r-subdivision*[1] of a graph H is a graph H' obtained by replacing edges with disjoint paths of length at least 2 and at most $r+1$. The *principal vertices* of H' are those vertices which are in H. We say a graph G contains an *r-subdivided induced clique* of size k if an r-subdivision of K_k is an induced subgraph of G. We say a graph G contains an *one-subdivided half-induced clique* of size k if there exist k vertices $v_1, \dots, v_k \subseteq V(G)$ such that for every $1 \leq i < j \leq k$ there exists a vertex $w_{i,j} \in V(G)$ with $N(w_{i,j}) \cap \{v_1, \dots, v_k\} = \{v_i, v_j\}$. We call $v_1, \dots, v_k$ the *principal vertices* and $w_{i,j}$ the *bridge* between v_i and v_j.

2.2 Probabilities and Random Graph Models

We denote probabilities by $\Pr[*]$ and expectation by $\mathrm{E}[*]$. We consider a random graph model to be a sequence of probability distributions. A random graph model describes for every $n \in \mathbf{N}$ a probability distribution on graphs with n vertices. In order to speak of probability distributions over graphs we fix a sequence of vertices $(u_i)_{i \geq 1}$ and require that a graph with n vertices has the vertex set $\{u_1, \dots, u_n\}$. A random graph model is a sequence $\mathcal{G} = (\mathcal{G}_n)_{n \in \mathbf{N}}$, where $\mathcal{G}_n$ is a probability distribution over all graphs G with $V(G) = \{u_1, \dots, u_n\}$. We write $G \sim \mathcal{G}_n$ if a graph G is is distributed as $\mathcal{G}_n$. In slight abuse of notation, we sometimes treat a probability distribution $\mathcal{G}_n$ as a random variable itself. For example the random variable $E(\mathcal{G}_n)$ stands for $E(G)$ with $G \sim \mathcal{G}_n$. We say a property of a random graph model holds *a.a.s.* if the probability that the property holds in $\mathcal{G}_n$ converges to one for $n \to \infty$.

[1] Usually, in an r-subdivision paths have length exactly $r+1$ but this definition is more convenient for us.

Erdős–Rényi graphs [6] with edge probability $p(n)$ are denoted by $G(n, p(n))$. The Chung–Lu model [10, 9, 7] has been proposed to generate random graphs that fit a certain degree sequence. For a given $n \in \mathbf{N}$ let $W_n = (w_1, \dots, w_n)$ be a sequence of weights. The Chung–Lu random graph to W_n is a random graph $\mathcal{G}_n$ with $V(\mathcal{G}_n) = \{u_1, \dots, u_n\}$ such that each edge $u_i u_j$ with $1 \leq i, j \leq n$ occurs in $\mathcal{G}_n$ independently with probability $w_i w_j / \sum_{k=1}^{n} w_k$. Let $\alpha > 2$. We say $\mathcal{G}$ is the *Chung–Lu random graph model with exponent* α if for every $n \in \mathbf{N}$, $\mathcal{G}_n$ is the Chung–Lu random graph to $W_n = \{w_1, \dots, w_n\}$ with $w_i = c \cdot (n/i)^{1/(\alpha-1)}$ where c is a constant depending on α.

We say a random graph model $\mathcal{G}$ is *expected polynomial time samplable* if there exists a randomized algorithm which runs on input $n \in \mathbf{N}$ in expected time polynomial in n and creates an output which is distributed like $\mathcal{G}_n$. This excludes for example Erdős–Rényi graphs where $p(n)$ is not computable.

2.3 First-Order Logic

We consider only first-order logic over colored graphs. We interpret a colored graph $G = (V, E, C_1, \dots, C_l)$, as a structure over the universe V with signate $(E, C_1, \dots, C_l)$. The binary relation E expresses adjacency between vertices and the unary relations $C_1, \dots, C_l$ indicate the colors of the vertices. Other structures can be easily converted into colored graphs. We write $\varphi(x_1, \dots, x_k)$ to indicate that a formula φ has *free variables* $x_1, \dots, x_k$. Furthermore, $|\varphi|$ is the number of symbols in φ.

An FO interpretation is a pair $I = (\nu(x), \psi(x, y))$ of FO formulas (with one and two free variables, respectively). For a colored graph G, this defines an uncolored graph $I(G)$ with $V(I(G)) = \{v \mid G \models \nu(v)\}$, $E(I(G)) = \{uv \mid G \models \psi(u, v)\}$. For an FO sentence φ, the interpretation I defines an FO sentence φ^I as follows: Every occurrence of $E(x, y)$ is replaced with $\psi(x, y) \vee \psi(y, x)$. Every occurrence of $\exists x \psi$ is replaced with $\exists x \nu(x) \wedge \psi$ and every occurrence of $\forall x \psi$ is replaced with $\forall x \nu(x) \to \psi$. Then $G \models \varphi^I \iff I(G) \models \varphi$.

2.4 Parameterized Complexity

The classes paraNP-BPFPT and FPT/poly are parameterized analogues of BPP and P/poly.

▶ **Definition 5.** [34] paraNP-BPFPT is the class of parameterized problems that can be decided by a randomized Turing machine with two-sided error $1/n$ (where n is the size of the input). The run time of the Turing machine on every input of size n and parameter k needs to be at most $f(k)n^{O(1)}$ for some computable function f.

▶ **Definition 6.** [12] FPT/poly is the class of all parameterized problems Q for which there exists a parameterized problem $Q' \in \text{FPT}$ and a constant c such that for each $(n, k) \in \mathbf{N} \times \mathbf{N}$ there exists some $\alpha(n, k) \in \Sigma^*$ of size n^c, with the property that for all instances $\langle x, k \rangle$ with $|x| = n$, it holds that $\langle x, k \rangle \in Q$ if and only if $\langle (x, \alpha(|x|, k)), k \rangle \in Q'$.

The complexity class AW[$*$] can be defined as the class of all parameterized problems that can be reduced to p-MC(FO, $\mathfrak{G}$) by fpt-reductions and contains the whole W-hierarchy. In summary

$$\text{FPT} \subseteq \text{W[1]} \subseteq \text{W[2]} \subseteq \text{W[3]} \subseteq \cdots \subseteq \text{AW[}*\text{]} \subseteq \text{XP},$$

$$\text{paraNP-BPFPT} = \text{FPT} \iff \text{BPP} = \text{P} \text{ (see [35])}.$$

It is widely believed that BPP = P.

3 Results

3.1 Hardness

In this section we define a property of random graph models (denoted by $\mathfrak{G} \preccurlyeq \mathcal{G}$, see Definition 7) and then show that p-MC(FO, $\mathfrak{G}_{col}$) is not tractable on average on random graph models with this property. We base our results either on the assumption AW[$*$] $\not\subseteq$ FPT/poly or on AW[$*$] $\not\subseteq$ paraNP-BPFPT. As discussed in Section 2.4, these are well-funded assumptions of parameterized worst-case complexity. In the following definition we characterize those random graph models which have enough structure such that an arbitrary graph can with high probability be embedded using FO-interpretations and colorings.

▶ **Definition 7.** Let $\mathcal{G}$ be a random graph model. We say $\mathcal{G}$ *polynomially FO-interprets to* $\mathfrak{G}$ *(short:* $\mathfrak{G} \preccurlyeq \mathcal{G}$*)* if $\mathcal{G}$ is expected polynomial time samplable and there exists an FO interpretation I, a polynomial p, a constant c and a randomized algorithm B which gets as input $G, H \in \mathfrak{G}$ with $|G| = p(|H|)$ and runs in polynomial time in $|H|$. Either the algorithm fails and $B(G,H)$ is a special failure symbol $\perp$ or $B(G,H)$ is a coloring of G with $I(B(G,H)) = H$. Furthermore for every H, $\Pr(B(\mathcal{G}_{p(|H|)}, H) \neq \perp) \geq 1/|H|^c$.

Our notion of tractability (as used in the following two theorems) is similar to the well-known notion of *average polynomial time* (for example [3, Definition 3]): Let $\mathcal{D} = \{\mathcal{D}_n\}_{n\in\mathbf{N}}$ be a sequence of distributions. A deterministic algorithm with run time $t(x)$ on input x runs in average polynomial time on $\mathcal{D}$ if there exists an $\varepsilon > 0$ and a polynomial p such that for every n and t, $\Pr_{x\sim\mathcal{D}_n}[t(x) \geq t] \leq p(n)/t^\varepsilon$. Average polynomial time is closed under polynomial subroutines and implies polynomial expected time. We show that if one can solve the model-checking problem *with adversary colorings* efficiently then AW[$*$] $\subseteq$ paraNP-BPFPT.

▶ **Theorem 8.** *Let $\mathcal{G}$ be a random graph model with $\mathfrak{G} \preccurlyeq \mathcal{G}$. If there exists a function f, a polynomial p, an $\varepsilon > 0$ and a deterministic algorithm $\mathcal{A}$ which decides p-MC(FO, $\mathfrak{G}_{col}$) on input G, φ in time $t_{\mathcal{A}}(G,\varphi)$ such that for all $n, t \in \mathbf{N}$, all FO-sentences φ and all $|\varphi|$-coloring functions C it holds that*

$$\Pr\big[t_{\mathcal{A}}\big(C(\mathcal{G}_n), \varphi\big) \geq t\big] \leq f(|\varphi|)p(n)/t^\varepsilon,$$

then AW[$*$] $\subseteq$ paraNP-BPFPT. *Moreover, then there exists a randomized algorithm which gets as input a graph $H \in \mathfrak{G}$ and a FO-sentence φ and returns whether $G \models \varphi$ or $\perp$, whereas $\perp$ is returned with probability at most $1/2$. The algorithm always runs in time $g(|\varphi|)|H|^{O(1)}$ for some function g and uses only $|H|^{O(1)}$ random bits.*

Proof. It is known that p-MC(FO, $\mathfrak{G}$) is AW[$*$]-complete [15]. We assume $\mathfrak{G} \preccurlyeq \mathcal{G}$ with interpretation I, polynomial q and algorithm B. Let H be a graph with $|H| = n$ and φ be a FO-sentence. Consider the following procedure: We sample a graph G with $G \sim \mathcal{G}_{q(n)}$. We then compute $B(G,H)$, which is either a coloring of G or $\perp$. If $B(G,H) = \perp$, we return $\perp$. If not then $I(B(G,H)) = H$. This means $H \models \varphi \iff B(G,H) \models \varphi^I$. In this case, we use $\mathcal{A}$ to compute whether $B(G,H) \models \varphi^I$.

Let us analyze this procedure: Since $\Pr(B(\mathcal{G}_{q(n)}, H) \neq \perp) \geq 1/n^c$ for some constant c, the probability that the procedure returns $\perp$ is at most $1 - 1/n^c$. Let C be a $|\varphi^I|$-coloring function such that for every graph $G' \in \mathfrak{G}$, $C(G')$ is a coloring of G' with $|\varphi^I|$ colors maximizing $t_{\mathcal{A}}(C(G'), \varphi^I)$. We can assume that $\mathcal{A}$ immediately returns $\perp$ on a malformed input $\perp$, thus $t_{\mathcal{A}}(B(G,H), \varphi^I) \leq t_{\mathcal{A}}(C(G), \varphi^I)$. Therefore for every $t \in \mathbf{N}$, $\Pr\big[t_{\mathcal{A}}(B(H, \mathcal{G}_{q(n)}), \varphi^I) \geq t\big] \leq \Pr\big[t_{\mathcal{A}}(C(\mathcal{G}_{q(n)}), \varphi^I) \geq t\big]$. Let $g(|\varphi|) = f(|\varphi^I|)^{1/\varepsilon}$. By our

assumption, we have $\Pr\big[t_{\mathcal{A}}(C(\mathcal{G}_{q(n)}), \varphi^I) \geq g(|\varphi|)t\big] \leq p(q(n))/t^\varepsilon$. Thus, for every $t \in \mathbf{N}$, $\mathcal{A}$ does not terminate on $(B(G,H), \varphi^I)$ after $g(|\varphi|)t$ steps with probability at most $p(q(n))/t^\varepsilon$. We choose $t = n^{O(1)}$ such that $p(q(n))/t^\varepsilon \leq 1/4n^c$, run $\mathcal{A}$ for $g(|\varphi^I|)t$ steps, and then abort it. The probability that it was aborted is at most $1/4n^c$. The graph $G \sim \mathcal{G}_{q(n)}$ is sampled in expected polynomial time. We also abort the sampling procedure after $n^{O(1)}$ steps such that the probability that it was aborted is at most $1/4n^c$. In total, the procedure was aborted with probability at most $1/2n^c$ and returns $\perp$ with probability at most $1 - 1/n^c$, leaving a probability of at least $1/2n^c$ to not be aborted and to not return $\perp$. Therefore, we compute whether $H \models \varphi$ with probability at least $1/2n^c$ and always run in FPT time. We repeat this $n^{O(1)}$ times to amplify the probability. Altogether we needed randomness to sample from $\mathcal{G}_n$ and to run B for a polynomial number of steps, which results in the usage of polynomially many random bits. ◀

Adleman's theorem states that BPP $\subseteq$ P/poly [1, Theorem 7.17]. The following proof is very similar to its proof, thus we are brief.

▶ **Theorem 9.** *Let $\mathcal{G}$ be a random graph model with $\mathfrak{G} \preccurlyeq \mathcal{G}$. If there exists a function f, a polynomial p, an $\varepsilon > 0$ and a deterministic algorithm $\mathcal{A}$ which decides p-MC(FO, $\mathfrak{G}_{col}$) on input G, φ in time $t_{\mathcal{A}}(G, \varphi)$ such that for all $n, t \in \mathbf{N}$, all FO-sentences φ and all $|\varphi|$-coloring functions C*

$$\Pr\big[t_{\mathcal{A}}\big(C(\mathcal{G}_n), \varphi\big) \geq t\big] \leq f(|\varphi|)p(n)/t^\varepsilon,$$

then AW[$*$] $\subseteq$ FPT/poly.

Proof. It is known that p-MC(FO, $\mathfrak{G}$) is AW[$*$]-complete [15]. If we assume that the preconditions of this theorem are fulfilled then by Theorem 8 there is a randomized algorithm that can decide for a graph $G \in \mathfrak{G}$ with $|G| = n$ and FO-sentence φ whether $G \models \varphi$ in FPT time with probability at least $1/2$ and that returns $\perp$ otherwise. Moreover, that algorithm uses only $n^{O(1)}$ random bits.

We can run that algorithm n^3 times and the probability that we get at least once the answer to $G \models \varphi$ is then at least $1 - 2^{-n^3}$. There are at most 2^{n^2} graphs with a fixed vertex set of size n. For each of these at most 2^{n^2} possible input graphs G there is a fraction of at most 2^{-n^3} of possible random bit strings that cause the amplified algorithm to fail on G. The total fraction of random bit strings that cause the algorithm to fail on at least one G is then at most $2^{-n^3} \cdot 2^{n^2} < 1$ and there must be at least one choice of random bits that causes the algorithm to give the right answer on every colored graph G of order n. Feeding the randomized algorithm this fixed string of bits instead of using true random bits yields a deterministic FPT algorithm that needs for each n an advice string of polynomial length, which places the problem in FPT/poly. ◀

A simple application of the Markov bound yields the following more compact statement.

▶ **Corollary 10.** *Let $\mathcal{G}$ be a random graph model with $\mathfrak{G} \preccurlyeq \mathcal{G}$. If p-MC(FO, $\mathfrak{G}_{col}$) can be solved on $\mathcal{G}$ in expected* FPT *time then* AW[$*$] $\subseteq$ FPT/poly.

3.2 Reductions

In this section we use colorings and FO-interpretations to define reductions between random graph models (Definition 11) and show that these reductions are transitive and propagate hardness (Lemma 12 and 13). For two random graph models X and Y we say Y *polynomially FO-interprets to* X ($X \preccurlyeq Y$) if an interpretation of a coloring of a graph G is nearly distributed like X, assuming G is distributed like Y. This section is a technical necessity, but contains no surprising results.

▶ **Definition 11.** Let X and Y be random graph models. We say Y *polynomially FO-interprets to* X ($X \preccurlyeq Y$) if Y is expected polynomial time samplable and there exists an interpretation I, a polynomial p, a constant c and a randomized algorithm C which gets as input an uncolored graph G, runs in polynomial time in $|G|$ and returns $C(G)$. Either the algorithm fails and $C(G)$ equals a special failure symbol $\perp$ or $C(G)$ is a coloring of G. Furthermore for every x in the support of X_n, $\Pr_{G\sim Y_{p(n)}}[I(C(G)) = x] \geq \Pr_{G\sim X_n}[G = x]/n^c$.

▶ **Lemma 12.** *Let X and Y be random graph models. If $\mathfrak{G} \preccurlyeq X$ and $X \preccurlyeq Y$ then $\mathfrak{G} \preccurlyeq Y$.*

Proof. Assume $X \preccurlyeq Y$ with algorithm C_1, interpretation I_1, polynomial p_1 and constant c_1. and $\mathfrak{G} \preccurlyeq X$ with algorithm B_2, interpretation I_2, polynomial p_2 and constant c_2. Let I be the interpretation obtained by applying I_1 and then I_2. By Definition 11, Y is expected polynomial time samplable. It is sufficient to construct a randomized algorithm B which gets as input $G, H \in \mathfrak{G}$ with $|G| = p_2(p_1(|H|))$ and $B(G, H)$ is either $\perp$ or a coloring of G with $I(B(G, H)) = H$, and for every $H \in \mathfrak{G}$, $\Pr_{G\sim Y_{p_1(p_2(|H|))}}(B(G, H) \neq \perp) \geq 1/|H|^c$ for some c.

This algorithm proceeds as follows: At first, we compute $C_1(G)$. If $G' = I_1(C_1(G))$ is $\perp$ or no graph of order $p_2(n)$ we return $\perp$. Then we compute $B_2(G', H)$. Again, if $B_2(G', H) = \perp$ we return $\perp$. With at least polynomial probability no $\perp$ was returned, since

$$\begin{aligned}\Pr_{G\sim Y_{p_1(p_2(|H|))}}[B_2(G', H) \neq \perp] &= \sum_{G'} \Pr_{G\sim Y_{p_1(p_2(|H|))}}[I_1(C_1(G)) = G', B_2(G', H) \neq \perp] \\ &\geq \Pr_{G'\sim X_{p_2(|H|)}}[B_2(G', H) \neq \perp]/p_2(|H|)^{c_1} \geq 1/(p_2(|H|)^{c_1}|H|^{c_2}) \geq 1/|H|^c,\end{aligned}$$

for some constant c. If no $\perp$ was returned, we know that $I_2(B_2(I_1(C_1(G)), H)) = H$. This means H can be obtained from G by first putting colors on it via C_1, then interpreting it via I_1, then again putting colors on it via B_2 and again interpreting it via I_2. The same result can be obtained by putting all colors directly on G and then applying I_1 and I_2. Let G^* be the coloring of G which contains the colors of $C(G)$ and the lifted colors of $B_2(G', H)$. The algorithm returns $B(G, H) = G^*$. Then $I(G^*) = I_2(I_1(G^*)) = H$. ◀

With the same techniques as in the lemma before one can show that $\preccurlyeq$ is transitive. Since the proof is straightforward, we leave it out.

▶ **Lemma 13.** *Let X, Y, and Z be random graph models. If $X \preccurlyeq Y$ and $Y \preccurlyeq Z$ then $X \preccurlyeq Z$.*

3.3 Hard Random Graph Models

In this section we prove intractability of p-MC(FO, $\mathfrak{G}_{col}$) for certain random graph models. As a side result, we obtain polynomial lower bounds on the size of shallow clique minors in Erdős–Rényi graphs (Lemma 15) and Chung–Lu graphs (Lemma 18). The main results are summarized in the following theorem.

▶ **Theorem 14.** *The following list of random graph models polynomially FO-interpret to $\mathfrak{G}$ (for $0 < \varepsilon < 1$):*

- **(i)** *every Erdős–Rényi random graph model $G(n, p(n))$ with $n^\varepsilon/n \leq p(n) \leq 1 - n^\varepsilon/n$ that is expected polynomial time samplable,*
- **(ii)** *every Chung–Lu random graph model with exponent $\alpha \in \mathbf{Q}$ with $2.5 \leq \alpha < 3$,*
- **(iii)** *every expected polynomial time samplable random graph model $\mathcal{G}$ such that in $\mathcal{G}_n$ every edge is independent and has an individual probability p with $n^\varepsilon/n \leq p \leq 1/n^\varepsilon$.*

Proof. The second and third case correspond to Lemma 19 and16 respectively. For the first case, the interval $[n^\varepsilon/n, 1 - n^\varepsilon/n]$ can be broken into three parts: $[n^\varepsilon/n, n^{-1/2}]$, $[n^{-1/2}, 1 - n^{-1/2}]$, $[1 - n^{-1/2}, 1 - n^\varepsilon/n]$. The first interval corresponds to Lemma 16. The third interval can be reduced to the first interval using the complement graph. The missing region is filled by Lemma 17. ◀

▶ **Lemma 15.** *Let $\varepsilon > 0$. Let $\mathcal{G}$ be a random graph model such that in $\mathcal{G}_n$ every edge is independent and has an individual probability p with $n^\varepsilon/n \leq p \leq 1/n^\varepsilon$. There exists a deterministic polynomial time algorithm which gets a graph with n vertices and either returns $\perp$ or an induced $6/\varepsilon$-subdivided clique of size $\lfloor n^{\varepsilon/5} \rfloor$. If the input is distributed according to $\mathcal{G}$ then a.a.s. the algorithm does not return $\perp$.*

Proof. Let $n \in \mathbf{N}$ and $k = \lfloor n^{\varepsilon/5} \rfloor$. We fix k principal vertices $v_1, \ldots, v_k$. Each edge in $\mathcal{G}_n$ occurs with probability at least n^ε/n. Thus, the expected degree of each principal vertex is at least $n^\varepsilon = \Theta(k^5)$. The degree of a vertex is a sum of independent Bernoulli variables. According to the Chernoff Bound, a.a.s., each principal vertex has degree at least k^2. We assume that this is the case. This means, there are k^2 distinct nodes w_j^i $(1 \leq i, j \leq k)$ such that w_j^i is adjacent to v_i.

We choose k^2 disjoint subsets $S_{i,j}$ $(1 \leq i, j \leq k)$ of size $N = \lceil n/k^3 \rceil$ such that $S_{i,j}$ contains w_i^j and w_j^i but none of the vertices $v_1, \ldots, v_k$. Each subgraph $\mathcal{G}_n[S_{i,j}]$ can be interpreted as a random graph of order N where edges have independent probability at least $n/n^\varepsilon = \Theta(N/N^{\varepsilon/3})$. It is known that the diameter of a graph $G(n, p(n))$ is a.a.s. at most $\lceil \log(n)/\log(np(n)) \rceil$ [8, Theorem 2] (also [27, 4, 37]). By the argument above, $\mathcal{G}_n[S_{i,j}]$ a.a.s. has diameter at most $\lceil \log(N)/\log(N^{\varepsilon/3}) \rceil = \lceil 3/\varepsilon \rceil \leq 4/\varepsilon$. While not explicitly mentioned in the references, this holds with sufficiently high probability to guarantee that that a.a.s. *each* subgraph $\mathcal{G}_n[S_{i,j}]$ $(1 \leq i, j \leq k)$ has diameter at most $4/\varepsilon$. We again assume that this is the case. Then for each pair of vertices w_i^j, w_j^i we compute a path $p_{i,j}$ in $S_{i,j}$ of length at most $4/\varepsilon$ connecting them. All paths $p_{i,j}$ are disjoint. Therefore, the principal vertices $v_1, \ldots, v_j$ together with the paths $p_{i,j}$ span a $4/\varepsilon + 2$-subdivided clique.

At last, we need to show that this subdivided clique is a.a.s. an induced subdivided clique. The subdivided clique consists of at most $O(k^2)$ vertices. Thus, there is a set F of at most $O(k^4) = O(n^{\varepsilon 4/5})$ possible edges whose presence would mean that the subdivided clique is no induced subdivided clique. Let A be the event that the degree of all principal vertices is at least k^2 and that each subgraph $\mathcal{G}_n[S_{i,j}]$ $(1 \leq i, j \leq k)$ has diameter at most $4/\varepsilon$. Now we need to show that with high probability no edge from F is present, assuming that A holds. Earlier, we showed that A holds a.a.s. Thus, there exists n_0 such that for $n \geq n_0$, $P[A] \geq 1/2$. For every event B and $n \geq n_0$ holds $\Pr[\bar{B} \mid A] \leq \Pr[\bar{B}]/\Pr[A] \leq 2\Pr[\bar{B}]$. Thus if a.a.s. no edge from F is present, then also a.a.s. no edge from F is present under the assumption that A holds. Each edge occurs with probability at most $n^{-\varepsilon}$. Thus, the probability that none of these edges in F is present is at least $\left(1 - n^{-\varepsilon}\right)^{O(n^{\varepsilon 4/5})}$ which converges to one. ◀

▶ **Lemma 16.** *Let $\varepsilon > 0$. Let $\mathcal{G}$ be an expected polynomial time samplable random graph model such that in $\mathcal{G}_n$ every edge is independent and has an individual probability p with $n^\varepsilon/n \leq p \leq 1/n^\varepsilon$. Then $\mathfrak{G} \preccurlyeq \mathcal{G}$.*

Proof. Let H be a graph with $|H| = n$ and $G \sim \mathcal{G}_{\lceil n^{5/\varepsilon} \rceil}$. We use the algorithm from Lemma 15 on G to a.a.s. compute an induced $6/\varepsilon$-subdivided clique of size n. If the algorithm from Lemma 15 returns $\perp$, we set $B(G, H) = \perp$, which a.a.s. never happens. Otherwise, we proceed as follows: Let $v_1, \ldots, v_n$ be the principal vertices of the induced clique and let $p_{i,j}$ $(1 \leq i < j \leq k)$ be the disjoint paths of length at most $6/\varepsilon$ connecting v_i and v_j. Each

path $p_{i,j}$ is adjacent to exactly two principal vertices and no other path. The final output $B(G,H)$ is constructed by adding colors C_1 and C_2 to G: We define $C_1 = \{v_1, \dots, v_n\}$ and $C_2 = \bigcup_{ij \in E(H)} V(p_{\min(i,j),\max(i,j)})$. Now i and j are adjacent in H if and only if v_i and v_j have a connecting path of length at most $6/\varepsilon$ in G which is colored completely with C_2.

Let further $I = (\nu(x), \psi(x,y))$ be the interpretation such that $\nu(x) = C_1(x)$ and $\psi(x,y)$ is the formula which checks if x and y have a connecting path of length at most $6/\varepsilon$ which is colored completely with C_2. The length of $\psi(x,y)$ depends on ε, which is a constant. Now if $B(G,H) \neq \bot$ then $B(G,H)$ is a coloring of G with $I(B(G,H)) = H$ and the probability of $B(G,H) = \bot$ is sufficiently low. Therefore $\mathfrak{G} \preccurlyeq \mathcal{G}$. ◀

▶ **Lemma 17.** *Let $G(n,p(n))$ be an expected polynomial time samplable Erdős–Rényi random graph model with $n^{-1/2} \leq p(n) \leq 1 - n^{-1/2}$. Then $\mathfrak{G} \preccurlyeq G(n,p(n))$.*

Proof. By Lemma 16 and Lemma 12, it is sufficient to prove $G(n, \lfloor\sqrt{n}\rfloor/n) \preccurlyeq G(n,p(n))$. We assume $n > 10$. Let $m = n^{14}$. Let G be the input graph with $G \sim G(m,p(m))$. We shall construct a randomized polynomial time algorithm C and an interpretation I such that $I(C(G))$ behaves similar to $G(n, \lfloor\sqrt{n}\rfloor/n)$. For sets $N \subseteq V(G)$ and vertices $u, v \in V(G) \setminus N$ with $u \neq v$ we say u and v have the same N-neighborhood if $N^G(u) \cap N = N^G(v) \cap N$. Since $G \sim G(m,p(m))$, the probability that u and v have the same N-neighborhood equals $c_N(m) := \big(p(m)^2 + (1-p(m))^2\big)^{|N|}$. We compute the minimal value $k \in \mathbf{N}$ such that $1/n^4 \leq \big(p(m)^2 + (1-p(m))^2\big)^k \leq 1/n^3$. Clearly, k exists and is computable in polynomial time. Furthermore, one can easily show that $k \leq m^{2/3}$.

We fix three sets $X, Y, Z \subseteq V(G)$ with $X = \{v_1, \dots, v_n\}$, $|Y| = |Z| = k$. This yields $1/n^4 \leq c_Y(m), c_Z(m) \leq 1/n^3$. We further fix $\binom{n}{2}$ subsets $S_{i,j} \subseteq V(G)$ for $1 \leq i < j \leq n$ with $|S_{i,j}| = n^{11}$. Since $n + k + \binom{n}{2} n^{11} \leq m$, we can assume all these subsets to be disjoint.

If there exist two vertices v_i and v_j with $1 \leq i \neq j \leq n$ which have the same Y-neighborhood or the same Z-neighborhood we return $\bot$. The probability that this happens is, by the union bound, at most $\binom{n}{2}(c_Y(m) + c_Z(m)) \leq 1/n$. Furthermore, if there exists $1 \leq i < j \leq n$ such that there is no vertex in $S_{i,j}$ which has the same Y-neighborhood as v_i and the same Z-neighborhood as v_j, we return $\bot$. For fixed i, j, the probability that there is no vertex in $S_{i,j}$ which has the same Y-neighborhood as v_i and the same Z-neighborhood as v_j is at most $(1 - c_Y(m)c_Z(m))^{|S_{i,j}|} \leq (1 - 1/n^8)^{n^{11}} \leq 1/n^3$. Thus, by the union bound, $\bot$ is returned with probability at most $\binom{n}{2}/n^3 \leq 1/n$. In total, $\bot$ is a.a.s. never returned.

If the algorithm did not return $\bot$, we know that the Y- and Z-neighborhoods of $v_1, \dots, v_n$ are distinct and for all $1 \leq i < j \leq n$ there exists a vertex $s_{i,j}$ which has the same Y-neighborhood as v_i and the same Z-neighborhood as v_j. We construct a set W which represents the edge set of a random graph $G(n, \lfloor\sqrt{n}\rfloor/n)$ by adding each vertex $s_{i,j}$ for $1 \leq i < j \leq k$ independently with probability $\lfloor\sqrt{n}\rfloor/n$ to W. Let the final output $C(G)$ be the graph G augmented with colors X, Y, Z, W. Let further $I = (\nu(x), \psi(x,y))$ be the interpretation where $\nu(x) = X(x)$ and $\psi(a,b)$ is the FO formula which checks if there exists a vertex $w \in W$ which has the same Y-neighborhood as a and the same Z-neighborhood as b. Now v_i and v_j with $i < j$ are adjacent in $I(C(G))$ if and only if $s_{i,j} \in W$, which occurs with probability $\lfloor\sqrt{n}\rfloor/n$. Therefore under the condition that $C(G) \neq \bot$ we have $I(C(G)) \sim G(n, \lfloor\sqrt{n}\rfloor/n)$. Since a.a.s. $I(C(G)) \neq \bot$, we have for every x in the support of $G(n, \lfloor\sqrt{n}\rfloor/n)$ and n sufficiently large, $\Pr_{G \sim G(m,p(m))}[I(C(G)) = x] \geq \Pr_{G \sim G(n, \lfloor\sqrt{n}\rfloor/n)}[G = x]/2$. This means $G(n, \lfloor\sqrt{n}\rfloor/n) \preccurlyeq G(n,p(n))$. ◀

The threshold $2.5 < \alpha$ in the following lemma has been chosen to ease the calculations and can be considerably improved.

▶ **Lemma 18.** *Let $\mathcal{G}$ be a Chung–Lu random graph model with exponent $2.5 \leq \alpha < 3$. There exists an $\varepsilon > 0$ such $\mathcal{G}_n$ contains a.a.s. a one-subdivided half-induced clique with principal vertices $u_1, \dots, u_{\lceil n^\varepsilon \rceil}$.*

Proof. Let $n \in \mathbf{N}$ and $V(\mathcal{G}_n) = \{u_1, \dots, u_n\}$. By definition [10], the probability of an edge $u_i u_j$ in $\mathcal{G}_n$ is $w_i w_j / \sum_{k=1}^n w_k$, where $w_k = \Theta(n/k)^{1/(\alpha-1)}$. One can easily verify that for $\alpha > 2$, $\sum_{k=1}^n w_k = \Theta(n)$. We choose $\varepsilon = 1/(\alpha-1) - 1/2$ and get

$$\Pr[u_i u_j \in E(\mathcal{G}_n)] = \frac{w_i w_j}{\sum_{k=1}^n w_k} = \frac{(n/i)^{1/(\alpha-1)}(n/j)^{1/(\alpha-1)}}{\Theta(n)} = \Theta(n^{2\varepsilon}(ij)^{-\varepsilon-1/2}).$$

Since $2.5 \leq \alpha < 3$, we have $0 < \varepsilon \leq 1/6$. Let $k = \lceil n^{\varepsilon/2} \rceil$. We fix $a, b \leq k$. For $n/2 \leq x \leq n$ let $p(x)$ be the probability that u_x is a bridge between u_a and u_b. Then

$$p(x) = \Omega\big(n^{2\varepsilon}(xa)^{-\varepsilon-1/2} n^{2\varepsilon}(xb)^{-\varepsilon-1/2}\big) \prod_{c=1}^{k} \Big(1 - O(n^{2\varepsilon}(xc)^{-\varepsilon-1/2})\Big) =$$
$$\Omega(k^{-2\varepsilon-1} n^{2\varepsilon-1}) \Big(1 - O(n^{\varepsilon-1/2})\Big)^k = \Omega(k^{-2} n^{2\varepsilon-1}) = \Omega(n^{\varepsilon-1}).$$

The probability that none of the vertices $u_{\lceil n/2 \rceil}, \dots, u_n$ are a bridge between u_a and u_b is at most

$$\prod_{x=\lceil n/2 \rceil}^{n} \Big(1 - p(x)\Big) = \Big(1 - \Theta(n^{\varepsilon-1})\Big)^{\Theta(n)} = e^{-\Theta(n^\varepsilon)}.$$

By the union bound, $\mathcal{G}_n$ contains no one-subdivided half-induced clique with principal vertices $u_1, \dots, v_k$ with probability at most $k e^{-\Theta(n^\varepsilon)}$. Since $k = \lceil n^{\varepsilon/2} \rceil$, this converges to zero. ◀

▶ **Lemma 19.** *Let $\mathcal{G}$ be a Chung–Lu random graph model with exponent $2.5 \leq \alpha < 3$, $\alpha \in \mathbf{Q}$. Then $\mathfrak{G} \preccurlyeq \mathcal{G}$.*

Proof. The proof is very similar to that of Lemma 16, therefore we merely sketch it. Since $\alpha \in Q$, we know that $\mathcal{G}$ is expected polynomial time samplable (see [1, Lemma 7.14]). Given a graph $G \sim \mathcal{G}_n$, by Lemma 18, the first $\lceil n^\varepsilon \rceil$ vertices of G a.a.s. are the principal vertices of a one-subdivided half-induced clique. If it exists, one can easily find the bridges of said clique in polynomial time. We color the principal vertices with color C_1 and a subset of the bridges with color C_2. We define an FO-interpretation which selects the principal vertices as nodes and connects them if they are joined by a bridge with color C_2. By choosing C_2 accordingly, we can construct an arbitrary graph of order $\lceil n^\varepsilon \rceil$. ◀

References

1. Sanjeev Arora and Boaz Barak. *Computational complexity: A modern approach*. Cambridge University Press, 2009.
2. Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
3. Andrej Bogdanov and Luca Trevisan. Average-Case Complexity. *Foundations and Trends in Theoretical Computer Science*, 2(1):1–106, 2006.
4. Béla Bollobás. The diameter of random graphs. *Transactions of the American Mathematical Society*, 267(1):41–52, 1981.
5. Béla Bollobás, Oliver Riordan, Joel Spencer, and Gábor Tusnády. The Degree Sequence of a Scale-free Random Graph Process. *Random Structures & Algorithms*, 18(3):279–290, May 2001.

6 B. Bollobás. *Random Graphs*. Cambridge University Press, 2nd edition, 2001.

7 Fan Chung, Fan RK Chung, Fan Chung Graham, Linyuan Lu, Kian Fan Chung, et al. *Complex graphs and networks*, volume 107. American Math. Soc., 2006.

8 Fan Chung and Linyuan Lu. The diameter of sparse random graphs. *Advances in Applied Mathematics*, 26(4):257–279, 2001.

9 Fan Chung and Linyuan Lu. Connected Components in Random Graphs with Given Expected Degree Sequences. *Annals of Combinatorics*, 6(2):125–145, 2002.

10 Fan Chung and Linyuan Lu. The average distances in random graphs with given expected degrees. *Proc. of the National Academy of Sciences*, 99(25):15879–15882, 2002.

11 Bruno Courcelle. The Monadic Second-Order Logic of Graphs I. Recognizable Sets of Finite Graphs. *Information and Computation*, 85(1):12–75, 1990.

12 Ronald de Haan. An Overview of Non-Uniform Parameterized Complexity. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:130, 2015.

13 E. D. Demaine, F. Reidl, P. Rossmanith, F. Sánchez Villaamil, S. Sikdar, and B. D. Sullivan. Structural Sparsity of Complex Networks: Random Graph Models and Linear Algorithms. *CoRR*, abs/1406.2587, 2014. To appear in JCSS. URL: `http://arxiv.org/abs/1406.2587`, `arXiv:1406.2587`.

14 R. Diestel. *Graph Theory*. Springer, Heidelberg, 2010.

15 Rodney G. Downey, Michael R. Fellows, and Udayan Taylor. The Parameterized Complexity of Relational Database Queries and an Improved Characterization of W[1]. In *First Conference of the Centre for Discrete Mathematics and Theoretical Computer Science, DMTCS 1996, Auckland, New Zealand, December, 9-13, 1996*, pages 194–213, 1996.

16 Z. Dvořák. *Asymptotical Structure of Combinatorial Objects*. PhD thesis, Charles University, Faculty of Mathematics and Physics, 2007.

17 Zdenek Dvořak, Daniel Král, and Robin Thomas. Deciding First-Order Properties for Sparse Graphs. In *Proceedings of the 51st Conference on Foundations of Computer Science*, pages 133–142, 2010.

18 Nikolaos Fountoulakis, Tobias Friedrich, and Danny Hermelin. On the Average-Case Complexity of Parameterized Clique. *Theoretical Computer Science*, 576:18–29, 2015.

19 Tobias Friedrich and Anton Krohmer. Parameterized Clique on Inhomogeneous Random Graphs. *Discrete Applied Mathematics*, 184:130–138, 2015.

20 Jakub Gajarský, Petr Hlinený, Daniel Lokshtanov, Jan Obdrzálek, and M. S. Ramanujan. A New Perspective on FO Model Checking of Dense Graph Classes. *CoRR*, abs/1805.01823, 2018. `arXiv:1805.01823`.

21 Jakub Gajarskỳ, Stephan Kreutzer, Jaroslav Nešetřil, Patrice Ossona de Mendez, Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. First-order interpretations of bounded expansion classes. *arXiv preprint arXiv:1810.02389*, 2018.

22 Robert Ganian, Petr Hlinený, Alexander Langer, Jan Obdrzálek, Peter Rossmanith, and Somnath Sikdar. Lower bounds on the complexity of MSO_1-model checking. *J. Comput. Syst. Sci.*, 80(1):180–194, 2014.

23 Martin Grohe. Generalized model-checking problems for first-order logic. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 12–26. Springer, 2001.

24 Martin Grohe. Logic, graphs, and algorithms. *Logic and Automata*, 2:357–422, 2008.

25 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding First-Order Properties of Nowhere Dense Graphs. *JACM*, 64(3):17, 2017.

26 Tao Jiang. Compact topological minors in graphs. *Journal of Graph Theory*, 67(2):139–152, 2011.

27 Victor Klee and David Larman. Diameters of random graphs. *Canadian Journal of Mathematics*, 33(3):618–640, 1981.

28 A. Kostochka and L. Pyber. Small topological complete subgraphs of "dense" graphs. *Combinatorica*, 8:83–86, 1988.

29 Stephan Kreutzer. On the Parameterized Intractability of Monadic Second-Order Logic. *Logical Methods in Computer Science*, 8(1), 2012.

30 Stephan Kreutzer and Siamak Tazari. Lower Bounds for the Complexity of Monadic Second-Order Logic. In *Proceedings of the 22nd Symposium on Logic in Computer Science*, pages 189–198. IEEE Computer Society, 2010.

31 Leonid A Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

32 M. Molloy and B. A. Reed. A Critical Point for Random Graphs with a Given Degree Sequence. *Random Structures & Algorithms*, 6(2/3):161–180, 1995.

33 M. Molloy and B. A. Reed. The Size of the Giant Component of a Random Graph with a Given Degree Sequence. *Combin., Probab. Comput.*, 7(3):295–305, 1998.

34 Juan Andrés Montoya and Moritz Müller. Parameterized random complexity. *Theory of Computing Systems*, 52(2):221–270, 2013.

35 Moritz Müller. *Parameterized Randomization*. PhD thesis, Albert-Ludwigs-Universität Freiburg, 2008.

36 Jaroslav Nešetřil, Patrice Ossona de Mendez, and David R Wood. Characterisations and examples of graph classes with bounded expansion. *European Journal of Combinatorics*, 33(3):350–373, 2012.

37 Oliver Riordan and Nicholas Wormald. The diameter of sparse random graphs. *Combinatorics, Probability and Computing*, 19(5-6):835–926, 2010.

38 Luc Segoufin and Alexandre Vigny. Constant Delay Enumeration for FO Queries over Databases with Local Bounded Expansion. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 68. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

39 Larry J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory*. PhD thesis, Dept. of Electrical Engineering, MIT, 1974.

Computing the Largest Bond of a Graph

Gabriel L. Duarte
Fluminense Federal University, Rio de Janeiro, Brazil
gabrield@id.uff.br

Daniel Lokshtanov
University of California Santa Barbara, CA, USA
daniello@ucsb.edu

Lehilton L. C. Pedrosa
University of Campinas, São Paulo, Brazil
lehilton@ic.unicamp.br

Rafael C. S. Schouery
University of Campinas, São Paulo, Brazil
rafael@ic.unicamp.br

Uéverton S. Souza[1]
Fluminense Federal University, Rio de Janeiro, Brazil
ueverton@ic.uff.br

Abstract

A bond of a graph G is an inclusion-wise minimal disconnecting set of G, i.e., bonds are cut-sets that determine cuts $[S, V \setminus S]$ of G such that $G[S]$ and $G[V \setminus S]$ are both connected. Given $s, t \in V(G)$, an st-bond of G is a bond whose removal disconnects s and t. Contrasting with the large number of studies related to maximum cuts, there are very few results regarding the largest bond of general graphs. In this paper, we aim to reduce this gap on the complexity of computing the largest bond and the largest st-bond of a graph. Although cuts and bonds are similar, we remark that computing the largest bond of a graph tends to be harder than computing its maximum cut. We show that Largest Bond remains NP-hard even for planar bipartite graphs, and it does not admit a constant-factor approximation algorithm, unless P = NP. We also show that Largest Bond and Largest st-Bond on graphs of clique-width w cannot be solved in time $f(w) \times n^{o(w)}$ unless the Exponential Time Hypothesis fails, but they can be solved in time $f(w) \times n^{O(w)}$. In addition, we show that both problems are fixed-parameter tractable when parameterized by the size of the solution, but they do not admit polynomial kernels unless NP $\subseteq$ coNP/poly.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases bond, cut, maximum cut, connected cut, FPT, treewidth, clique-width

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.12

Related Version A full version of the paper is available at `http://arxiv.org/abs/1910.01071`.

Funding Supported by Grant 2015/11937-9, São Paulo Research Foundation (FAPESP) and by Grant E-26/203.272/2017, Rio de Janeiro Research Foundation (FAPERJ) and by Grant 308689/2017-8, 425340/2016-3, 313026/2017-3, 422829/2018-8, 303726/2017-2, National Council for Scientific and Technological Development (CNPq).

Acknowledgements We thank the organizers of WoPOCA 2017 for the opportunity to bring together some of the co-authors of this paper.

[1] corresponding author

1 Introduction

Let $G = (V, E)$ be a simple, connected, undirected graph. A *disconnecting set* of G is a set of edges $F \subseteq E(G)$ whose removal disconnects G. The edge-connectivity of G is $\kappa'(G) = \min\{|F| : F \text{ is a disconnecting set of } G\}$. A cut $[S, T]$ of G is a partition of V into two subsets S and $T = V \setminus S$. The cut-set $\partial(S)$ of a cut $[S, T]$ is the set of edges that have one endpoint in S and the other endpoint in T; these edges are said to cross the cut. In a connected graph, each cut-set determines a unique cut. Note that every cut-set is a disconnecting set, but the converse is not true. An inclusion-wise minimal disconnecting set of a graph is called a *bond*. It is easy to see that every bond is a cut-set, but there are cut-sets that are not bonds. More precisely, a nonempty set of edges F of G is a bond if and only if F determines a cut $[S, T]$ of G such that $G[S]$ and $G[T]$ are both connected. Let $s, t \in V(G)$. An st-bond of G is a bond whose removal disconnects s and t.

In this paper, we are interested in the complexity aspects of the following problem.

Largest Bond
Instance: A graph $G = (V, E)$; a positive integer k.
Question: Is there a proper subset $S \subset V(G)$ such that $G[S]$ and $G[V \setminus S]$ are connected and $\lvert\partial(S)\rvert \geq k$?

We also consider Largest st-Bond, where given a graph $G = (V, E)$, vertices $s, t \in V(G)$, and a positive integer k, we are asked whether G has an st-bond of size at least k.

A minimum (maximum) cut of a graph G is a cut with cut-set of minimum (maximum) size. Every minimum cut is a bond, thus a minimum bond is also a minimum cut of G, and it can be found in polynomial time using the classical Edmonds–Karp algorithm [11]. Besides that, minimum st-bonds are well-known structures, since they are precisely the st-cuts involved in the Gomory-Hu trees [20].

Regarding bonds on planar graphs, a folklore theorem states that if G is a connected planar graph, then a set of edges is a cycle in G if and only if it corresponds to a bond in the dual graph of G [18]. Note that each cycle separates the faces of G into the faces in the interior of the cycle and the faces of the exterior of the cycle, and the duals of the cycle edges are exactly the edges that cross from the interior to the exterior [28]. Consequently, the girth of a planar graph equals the edge connectivity of its dual [4].

Although cuts and bonds are similar, computing the largest bond of a graph seems to be harder than computing its maximum cut. Maximum Cut is NP-hard in general [16], but becomes polynomial for planar graphs [21]. On the other hand, finding a longest cycle in a planar graph is NP-hard, implying that finding a largest bond of a planar multigraph (or of a simple edge-weighted planar graph) is NP-hard. In addition, it is well-known that if a simple planar graph is 3-vertex-connected, then its dual is a simple planar graph. In 1976, Garey, Johnson, and Tarjan [17] proved that the problem of establishing whether a 3-vertex-connected planar graph is Hamiltonian is NP-complete, thus, as also noted by Haglin and Venkatesan [22], finding the largest bond of a simple planar graph is also NP-hard, contrasting with the polynomial-time solvability of Maximum Cut on planar graphs.

From the point of view of parameterized complexity, it is well known that Maximum Cut can be solved in FPT time when parametrized by the size of the solution [25], and since every graph has a cut with at least half the edges [12], it follows that it has a linear kernel. Concerning approximation algorithms, a 1/2-approximation algorithm can be obtained by randomly partitioning the set vertices into two parts, which induces a cut-set whose expected size is at least half of the number of edges [26]. The best-known result is the seminal work of Goemans and Williamson [19], who gave a 0.878-approximation based on

semidefinite programming. This has the best approximation factor unless the Unique Games Conjecture fails [24]. To the best of our knowledge, there is no algorithmic study regarding the parameterized complexity of computing the largest bond of a graph as well as the approximability of the problem.

A closely related problem is the Connected Max Cut [23], which asks for a cut $[S, T]$ of a given a graph G such that $G[S]$ is connected, and that the cut-set $\partial(S)$ has size at least k. Observe that a bond induces a feasible solution of Connected Max Cut, but not the other way around, since $G[T]$ may be disconnected. Indeed, the size of a largest bond can be arbitrarily smaller than the size of the maximum connected cut; take, e.g., a star with n leaves. For Connected Max Cut on general graphs, there exists a $\Omega(1/\log n)$-approximation [15], where n is the number of vertices. Also, there is a constant-factor approximation with factor $1/2$ for graphs of bounded treewidth [31], and a polynomial-time approximation scheme for graphs of bounded genus [23].

Recently, Saurabh and Zehavi [30] considered a generalization of Connected Max Cut, named Multi-Node Hub. In this problem, given numbers l and k, the objective is to find a cut $[S, T]$ of G such that $G[S]$ is connected, $|S| = l$ and $|\partial(S)| \geq k$. They observed that the problem is $W[1]$-hard when parameterized on l, and gave the first parameterized algorithm for the problem with respect to the parameter k. We remark that the $W[1]$-hardness also holds for Largest Bond parameterized by $|S|$.

Since every nonempty bond determines a cut $[S, T]$ such that $G[S]$ and $G[T]$ are both connected, every bond of G has size at most $|E(G)| - |V(G)| + 2$. A graph G has a bond of size $|E(G)| - |V(G)| + 2$ if and only if $V(G)$ can be partitioned into two parts such that each part induces a tree. Such graphs are known as *Yutsis graphs*. The set of planar Yutsis graphs is exactly the dual class of Hamiltonian planar graphs. According to Aldred, Van Dyck, Brinkmann, Fack, and McKay [1], cubic Yutsis graphs appear in the quantum theory of angular momenta as a graphical representation of general recoupling coefficients. They can be manipulated following certain rules in order to generate the so-called summation formulae for the general recoupling coefficient (see [2, 10, 32]).

There are very few results about the largest bond size in general graphs. In 2008, Aldred, Van Dyck, Brinkmann, Fack, and McKay [1] showed that if a Yutsis graph is regular with degree 3, the partition of the vertex set from the largest bond will result in two sets of equal size. In 2015, Ding, Dziobiak and Wu [9] proved that any simple 3-connected graph G will have a largest bond with size at least $\frac{2}{17}\sqrt{\log n}$, where $n = |V(G)|$. In 2017, Flynn [13] verified the conjecture that any simple 3-connected graph G has a largest bond with size at least $\Omega(n^{\log_3 2})$ for a variety of graph classes including planar graphs.

In this paper, we complement the state of the art on the problem of computing the largest bond of a graph. Preliminarily, we observe that while Maximum Cut is trivial for bipartite graphs, Largest Bond remains NP-hard for such a class of graphs, and we also present a general reduction that allows us to observe that Largest Bond is NP-hard for several classes for which Maximum Cut is NP-hard. Using this framework, we are able to show that Largest Bond on graphs of clique-width w cannot be solved in time $f(w) \times n^{o(w)}$ unless the ETH fails. Moreover, we show that Largest Bond does not admit a constant-factor approximation algorithm, unless P = NP, and thus is asymptotically harder to approximate than Maximum Cut.

As for positive results, the main contributions of this work concern the parameterized complexity of Largest Bond. Using win/win approaches, we consider the strategy of preprocessing the input in order to bound the treewidth of the resulting instance. After that, by presenting a dynamic programming algorithm for Largest Bond parameterized by the

treewidth, we show that the problem is fixed-parameter tractable when parameterized by the size of the solution. Finally, we remark that Largest Bond and Largest st-Bond do not admit polynomial kernels, unless NP $\subseteq$ coNP/poly. Due to space, some proofs were omitted.

2 Intractability results

In this section, we discuss aspects of the hardness of computing the largest bond. Notice that Largest Bond is Turing reducible to Largest st-Bond. Therefore, the results presented in this section also holds for Largest st-Bond.

Although Maximum Cut is trivial for bipartite graphs, we first observe that the same does not apply to compute the largest bond. Since a connected planar graph is Eulerian if and only if its dual graph is bipartite; subdivision of edges does not increase the size of the largest bond; and to decide whether a 4-regular planar graph has a Hamiltonian cycle is NP-complete [29]. The following holds.

▶ **Theorem 1.** Largest Bond *is NP-complete for planar bipartite graphs.*

▶ **Theorem 2.** *Let G be a simple bipartite graph and $\ell \in \mathbb{N}$. To determine the largest bond $\partial(S)$ of G with $|S| = \ell$ is $W[1]$-hard with respect to ℓ.*

Next, we present a general framework for reducibility from Maximum Cut to Largest Bond, by defining a special graph operator ψ such that Maximum Cut on a graph class $\mathcal{F}$ is reducible to Largest Bond on the image of $\mathcal{F}$ via ψ. An interesting particular case occurs when $\mathcal{F}$ is closed under ψ (for instance, chordal graphs are closed under ψ).

▶ **Definition 3.** *Let G be a graph and let $n = V(G)$. The graph $\psi(G)$ is constructed as follows: (i) create n disjoint copies $G_1, G_2, \ldots, G_n$ of G; (ii) add vertices v_a and v_b; (iii) add an edge between v_a and v_b; (iv) add all possible edges between $V(G_1 \cup G_2 \cup \ldots \cup G_n)$ and $\{v_a, v_b\}$.*

▶ **Definition 4.** *A set of graphs $\mathcal{G}$ is closed under operator ψ if whenever $G \in \mathcal{G}$, then $\psi(G) \in \mathcal{G}$.*

From the fact that a graph G has a cut $[S, V(G) \setminus S]$ of size k if and only if $\psi(G)$ has a bond $\partial(S')$ of size at least $nk + n^2 + 1$, the following theorem holds.

▶ **Theorem 5.** Largest Bond *is NP-complete for any graph class $\mathcal{G}$ such that: $\mathcal{G}$ is closed under operator ψ; and* MaxCut *is NP-complete for graphs in $\mathcal{G}$.*

▶ **Corollary 6.** Largest Bond *is NP-complete for the following classes: chordal graphs; co-comparability graphs; P_5-free graphs.*

2.1 Algorithmic lower bound for clique-width parameterization

In the '90s, Courcelle, Makowsky, and Rotics [6] proved that all problems expressible in MS1-logic are fixed-parameter tractable when parameterized by the clique-width of a graph and the logical expression size. The applicability of this meta-theorem has made clique-width become one of the most studied parameters in parameterized complexity. However, although several problems are MS1-expressible, this is not the case with Maximum Cut.

In 2014, Fomin, Golovach, Lokshtanov and Saurabh [14] showed that Maximum Cut on a graph of clique-width w cannot be solved in time $f(w) \times n^{o(w)}$ for any function f of w unless Exponential Time Hypothesis (ETH) fails. Using operator ψ, we are able to extend this result to Largest Bond.

▶ **Lemma 7.** Largest Bond *on graphs of clique-width w cannot be solved in time $f(w) \times n^{o(w)}$ unless the ETH fails.*

Proof. Maximum Cut cannot be solved in time $f(w) \times n^{o(w)}$ on graphs of clique-width w, unless Exponential Time Hypothesis (ETH) fails [14]. Therefore, by the polynomial-time reduction presented in Theorem 5, it is enough to show that the clique-width of $\psi(G)$ is upper bounded by a linear function of the clique-width of G.

If G has clique-width w, then the disjoint union $H_1 = G_1 \oplus G_2 \oplus \ldots \oplus G_n$ has clique-width w. Suppose that all vertices in H_1 have label 1. Now, let H_2 be the graph isomorphic to a K_2 such that $V(H) = \{v_a, v_b\}$, and v_a, v_b are labeled with 2. In order to construct $\psi(G)$ from $H_1 \oplus H_2$ it is enough to apply the join $\eta(1, 2)$. Thus, $\psi(G)$ has clique-width equals w. ◀

2.2 Inapproximability

While the maximum cut of a graph has at least a constant fraction of the edges, the size of the largest bond can be arbitrarily smaller than the number of edges; take, e.g., a cycle on n edges, for which a largest bond has size 2. This discrepancy is also reflected on the approximability of the problems. Indeed, we show that Largest Bond is strictly harder to approximate than Maximum Cut. To simplify the presentation, we consider a weighted version of the problem in which edges are allowed to have weights 0 or 1; the hardness results will follow for the unweighted case as well. In the Binary Weighted Largest Bond, the input is given by a connected weighted graph H with weights $w : E(H) \to \{0, 1\}$. The objective is to find a bond whose total weight is maximum.

Let G be a graph on n vertices and whose maximum cut has size k. Next, we define the *G-edge embedding* operator ξ_G. Given a connected weighted graph H, the weighted graph $\xi_G(H)$ is constructed by replacing each edge $\{u, v\} \in E(H)$ with weight 1 by a copy of G, denoted by G_{uv}, whose edges have weight 1, and, for each vertex t of G_{uv}, new edges $\{u, t\}$ and $\{v, t\}$, both with weight 0.

We can also apply the G-edge embedding operation on the graph $\xi_G(H)$, then on $\xi_G(\xi_G(H))$, and so on. In what follows, for an integer $h \geq 0$, denote by $\xi_G^h(H)$ the graph resulting from the operation that receives a graph H and applies ξ_G successively h times. Notice that $\xi_G^h(H)$ can be constructed in $\mathcal{O}(|V(G)|^{h+1})$ time. For some j, $0 \leq j \leq h-1$, observe that an edge $\{u, v\} \in E(\xi_G^j(H))$ will be replaced by a series of vertices added in iterations $j+1, j+2, \ldots, h$. These vertices will be called the *descendants* of $\{u, v\}$, and will be denoted by V_{uv}.

Let K_2 be the graph composed of a single edge $\{u, v\}$, and consider the problem of finding a bond of $\xi_G(K_2)$ with maximum weight. Since edges connecting u or v have weight 0, one can assume that u and v are in different sides of the bond, and the problem reduces to finding a maximum cut of G. In other words, the operator ξ_G embeds an instance G of Maximum Cut into an edge $\{u, v\}$ of K_2.

This suggests the following strategy to solve an instance of Maximum Cut. For some constant integer $h \geq 1$, calculate $H = \xi_G^h(K_2)$, and obtain a bond F of H with maximum weight. Note that, to solve H, one must solve embedded instances of Maximum Cut in multiple levels simultaneously. For a level j, $1 \leq j \leq h-1$, each edge $\{u, v\} \in E(\xi_G^j(K_2))$ with weight 1 will be replaced by a graph G_{uv} which is isomorphic to G. In Lemma 9 below, we argue that F is such that either $V(G_{uv}) \cup \{u, v\}$ are all in the same side of the cut, or u and v are in distinct sides. In the latter case, the edges of F that separate u and v will induce a cut of G.

In the remaining of this section, we consider a constant integer $h \geq 0$. Then, we define $H^j = \xi_G^j(K_2)$ for every j, $0 \leq j \leq h$, and $H = H^h$. Also, we write $[S, T]$ to denote the cut induced by a bond F of H.

▶ **Definition 8.** *Let F be a bond of H with cut $[S, T]$. We say that an edge $\{u, v\} \in E(H^j)$ with weight 1 is* nice for F *if either*

- $|\{u, v\} \cap S| = 1$, *or*
- $(\{u, v\} \cup V_{uv}) \subseteq S$, *or*
- $(\{u, v\} \cup V_{uv}) \subseteq T$.

Also, we say that F is nice *if, for every j, $0 \leq j \leq h-1$, and every edge $\{u, v\} \in E(H^j)$ with weight 1, $\{u, v\}$ is nice for F.*

▶ **Lemma 9.** *There is a polynomial-time algorithm that receives a bond F, and finds a nice bond F' such that $w(F') = w(F)$.*

In the following, assume that F is a nice bond with cut $[S, T]$. Consider a level j, $0 \leq j \leq h$, and an edge $\{u, v\} \in E(H^j)$ with weight 1 such that $|\{u, v\} \cap S| = 1$. If $j < h$, then we define F_{uv} to be the subset of edges in F which are incident with some vertex of V_{uv}; if $j = h$, then we define $F_{uv} = \{\{u, v\}\}$. Note that, because F is nice, if $|\{u, v\} \cap S| \neq 1$, then no edge of F is incident with V_{uv}.

Suppose now that $|\{u, v\} \cap S| = 1$ for some edge $\{u, v\} \in E(H^j)$ with weight 1 and $0 \leq j \leq h-1$. In this case, F induces a cut-set of G_{uv}. Namely, define $\hat{S}_{uv} := S \cap V(G_{uv})$ and $\hat{T}_{uv} := T \cap V(G_{uv})$ and let $\hat{F}_{uv}$ be the cut-set of G_{uv} corresponding to cut $[\hat{S}_{uv}, \hat{T}_{uv}]$.

Observe that for distinct edges $\{u, v\}$ and $\{s, t\}$, it is possible that $|\hat{F}_{uv}| \neq |\hat{F}_{st}|$. We will consider bonds F for which all induced cut-sets $\hat{F}_{uv}$ have the same size.

▶ **Definition 10.** *Let ℓ be a positive integer. A bond F of H with cut $[S, T]$ is said to be* ℓ-uniform *if,* (i) *F is nice, and* (*ii*) *for every j, $0 \leq j \leq h-1$, and every edge $\{u, v\} \in E(H^j)$ with weight 1 such that $|\{u, v\} \cap S\}| = 1$, $|\hat{F}_{uv}| = \ell$.*

An ℓ-uniform bond induces a cut-set of G of size ℓ.

▶ **Lemma 11.** *Suppose F is an ℓ-uniform bond of H. One can find in polynomial time a cut-set L of G with $|L| = \ell$.*

Proof. Let u, v be the vertices of K_2 to which ξ_G was applied. Since F is ℓ-uniform, $|\hat{F}_{uv}| = \ell$. Note that $\hat{F}_{uv}$ induces a cut-set of size ℓ on G. ◀

In the opposite direction, a cut of G induces an ℓ-uniform bond of H.

▶ **Lemma 12.** *Suppose L is a cut-set of G with $|L| = \ell$. One can find in polynomial time an ℓ-uniform bond F of H with $w(F) = \ell^h$.*

Proof. For each $j \geq 0$, we construct a bond F^j of H^j. For $j = 0$, let F^0 be the set containing the unique edge of $H^0 = K_2$. Suppose now that we already constructed a bond F^{j-1} of H^{j-1}. For each edge $\{u, v\} \in F^{j-1}$, let L_{uv} be the set of edges of G_{uv} corresponding to L. Define $F^j := \cup_{\{u,v\} \in F^{j-1}} L_{uv}$. One can verify that indeed F^j is a bond of H^j, and that $w(F_j) = |L| \times w(F_{j-1}) = \ell^j$. ◀

▶ **Lemma 13.** *There is a polynomial-time algorithm that receives a bond F of H, and finds an ℓ-uniform bond F' of H such that $w(F') = \ell^h \geq w(F)$.*

▶ **Lemma 14.** *Let F^* be a bond of H with maximum weight. Then $w(F^*) = k^h$.*

Proof. We assume that F^* is ℓ-uniform such that $w(F^*) = \ell^h$ for some ℓ; if this is not the case, then use Lemma 13.

Since F^* is ℓ-uniform, using Lemma 12 one obtains a cut-set L of G with size ℓ, then $\ell \leq k$, and thus $w(F^*) \leq k^h$.

Conversely, let L be a cut-set of G with size k. Using Lemma 12 for L, we obtain a bond F of H with weight k^h, and thus $w(F^*) \geq k^h$. ◀

▶ **Lemma 15.** *If there exists a constant-factor approximation algorithm for* Weighted Largest Bond*, then $P = NP$.*

Proof. Consider a graph G whose maximum cut has size k. Construct graph H and obtain a bond F of H using an α-approximation, for some constant $0 < \alpha < 1$. Using the algorithm of Lemma 13, obtain an ℓ-uniform bond F' of H such that $w(F') = \ell^h \geq w(F)$. Using Lemma 14 and the fact that F' is an α-approximation, $\ell^h \geq \alpha \times k^h$. Using Lemma 11, one can obtain a cut-set L of G with size $\ell \geq \alpha^{\frac{1}{h}} k$.

For any constant ε, $0 < \varepsilon < 1$, we can set $h = \lceil \log_{1-\varepsilon} \alpha \rceil$, such that the cut-set L has size at least $\ell \geq (1-\varepsilon)k$. Since Maximum Cut is APX-hard, this implies P = NP. ◀

▶ **Theorem 16.** *If there exists a constant-factor approximation algorithm for* Largest Bond*, then $P = NP$.*

Proof. We show that if there exists an α-approximation algorithm for Largest Bond, for constant $0 < \alpha < 1$, then there is an $\alpha/2$-approximation algorithm for the Binary Weighted Largest Bond, so the theorem will follow from Lemma 15.

Let H be a weighted graph whose edge weights are all 0 or 1. Let m be the number of edges with weight 0, and let l be the weight of a bond of H with maximum weight. Assume $l \geq 2/\alpha$, as otherwise, one can find an optimal solution in polynomial time by enumerating sets of up to $2/\alpha$ edges.

Construct an unweighted graph G as follows. Start with a copy of H and, for each edge $\{u, v\} \in E(H)$ with weight 1, replace $\{u, v\} \in E(G)$ by m parallel edges. Finally, to obtain a simple graph, subdivide each edge of G. If F is a bond of G, then one can construct a bond F' of H by undoing the subdivision and removing the parallel edges. Each edge of F' has weight 1, with exception of at most m edges. Thus, $w(F') \geq (|F| - m)/m$.

Observe that an optimal bond of H induces a bond of G with size at least ml. Thus, if F is an α-approximation for G, then $|F| \geq \alpha ml$ and therefore

$$w(F') \geq \frac{\alpha ml - m}{m} = \alpha l - 1 \geq \alpha l - \alpha l/2 = \alpha l/2.$$

We conclude that F' is an $\alpha/2$-approximation for H. ◀

3 Algorithmic upper bounds for clique-width parameterization

Lemma 7 shows that Largest Bond on graphs of clique-width w cannot be solved in time $f(w) \times n^{o(w)}$ unless the ETH fails. Now, we show that given an expression tree of width at most w, Largest Bond can be solved in $f(w) \times n^{O(w)}$ time.

An expression tree $\mathcal{T}$ is irredundant if for any join node $\eta(i, j)$, the vertices labeled by i and j are not adjacent in the graph associated with its child. It was shown by Courcelle and Olariu [7] that every expression tree $\mathcal{T}$ of G can be transformed into an irredundant expression tree $\mathcal{T}$ of the same width in time linear in the size of $\mathcal{T}$. Therefore, without loss of generality, we can assume that $\mathcal{T}$ is irredundant.

Our algorithm is based on dynamic programming over the expression tree of the input graph. We first describe what we store in the tables corresponding to the nodes in the expression tree.

Given a w-labeled graph G, two connected components of G has the same *type* if they have the same set of labels. Thus, a w-labeled graph G has at most $2^w - 1$ types of connected components.

Now, for every node X_ℓ of $\mathcal{T}$, denote by G_{X_ℓ} the w-labeled graph associated with this node, and let $L_1(X_\ell), \ldots, L_w(X_\ell)$ be the sets of vertices of G_{X_ℓ} labeled with $1, \ldots, w$, respectively. We define a table where each entry is of the form $c[\ell, s_1, ..., s_w, r, e_1, ..., e_{2^w-1}, d_1, ..., d_{2^w-1}]$, such that: $0 \leq s_i \leq |L_i(X_\ell)|$ for $1 \leq i \leq w$; $0 \leq r \leq |E(G_{X_\ell})|$; $0 \leq e_i \leq \min\{2, |L_i(X_\ell)|\}$ for $1 \leq i \leq 2^w - 1$; and $0 \leq d_i \leq \min\{2, |L_i(X_\ell)|\}$ for $1 \leq i \leq 2^w - 1$.

Each entry of the table represents whether there is a partition V_1, V_2 of $V(G_{X_\ell})$ such that: $|V_1 \cap L_i(G_{X_\ell})| = s_i$; the cut-set of $[V_1, V_2]$ has size at least r; $G_{X_\ell}[V_1]$ has e_i connected components of type i; $G_{X_\ell}[V_2]$ has d_i connected components of type i, where $e_i = 2$ means that $G_{X_\ell}[V_1]$ has *at least* two connected components of type i. The same holds for d_i.

Notice that this table contains $f(w) \times n^{\mathcal{O}(w)}$ entries. If X_ℓ is the root node of $\mathcal{T}$ (that is, $G = G_{X_\ell}$), then the size of the largest bond of G is equal to the maximum value of r for which the table for X_ℓ contains a valid entry (true value), such that there are j and k such that $e_i = 0$, $e_j = 1$ for $1 \leq i, j \leq 2^w - 1$, $i \neq j$; and $d_i = 0$, $d_k = 1$ for $1 \leq i, k \leq 2^w - 1$, $i \neq k$.

It is easy to see that we store enough information to compute a largest bond. Note that a w-labeled graph is connected if and only if it has exactly one type of connected components and exactly one component of such a type.

Now we provide the details of how to construct and update such tables. The construction for introduce nodes of $\mathcal{T}$ is straightforward.

Relabel node: Suppose that X_ℓ is a relabel node $\rho(i, j)$, and let $X_{\ell'}$ be the child of X_ℓ. Then the table for X_ℓ contains a valid entry $c[\ell, s_1, ..., s_w, r, e_1, ..., e_{2^w-1}, d_1, ..., d_{2^w-1}]$ if and only if the table for $X_{\ell'}$ contains an entry $c[\ell', s'_1, ..., s'_w, r, e'_1, ..., e'_{2^w-1}, d'_1, ..., d'_{2^w-1}] = true$, where: $s_i = 0$; $s_j = s'_i + s'_j$; $s_p = s'_p$ for $1 \leq p \leq w$, $p \neq i, j$; $e_p = e'_p$ for any type that contain neither i nor j; $e_p = 0$ for any type that contains i; and for any type e_p that contains j, it holds that $e_p = \min\{2, e'_p + e'_q + e'_r\}$ where e'_q represent the set of labels $(C_p \setminus \{j\}) \cup \{i\}$, e'_r represent the set of labels $C_p \cup \{i\}$, and C_p is the set of labels associated to p. The same holds for $d_1, ..., d_{2^w-1}$.

Union node: Suppose that X_ℓ is a union node with children $X_{\ell'}$ and $X_{\ell''}$. It holds that $c[\ell, s_1, ..., s_w, r, e_1, ..., e_{2^w-1}, d_1, ..., d_{2^w-1}]$ equals true if and only if there are valid entries $c[\ell', s'_1, ..., s'_w, r', e'_1, ..., e'_{2^w-1}, d'_1, ..., d'_{2^w-1}]$ and $c[\ell'', s''_1, ..., s''_w, r'', e''_1, ..., e''_{2^w-1}, d''_1, ..., d''_{2^w-1}]$, having: $s_i = s'_i + s''_i$ for $1 \leq i \leq w$; $r' + r'' \geq r$; $e_k = \min\{2, e'_k + e''_k\}$, and $d_k = min\{2, d'_k + d''_k\}$ for $1 \leq k \leq 2^w - 1$.

Join node: Finally, let X_ℓ be a join node $\eta(i, j)$ with the child $X_{\ell'}$. Remind that since the expression tree is irredundant then the vertices labeled by i and j are not adjacent in the graph $G_{X_{\ell'}}$. Therefore, the entry $c[\ell, s_1, ..., s_w, r, e_1, ..., e_{2^w-1}, d_1, ..., d_{2^w-1}]$ equals true if and only if there is a valid entry $c[\ell', s_1, ..., s_w, r', e'_1, ..., e'_{2^w-1}, d'_1, ..., d'_{2^w-1}]$ where

$$r' + s_i \times (|L_j(X_{\ell'})| - s_j) + s_j \times (|L_i(X_{\ell'})| - s_i) \geq r,$$

and $e_p = e'_p$, case p is associated to a type that contains neither i nor j; $e_p = 1$, case p is associated to $C^{\ell'}_{i,j} \setminus \{i\}$, where $C^{\ell'}_{i,j}$ is the set of labels obtained by the union of the types of $G_{X_{\ell'}}$ with some connected component having either label i or label j; $e_p = 0$, otherwise. The same holds for $d_1, ..., d_{2^w-1}$.

The correctness of the algorithm follows from the description of the procedure. Since for each ℓ, there are $\mathcal{O}((n+1)^w \times m \times (3^{2^w-1})^2)$ entries, the running time of the algorithm is $f(w) \times n^{\mathcal{O}(w)}$. This algorithm together with Lemma 7 concludes the proof of the Theorem 17.

▶ **Theorem 17.** Largest Bond *cannot be solved in time* $f(w) \times n^{o(w)}$ *unless ETH fails, where* w *is the clique-width of the input graph. Moreover, given an expression tree of width at most* w, Largest Bond *can be solved in time* $f(w) \times n^{\mathcal{O}(w)}$.

In order to extend this result to Largest st-Bond, it is enough to observe that given a tree expression $\mathcal{T}$ of G with width w, it is easy to construct a tree expression $\mathcal{T}'$ with width equals $w+2$, where no vertex of $V(G)$ has the same label than either s or t. Let $w+1$ be the label of s, and let $w+2$ be the label of t. By fixing, for each ℓ, $s_{w+1} = |L_{w+1}(X_\ell)|$ and $s_{w+2} = 0$, one can solve Largest st-Bond in time $f(w) \times n^{\mathcal{O}(w)}$.

4 Bounding the treewidth of G

In the remainder of this paper we deal with our main problems: Largest Bond and Largest st-Bond parameterized by the size of the solution (k). Inspired by the principle of preprocessing the input to obtain a kernel, we consider the strategy of preprocessing the input in order to bound the treewidth of the resulting instance.

We start our analysis with Largest Bond.

▶ **Definition 18.** *A graph* H *is called a minor of a graph* G *if* H *can be formed from* G *by deleting edges, deleting vertices, and by contracting edges. For each vertex* v *of* H*, the set of vertices of* G *that are contracted into* v *is called a branch set of* H.

▶ **Lemma 19.** *Let* G *be a simple connected undirected graph, and* k *be a positive integer. If* G *contains* $K_{2,k}$ *as a minor then* G *has a bond of size at least* k.

Proof. Let H be a minor of G isomorphic to $K_{2,k}$. Since G is connected and each branch set of H induces a connected subgraph of G, from H it is easy to construct a bond of G of size at least k. ◀

Combined with Lemma 19, the following results show that, without loss of generality, our study on k-bonds can be reduced to graphs of treewidth $\mathcal{O}(k)$.

▶ **Lemma 20.** *[3] Every graph* $G = (V, E)$ *contains* $K_{2,k}$ *as a minor or has treewidth at most* $2k-2$.

▶ **Lemma 21.** *[3] There is a polynomial-time algorithm that either concludes that the input graph* G *contains* $K_{2,k}$ *as a minor, or outputs a tree-decomposition of* G *of width at most* $2k-2$.

From Lemma 19 and Lemma 21 it follows that there is a polynomial-time algorithm that either concludes that the input graph G has a bond of size at least k, or outputs a tree-decomposition of G of width at most $2k-2$.

4.1 The st-bond case

Let $S \subseteq V(G)$ and let $\partial(S)$ be a bond of a connected graph G. Recall that a block is a 2-vertex-connected subgraph of G which is inclusion-wise maximal, and a block-cut tree of G is a tree whose vertices represent the blocks and the cut vertices of G, and there is an edge in the block-cut tree for each pair of a block and a cut vertex that belongs to that

block. Then, $\partial(S)$ intersects at most one block of G. More precisely, for any two distinct blocks B_1 and B_2 of G, if $S \cap V(B_1) \neq \emptyset$ and $S \cap V(B_1) \neq V(B_1)$, then either $V(B_2) \subseteq S$, or $V(B_2) \subseteq V \setminus S$. Indeed, if this is not the case, then either $G[S]$ or $G[V \setminus S]$ would be disconnected. Thus, to solve Largest st-Bond, it is enough to consider, individually, each block on the path between s and t in the block-cut tree of G. Also, if a block is composed of a single edge, then it is a bridge in G, which is not a solution for the problem unless $k = 1$. Thus, we may assume without loss of generality that G is 2-vertex-connected.

▶ **Lemma 22.** *Let G be a 2-vertex-connected graph. For all $v \in V(G) \setminus \{s, t\}$, there is an sv-path and a tv-path which are internally disjoint.*

▶ **Lemma 23.** *Let G be a 2-vertex-connected graph. If G contains $K_{2,2k}$ as a minor, then there exists $S \subseteq V(G)$ such that $\partial(S)$ is a bond of size at least k.*

Proof. Let G be a graph containing a $K_{2,2k}$ as a minor. If $k = 1$, the statement holds trivially, thus assume $k \geq 2$. Also, since G is connected, one can assume that this minor was obtained by contracting or removing edges only, and thus its branch sets contain all vertices of G. Let A and B be the branch sets corresponding to first side of $K_{2,2k}$, and let $X_1, X_2, \ldots, X_{2k}$ be the remaining branch sets.

First, suppose that s and t are in distinct branch sets. If this is the case, then there exist distinct indices $a, b \in \{1, \ldots, 2k\}$ such that $s \in A \cup X_a$ and $t \in B \cup X_b$. Now observe that $G[A \cup X_a]$ and $G[B \cup X_b]$ are connected, which implies an st-bond with at least $2k - 1 \geq k$ edges. Now, suppose that s and t are in the same branch set. In this case, one can assume without loss of generality that $s, t \in A \cup X_{2k}$.

Define $U = A \cup X_{2k}$ and $Q = V(G) \setminus U$. Observe that $G[U]$ and $G[Q]$ are connected. Consider an arbitrary vertex v in the set Q. Since G is 2-vertex-connected, Lemma 22 implies that there exist an sv-path P_s and a tv-path P_t which are internally disjoint. Let P'_s and P'_t be maximal prefixes of P_s and P_t, respectively, whose vertices are contained in U.

We partition the set U into parts U_s and U_t such that $G[U_s]$ and $G[U_t]$ are connected. Since $G[U]$ is connected, there exists a tree T spanning U. Direct all edges of T towards s and partition U as follows. Every vertex in P'_s belongs to U_s and every vertex in P'_t belongs to U_t. For a vertex $u \notin V(P'_s \cup P'_t)$, let w be the first ancestor of u (accordingly to T) which is in $P'_s \cup P'_t$. Notice that w is well-defined since $u \in V(T)$ and the root of T is $s \in V(P'_s \cup P'_t)$. Then u belongs to U_s if $w \in V(P'_s)$, and u belongs to U_t if $w \in V(P'_t)$.

Observe that that there are at least $2k - 1$ edges between U and Q, and thus there are at least k edges between U_s and Q, or between U_t and Q. Assume the former holds, as the other case is analogous. It follows that $G[U_s]$ and $G[U_t \cup Q]$ are connected and induce a bond of G with at least k edges. ◀

Lemma 21 and Lemma 23 imply that there is an algorithm that either concludes that the input graph G has a bond of size at least k, or outputs a tree-decomposition of an equivalent instance G' of width $\mathcal{O}(k)$.

▶ **Corollary 24.** *Given a graph G, vertices $s, t \in V(G)$, and an integer k, there exists a polynomial-time algorithm that either concludes that G has an st-bond of size at least k or outputs a subgraph G' of G together with a tree decomposition of G' of width equals $\mathcal{O}(k)$, such that G' has an st-bond of size at least k if and only if G has an st-bond of size at least k.*

Proof. Find a block-cut tree of G in linear time [5], and let B_s and B_t be the blocks of G that contain s and t, respectively. Remove each block that is not in the path from B_s to B_t in the block-cut tree of G. Let G' be the remaining graph. For each block B of G', consider

the vertices s' and t' of B which are nearest to s and t, respectively. Using Lemmas 21 and 23 one can in polynomial time either conclude that B has an $s't'$-bond, in which case G is a yes-instance, or compute a tree decomposition of B with width at most $\mathcal{O}(k)$.

Now, construct a tree decomposition of G' as follows. Start with the union of the tree decompositions of all blocks of G'. Next, create a bag $\{u\}$ for each cut vertex u of G'. Finally, for each cut vertex u and any bag corresponding to a block B connected through u, add an edge between $\{u\}$ and one bag of the tree decomposition of B containing u. Note that this defines a tree decomposition of G' and that each bag has at most $\mathcal{O}(k)$ vertices. ◀

Note that since k-bonds are solutions for Connected Max Cut, the results presented in this section naturally apply to such a problem as well.

5 Taking the treewidth as parameter

In the following, given a tree decomposition $\mathcal{T}$, we denote by ℓ one node of $\mathcal{T}$ and by X_ℓ the vertices contained in the *bag* of ℓ. We assume w.l.o.g that $\mathcal{T}$ is a extended version of a *nice* tree decomposition (see [8]), that is, we assume that there is a special root node r such that $X_\ell = \emptyset$ and all edges of the tree are directed towards r and each node ℓ has one of the following five types: *Leaf* ; *Introduce vertex*; *Introduce edge*; *Forget vertex*; and *Join*. Moreover, define G_ℓ to be the subgraph of G which contains only vertices and edges that have been introduced in ℓ or in a descendant of ℓ.

The number of partitions of a set of k elements is the *k-th Bell number*, which we denote by $B(k)$ ($B(k) \leq k!$ [27]).

▶ **Theorem 25.** *Given a nice tree decomposition of G with width tw, one can find a bond of maximum size in time $2^{\mathcal{O}(tw \log tw)} \times n$ where n is the number of vertices of G.*

Proof. Let $\partial_G(U)$ be a bond of G, and $[U, V \setminus U]$ be the cut defined by such a bond. Set $S^\ell_U = U \cap X_\ell$. The removal of $\partial_G(U)$ partitions $G_\ell[U]$ into a set C^ℓ_U of connected components, and $G_\ell[V \setminus U]$ into a set $C^\ell_{V\setminus U}$ of connected components. Note that C^ℓ_U and $C^\ell_{V\setminus U}$ define partitions of S^ℓ_U and $X_\ell \setminus S^\ell_U$, denoted by ρ^ℓ_1 and ρ^ℓ_2 respectively, where the intersection of each connected component of C^ℓ_U with S^ℓ_U corresponds to one part of ρ^ℓ_1. The same holds for $C^\ell_{V\setminus U}$ with respect to $X_\ell \setminus S^\ell_U$ and ρ^ℓ_2.

We define a table for which an entry $c[\ell, S, \rho_1, \rho_2]$ is the size of a largest cut-set (partial solution) of the subgraph G_ℓ, where S is the subset of X_ℓ to the left part of the bond, $X_\ell \setminus S$ is the subset to the right part, and ρ_1, ρ_2 are the partitions of S and $X_\ell \setminus S$ representing, after the removal of the partial solution, the intersection with the connected components to the left and to the right, respectively. If there is no such a partial solution then $c[\ell, S, \rho_1, \rho_2] = -\infty$.

For the case that S is empty, two special cases may occur: either $U \cap V(G_\ell) = \emptyset$, in which case there are no connected components in C^ℓ_U, and thus $\rho_1 = \emptyset$; or C^ℓ_U has only one connected component which does not intersect X_ℓ, i.e., $\rho_1 = \{\emptyset\}$, this case means that the connected component in C^ℓ_U was completely forgotten. Analogously, we may have $\rho_2 = \emptyset$ and $\rho_2 = \{\emptyset\}$. Note that we do not need to consider the case $\{\emptyset\} \subsetneq \rho_i$ since it would imply in a disconnected solution. The largest bond of a connected graph G corresponds to the root entry $c[r, \emptyset, \{\emptyset\}, \{\emptyset\}]$.

To describe a dynamic programming algorithm, we only need to present the recurrence relation for each node type.

Leaf: In this case, $X_\ell = \emptyset$. There are a few combinations for ρ_1 and ρ_2: either $\rho_1 = \emptyset$, or $\rho_1 = \{\emptyset\}$, and either $\rho_2 = \emptyset$, or $\rho_2 = \{\emptyset\}$. Since for this case G_ℓ is empty, there can be no connected components, so having $\rho_1 = \emptyset$ and $\rho_2 = \emptyset$ is the only feasible choice.

$$c[\ell, S, \rho_1, \rho_2] = \begin{cases} 0 & \text{if } S = \emptyset,\ \rho_1 = \emptyset \text{ and } \rho_2 = \emptyset, \\ -\infty & \text{otherwise.} \end{cases}$$

Introduce vertex: We have only two possibilities in this case, either v is an isolated vertex to the left ($v \in S$) or it is an isolated vertex to the right ($v \notin S$). Thus, a partial solution on ℓ induces a partial solution on ℓ', excluding v from its part.

$$c[\ell, S, \rho_1, \rho_2] = \begin{cases} c[\ell', S \setminus \{v\}, \rho_1 \setminus \{\{v\}\}, \rho_2] & \text{if } \{v\} \in \rho_1, \\ c[\ell', S, \rho_1, \rho_2 \setminus \{\{v\}\}] & \text{if } \{v\} \in \rho_2, \\ -\infty & \text{if } \{v\} \notin \rho_1 \cup \rho_2. \end{cases}$$

Introduce edge: In this case, either the edge $\{u, v\}$ that is being inserted is incident with one vertex of each side, or the two endpoints are at the same side. In the former case, a solution on ℓ corresponds to a solution on ℓ' with the same partitions, but with value increased. In the latter case, edge $\{u, v\}$ may connect two connected components of a partial solution on ℓ'.

$$c[\ell, S, \rho_1, \rho_2] = \begin{cases} c[\ell', S, \rho_1, \rho_2] + 1 & \text{if } u \in S \text{ and } v \notin S \text{ or } u \notin S \text{ and } v \in S, \\ max_{\rho_1'}\{c[\ell', S, \rho_1', \rho_2]\} & \text{if } u \in S \text{ and } v \in S, \\ max_{\rho_2'}\{c[\ell', S, \rho_1, \rho_2']\} & \text{if } u \notin S \text{ and } v \notin S. \end{cases}$$

Here, ρ_1' spans over all refinements of ρ_1 such that the union of the parts containing u and v results in the partition ρ_1. The same holds for ρ_2'.

Forget vertex: In this case, either the forgotten vertex v is in the left side of the partial solution induced on ℓ, or is in the right side. Thus, v must be in the connected component which contains some part of ρ_1, or some part of ρ_2. We select the possibility that maximizes the value

$$c[\ell, S, \rho_1, \rho_2] = max_{\rho_1', \rho_2'}\{c[\ell', S \cup \{v\}, \rho_1', \rho_2], c[\ell', S, \rho_1, \rho_2']\}.$$

Here, ρ_1' spans over all partitions obtained from ρ_1 by adding v in some part of ρ_1 (if $\rho_1 = \{\emptyset\}$ then $\rho_1' = \{v\}$). The same holds for ρ_2'.

Join: This node represents the join of two subgraphs $G_{\ell'}$ and $G_{\ell''}$ and $X_\ell = X_{\ell'} = X_{\ell''}$. By counting the bond edges contained in $G_{\ell'}$ and in $G_{\ell''}$, each edge is counted at least once, but edges in X_ℓ are counted twice. Thus

$$c[\ell, S, \rho_1, \rho_2] = max\{c[\ell', S, \rho_1', \rho_2'] + c[\ell', S, \rho_1'', \rho_2'']\} - |\{\{u, v\} \in E, u \in S, v \in X_\ell \setminus S\}|.$$

In this case, we must find the best combination between the two children. Namely, for $i \in \{1, 2\}$, we consider combinations of ρ_i' with ρ_i'' which merge into ρ_i. If $\rho_i = \{\emptyset\}$ then either $\rho_i' = \{\emptyset\}$ and $\rho_i'' = \emptyset$; or $\rho_i' = \emptyset$ and $\rho_i'' = \{\emptyset\}$. Also, if $\rho_i = \emptyset$ then $\rho_i' = \emptyset$ and $\rho_i'' = \emptyset$.

The running time of the dynamic programming algorithm can be estimated as follows. The number of nodes in the decomposition is $\mathcal{O}(tw \times n)$ [8]. For each node ℓ, the parameters ρ_1 and ρ_2 induce a partition of X_ℓ; the number of partitions of X_ℓ is given by the corresponding Bell number, $B(|X_\ell|) \leq B(tw+1)$. Each such a partition ρ corresponds to a number of choice of parameter S that corresponds to a subset of the parts of ρ; thus the number of choices for S is not larger than $2^{|\rho|} \leq 2^{|X_\ell|} \leq 2^{tw+1}$. Therefore, we conclude that the table size is at most $\mathcal{O}(B(tw+1) \times 2^{tw} \times tw \times n)$. Since each entry can be computed in $2^{\mathcal{O}(tw \log tw)}$ time, the total complexity is $2^{\mathcal{O}(tw \log tw)} \times n$. The correctness of the recursive formulas is straightforward. ◀

The reason for the $2^{\mathcal{O}(tw \log tw)}$ dependence on treewidth is because we enumerate all partitions of a bag to check connectivity. However, one can obtain single exponential-time dependence by modifying the presented algorithm using techniques based on Gauss elimination, as described in [8, Chapter 11] for Steiner Tree.

▶ **Theorem 26.** Largest *st*-Bond *is fixed-parameter tractable when parameterized by treewidth.*

Proof. The solution of Largest *st*-Bond can be found by a dynamic programming as presented in Theorem 25 where we add s and t in all the nodes and we fix $s \in S$ and $t \notin S$. ◀

Finally, the following holds.

▶ **Corollary 27.** Largest Bond *and* Largest *st*-Bond *are fixed-parameter tractable when parameterized by the size of the solution, k.*

Proof. Follows from Lemma 19, Lemma 21, Corollary 24, Theorem 25 and Theorem 26. ◀

6 Infeasibility of polynomial kernels

An or-composition for Largest Bond parameterized by k can be done from the disjoint union of ℓ inputs, by selecting exactly one vertex of each input graph and contracting them into a single vertex. Now, let $(G_1, k, s_1, t_1), (G_2, k, s_2, t_2), \ldots, (G_\ell, k, s_\ell, t_\ell)$ be ℓ instances of Largest *st*-Bond parameterized by k. An or-composition for Largest *st*-Bond parameterized by k can be done from the disjoint union of $G_1, G_2, \ldots, G_\ell$, by contracting t_i, s_{i+1} into a single vertex, $1 \leq i \leq \ell - 1$, and setting $s = s_1$ and $t = t_\ell$.

▶ **Theorem 28.** Largest Bond *and* Largest *st*-Bond *do not admit polynomial kernel unless NP $\subseteq$ coNP/poly.*

References

1. Robert E. L. Aldred, Dries V. Dyck, Gunnar Brinkmann, Veerle Fack, and Brendan D McKay. Graph structural properties of non-Yutsis graphs allowing fast recognition. *Discrete Applied Mathematics*, 157(2):377–386, 2009.
2. Lawrence Christian Biedenharn and James D Louck. *The Racah-Wigner algebra in quantum theory*. Addison-Wesley, 1981.
3. Hans L Bodlaender, Jan Van Leeuwen, Richard Tan, and Dimitrios M Thilikos. On interval routing schemes and treewidth. *Information and Computation*, 139(1):92–109, 1997.
4. Jung Jin Cho, Yong Chen, and Yu Ding. On the (co)girth of a connected matroid. *Discrete Applied Mathematics*, 155(18):2456–2470, 2007.
5. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 2001. URL: `https://books.google.co.in/books?id=NLngYyWFl_YC`.

6 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.

7 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

8 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015.

9 Guoli Ding, Stan Dziobiak, and Haidong Wu. Large-or-Minors in 3-Connected Graphs. *Journal of Graph Theory*, 82(2):207–217, 2016.

10 Dries V. Dyck and Veerle Fack. On the reduction of Yutsis graphs. *Discrete Mathematics*, 307(11):1506–1515, 2007. The Fourth Caracow Conference on Graph Theory.

11 Jack Edmonds and Richard M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM*, 19(2):248–264, April 1972.

12 Paul Erdös. On some extremal problems in graph theory. *Israel Journal of Mathematics*, 3(2):113–116, 1965.

13 Melissa Flynn. *The Largest Bond in 3-Connected Graphs*. PhD thesis, The University of Mississippi, 2017.

14 Fedor V Fomin, Petr A Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM Journal on Computing*, 43(5):1541–1563, 2014.

15 Rajiv Gandhi, Mohammad T. Hajiaghayi, Guy Kortsarz, Manish Purohit, and Kanthi Sarpatwar. On maximum leaf trees and connections to connected maximum cut problems. *Information Processing Letters*, 129:31–34, 2018.

16 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

17 Michael R. Garey, David S. Johnson, and Robert E. Tarjan. The Planar Hamiltonian Circuit Problem is NP-Complete. *SIAM Journal on Computing*, 5(4):704–714, 1976.

18 Christopher D. Godsil and Gordon F. Royle. *Algebraic Graph Theory*. Graduate texts in mathematics. Springer, 2001.

19 Michel X. Goemans and David P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J. ACM*, 42(6):1115–1145, 1995.

20 Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

21 F. Hadlock. Finding a Maximum Cut of a Planar Graph in Polynomial Time. *SIAM Journal on Computing*, 4(3):221–225, 1975.

22 D. J. Haglin and S. M. Venkatesan. Approximation and intractability results for the maximum cut problem and its variants. *IEEE Transactions on Computers*, 40(1):110–113, January 1991. doi:10.1109/12.67327.

23 Mohammad Taghi Hajiaghayi, Guy Kortsarz, Robert MacDavid, Manish Purohit, and Kanthi Sarpatwar. Approximation Algorithms for Connected Maximum Cut and Related Problems. In *In Proceedings of the 23rd European Symposium on Algorithms*, pages 693–704, 2015.

24 S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal Inapproximability Results for MAX-CUT and Other 2-Variable CSPs? *SIAM Journal on Computing*, 37(1):319–357, 2007.

25 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31(2):335–354, 1999.

26 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

27 Andrew M Odlyzko. Asymptotic enumeration methods. *Handbook of combinatorics*, 2(1063):1229, 1995.

28 James G. Oxley. *Matroid theory*, volume 3. Oxford University Press, USA, 2006.

29 Christophe Picouleau. Complexity of the hamiltonian cycle in regular graph problem. *Theoretical Computer Science*, 131(2):463–473, 1994.

30 Saket Saurabh and Meirav Zehavi. Parameterized Complexity of Multi-Node Hubs. In *13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, pages 8:1–8:14, 2019.

31 Xiangkun Shen, Jon Lee, and Viswanath Nagarajan. Approximating graph-constrained max-cut. *Mathematical Programming*, 172(1):35–58, November 2018.

32 A. P. Yutsis, V. V. Vanagas, and I. B. Levinson. *Mathematical apparatus of the theory of angular momentum*. Israel program for scientific translations, 1962.

Parameterized Algorithms for Maximum Cut with Connectivity Constraints

Hiroshi Eto
Kyushu University, Fukuoka, Japan
h-eto@econ.kyushu-u.ac.jp

Tesshu Hanaka
Chuo University, Tokyo, Japan
hanaka.91t@g.chuo-u.ac.jp

Yasuaki Kobayashi
Kyoto University, Kyoto, Japan
kobayashi@iip.ist.i.kyoto-u.ac.jp

Yusuke Kobayashi
Kyoto University, Kyoto, Japan
yusuke@kurims.kyoto-u.ac.jp

Abstract

We study two variants of Maximum Cut, which we call Connected Maximum Cut and Maximum Minimal Cut, in this paper. In these problems, given an unweighted graph, the goal is to compute a maximum cut satisfying some connectivity requirements. Both problems are known to be NP-complete even on planar graphs whereas Maximum Cut on planar graphs is solvable in polynomial time. We first show that these problems are NP-complete even on planar bipartite graphs and split graphs. Then we give parameterized algorithms using graph parameters such as clique-width, tree-width, and twin-cover number. Finally, we obtain FPT algorithms with respect to the solution size.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Maximum cut, Parameterized algorithm, NP-hardness, Graph parameter

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.13

Related Version A full version of the paper is available at `https://arxiv.org/abs/1908.03389`.

Acknowledgements We thank Akitoshi Kawamura and Yukiko Yamauchi for giving an opportunity to discuss in the Open Problem Seminar at Kyushu University. This work is partially supported by JST CREST JPMJCR1401 and JSPS KAKENHI Grant Numbers JP17H01788, JP18H06469, JP16K16010, JP17K19960, and JP18H05291.

1 Introduction

Maximum Cut is one of the most fundamental problems in theoretical computer science. Given a graph and an integer k, the problem asks for a subset of vertices such that the number of edges having exactly one endpoint in the subset is at least k. This problem was shown to be NP-hard in Karp's seminal work [34]. To overcome this intractability, a lot of researches have been done from various view points, such as approximation algorithms [25], fixed-parameter tractability [40], and special graph classes [7, 9, 20, 28, 29, 37].

In this paper, we study two variants of Maximum Cut, called Connected Maximum Cut and Maximum Minimal Cut. A cut $(S, V \setminus S)$ is *connected* if the subgraph of G induced by S is connected. Given a graph $G = (V, E)$ and an integer k, Connected Maximum Cut is the problem to determine whether there is a connected cut $(S, V \setminus S)$ of size at least k . This problem is defined in [30] and known to be NP-complete even on planar graphs [31] whereas Maximum Cut on planar graphs is solvable in polynomial time [29, 37].

Table 1 The summary of the computational complexity of Maximum Cut and its variants. MC, CMC, and MMC stand for Maximum Cut, Connected Maximum Cut, and Maximum Minimal Cut.

	Graph Class			Parameter				kernel
	Split	Bipartite	Planar	cw	tw	tc	k	k
MC	NP-c [7]	P [trivial]	P [29, 37]	$n^{O(\mathrm{cw})}$ [22]	2^{tw} [7]	2^{tc} [23]	1.2418^k [40]	$O(k)$ [30, 36]
CMC	NP-c [Th. 5]	NP-c [Th. 3]	NP-c [31]	$n^{O(\mathrm{cw})}$ [Th. 17]	3^{tw} [Th. 12]	$2^{2^{\mathrm{tc}}+\mathrm{tc}}$ [Th. 18]	9^k [Th. 22]	No [Th. 24]
MMC	NP-c [Th. 6]	NP-c [Th. 4]	NP-c [30]	$n^{O(\mathrm{cw})}$ [Th. 17]	4^{tw} [Th. 11]	$2^{\mathrm{tc}}3^{2^{\mathrm{tc}}}$ [Th. 19]	$2^{O(k^2)}$ [Th. 21]	No [Th. 24]

Suppose G is connected. We say that a cut $(S, V \setminus S)$ of G is *minimal* if there is no another cut of G whose cutset properly contains the cutset of $(S, V \setminus S)$, where the cutset of a cut is the set of edges between different parts. We can also define minimal cuts for disconnected graphs (See Section 2). Maximum Minimal Cut is the following problem: given a graph $G = (V, E)$ and an integer k, determine the existence of a minimal cut $(S, V \setminus S)$ of size at least k. This type of problems, finding a maximum minimal (or minimum maximal) solution on graphs such as Maximum Minimal Vertex Cover [8, 46], Maximum Minimal Dominating Set [1], Maximum Minimal Edge Cover [35], Maximum Minimal Separator [32], Minimum Maximal matching [24, 45], and Minimum Maximal Independent Set [18], has been long studied.

As a well-known fact, a cut $(S, V \setminus S)$ is minimal if and only if both subgraphs induced by S and $V \setminus S$ are connected when the graph is connected [19]. Therefore, a minimal cut is regarded as a *two-sided* connected cut, while a connected cut is a *one-side* connected cut[1]. Haglin and Venkatean [30] showed that deciding if the input graph has a two-sided connected cut (i.e., a minimal cut) of size at least k is NP-complete even on triconnected cubic planar graphs. This was shown by the fact that for any two-sided connected cut on a connected planar graph G, the cutset corresponds to a cycle on the dual graph of G and vise versa. Hence, the problem is equivalent to the longest cycle problem on planar graphs [30]. Recently, Chaourar proved that Maximum Minimal Cut can be solved in polynomial time on series parallel graphs and graphs without $K_5 \setminus e$ as a minor in [12, 13].

Even though there are many important applications of Connected Maximum Cut and Maximum Minimal Cut such as image segmentation [44], forest planning [11], and computing a market splitting for electricity markets [26], the known results are much fewer than those for Maximum Cut due to the difficult nature of simultaneously maximizing its size and handling the connectivity of a cut.

1.1 Our contribution

Our contribution is summarized in Table 1. We prove that both Connected Maximum Cut and Maximum Minimal Cut are NP-complete even on planar bipartite graphs. Interestingly, although Maximum Cut can be solved in polynomial time on planar graphs [29, 37] and

[1] In [12, 13, 30], the authors used the term "connected cut" for two-sided connected cut. In this paper, however, we use "minimal cut" for two-sided connected cut and "connected cut" for one-sided connected cut for distinction.

bipartite graphs, both problems are intractable even on the intersection of these tractable classes. We also show that the problems are NP-complete on split graphs.

To tackle to this difficulty, we study both problems from the perspective of the parameterized complexity. We give $O^*(\mathrm{tw}^{O(\mathrm{tw})})$-time algorithms for both problems[2], where tw is the tree-width of the input graph. Moreover, we can improve the running time using the rank-based approach [3] to $O^*(c^{\mathrm{tw}})$ for some constant c and using the Cut & Count technique [17] to $O^*(3^{\mathrm{tw}})$ for Connected Maximum Cut and $O^*(4^{\mathrm{tw}})$ for Maximum Minimal cut with randomization. Let us note that our result generalizes the polynomial time algorithms for Maximum Minimal Cut on series parallel graphs and graphs without $K_5 \setminus e$ as a minor due to Chaourar [12, 13] since such graphs are tree-width bounded [42].

Based on these algorithms, we give $O^*(2^{k^{O(1)}})$-time algorithms for both problems. For Connected Maximum Cut, we also give a randomized $O^*(9^k)$-time algorithm. As for polynomial kernelization, we can observe that Connected Maximum Cut and Maximum Minimal Cut admit no polynomial kernel when parameterized by solution size k under a reasonable complexity assumption (see, Theorem 24).

We also consider different structural graph parameters. We design XP-algorithms for both problems when parameterized by clique-width cw. Also, we give $O^*(2^{2^{\mathrm{tc}}+\mathrm{tc}})$-time and $O^*(2^{\mathrm{tc}}3^{2^{\mathrm{tc}}})$-time FPT algorithms for Connected Maximum Cut and Maximum Minimal Cut, respectively, where tc is the minimum size of a twin-cover of the input graph.

1.2 Related work

Maximum Cut is a classical graph optimization problem and there are many applications in practice. The problem is known to be NP-complete even on split graphs, tripartite graphs, co-bipartite graphs, undirected path graphs [7], unit disc graphs [20], and total graphs [28]. On the other hand, it is solvable in polynomial time on bipartite graphs, planar graphs [29, 37], line graphs [28], and proper interval graphs [9]. For the optimization version of Maximum Cut, there is a 0.878-approximation algorithm using semidefinite programming [25]. As for parameterized complexity, Maximum Cut is FPT [40] and has a linear kernel [30, 36] when parameterized by the solution size k. Moreover, the problem is FPT when parameterized by tree-width [7] and twin-cover number [23]. Fomin et al. [22] proved that Maximum Cut is W[1]-hard but XP when parameterized by clique-width.

Connected Maximum Cut was proposed in [30]. The problem is a connected variant of Maximum Cut as with Connected Dominating Set [27] and Connected Vertex Cover [15]. Hajiaghayi et al. [31] showed that the problem is NP-complete even on planar graphs whereas it is solvable in polynomial time on bounded treewidth graphs. For the optimization version of Connected Maximum Cut, they proposed a polynomial time approximation scheme (PTAS) on planar graphs and more generally on bounded genus graphs and an $\Omega(1/\log n)$-approximation algorithm on general graphs.

Maximum Minimal Cut was considered in [30] and shown to be NP-complete on planar graphs. Recently, Chaourar proved that the problem can be solved in polynomial time on series parallel graphs [12] and graphs without $K_5 \setminus e$ as a minor [13].

As another related problem, Multi-Node Hubs was proposed by Saurabh and Zehavi [43]: Given a graph $G = (V, E)$ and two integers k, p, determine whether there is a connected cut of size at least k such that the size of the connected part is *exactly* p. They proved that Multi-Node Hubs is W[1]-hard with respect to p, but solvable in time $O^*(2^{2^{O(k)}})$. As an immediate corollary of their result, we can solve Connected Maximum Cut in time $O^*(2^{2^{O(k)}})$ by solving Multi-Node Hubs for each $0 \leq p \leq n$.

[2] The $O^*(\cdot)$ notation suppresses polynomial factors in the input size.

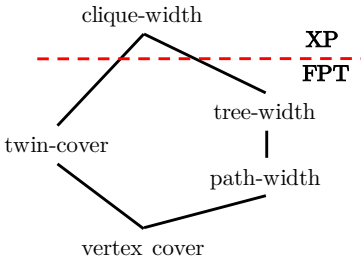


Figure 1 Graph parameters and the parameterized complexity of Maximum Cut, Connected Maximum Cut, and Maximum Minimal Cut. Connections between two parameters imply the above one is bounded by some function in the below one.

▶ **Proposition 1** ([43]). Connected Maximum Cut *can be solved in time* $O^*(2^{2^{O(k)}})$.

In this paper, we improve the running time in Proposition 1 by giving an $O^*(2^{O(k)})$-time algorithm for Connected Maximum Cut in Section 4.4.

2 Preliminaries

In this paper, we use the standard graph notations. Let $G = (V, E)$ be an undirected graph. For $V' \subseteq V$, we denote by $G[V']$ the subgraph of G induced by V'. We denote the open neighbourhood of v by $N(v)$ and the closed neighbourhood by $N[v]$.

A *cut* of G is a pair $(S, V \setminus S)$ for some subset $S \subseteq V$. Note that we allow S (and $V \setminus S$) to be empty. For simplicity, we sometimes denote a cut $(S, V \setminus S)$ by (S_1, S_2) where $S_1 = S$ and $S_2 = V \setminus S$. If the second part $V \setminus S$ of a cut is clear from the context, we may simply denote $(S, V \setminus S)$ by S. The *cutset* of S, denoted by $\delta(S)$, is the set of *cut edges* between S and $V \setminus S$. The size of a cut is defined as the number of edges in its cutset (i.e., $|\delta(S)|$). A cut S is *connected* if the subgraph induced by S is connected. We say that a cutset is *minimal* if there is no non-empty proper cutset of it. A cut is *minimal* if its cutset is minimal. It is well known that for every minimal cut S, $G[S]$ and $G[V \setminus S]$ are connected when G is connected [19]. If G has two or more connected component, every cutset of a minimal cut of G corresponds to a minimal cutset of its connected component. Therefore, throughout the paper, except in Theorem 24, we assume the input graph G is connected. Let p be a predicate. We define the function $[p]$ as follows: if p is true, then $[p] = 1$, otherwise $[p] = 0$.

2.1 Graph parameters

In this paper, we use the following graph parameters: clique-width $\mathrm{cw}(G)$, tree-width $\mathrm{tw}(G)$, path-width $\mathrm{pw}(G)$, twin-cover number $\mathrm{tc}(G)$, and vertex cover number $\mathrm{vc}(G)$. The definitions of them can be found in [14, 16, 23]. For clique-width, tree-width, path-width, twin-cover number, and vertex cover number, the following relations hold.

▶ **Proposition 2** ([6, 14, 23]). *For any graph G, the following inequalities hold:* $\mathrm{cw}(G) \leq 2^{\mathrm{tw}(G)+1} + 1$, $\mathrm{cw}(G) \leq \mathrm{pw}(G) + 1$, $\mathrm{tw}(G) \leq \mathrm{pw}(G) \leq \mathrm{vc}(G)$, $\mathrm{cw}(G) \leq 2^{\mathrm{tc}(G)} + \mathrm{tc}(G)$, *and* $\mathrm{tc}(G) \leq \mathrm{vc}(G)$.

From Proposition 2, we can illustrate the parameterized complexity of Maximum Cut, Connected Maximum Cut, and Maximum Minimal Cut associated with graph parameters in Figure 1.

3 Computational Complexity on Graph Classes

In this section, we prove that Connected Maximum Cut and Maximum Minimal Cut are NP-complete on planar bipartite graphs and split graphs.

3.1 Planar bipartite graphs

▶ **Theorem 3.** Connected Maximum Cut *is NP-complete on planar bipartite graphs.*

▶ **Theorem 4.** Maximum Minimal Cut *is NP-complete on planar bipartite subcubic graphs.*

Proof. We give a reduction from Maximum Minimal Cut on planar cubic graphs, which is known to be NP-complete [30]. Given a connected planar cubic graph $G = (V, E)$, we split each edge $e = \{u, w\} \in E$ by a vertex v_e, that is, we introduce a new vertex v_e and replace e by $\{u, v_e\}$ and $\{w, v_e\}$. Let $V_E = \{v_e \mid e \in E\}$ and $G' = (V \cup V_E, E')$ the reduced graph. Since we split each edge by a vertex and G is a planar cubic graph, G' is not only planar but also bipartite and subcubic. In the following, we show that there is a minimal cut of size at least k in G if and only if so is in G'. We can assume that $k > 2$.

Let (S_1, S_2) be a minimal cut of G. We construct a cut (S'_1, S'_2) of G' with $S_i \subseteq S'_i$ for $i = 1, 2$. For each edge $e \in E$, we add v_e to S'_2 if both endpoints of e are contained in S_2, and otherwise add v_e to S'_1. Recall that a cut is minimal if and only if both sides of the cut induce connected subgraphs. We claim that both $G'[S'_1]$ and $G'[S'_2]$ are connected. To see this, consider vertices $u, v \in S_1$. As $G[S_1]$ is connected, there is a path between u and v in $G[S_1]$. By the construction of S'_1, every vertex of the path is in S'_1 and for every edge e in the path, we have $v_e \in S'_1$. Therefore, there is a path between u and v in $G'[S'_1]$. Moreover, for every $v_e \in S'_1$, at least one endpoint of e is in S'_1. Hence, $G'[S'_1]$ is connected. Symmetrically, we can conclude that $G'[S'_2]$ is connected. Moreover, for each $e = \{u, w\}$ with $u \in S'_1$ and $w \in S'_2$, $\{v_e, w\}$ is a cut edge in G'. Therefore, (S'_1, S'_2) is a minimal cut of size at least k.

Conversely, we are given a minimal cut (S'_1, S'_2) of size k. We let $S_i = S'_1 \cap V$ for $i = 1, 2$. For each $e = \{u, v\}$, we can observe that $v_e \in S'_i$ if $u, w \in S'_i$ due to the connectivity of S'_i and $k > 2$. This means that an edge $\{u, v_e\}$ (or $\{w, v_e\}$) contributes to the cut if and only if exactly one of u and w is contained in S'_1 (and hence S_1), that is, the edge e contributes to the cut (S_1, S_2) in G. Therefore, the size of the cut (S_1, S_2) is at least k. Moreover, u and v are connected by a path through v_e in $G'[S'_i]$ if and only if u and v are contained in S_i and adjacent to each other in $G[S_i]$. Hence, $G[S_i]$ is connected for each $i = 1, 2$, and the theorem follows. ◀

3.2 Split graphs

▶ **Theorem 5.** Connected Maximum Cut *is NP-complete on split graphs.*

Proof. We reduce the following problem called Exact 3-cover, which is known to be NP-complete: Given a set $X = \{x_1, x_2, \ldots, x_{3n}\}$ and a family $\mathcal{F} = \{F_1, F_2, \ldots, F_m\}$, where each $F_i = \{x_{i_1}, x_{i_2}, x_{i_3}\}$ has three elements of X, the objective is to find a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ such that every element in X is contained in exactly one of the subsets $\mathcal{F}'$. By making some copies of 3-element sets if necessary, we may assume that $|\{F \in \mathcal{F} \mid x \in F\}| \geq 3(n + 2)$ for each $x \in X$, which implies that m is sufficiently large compared to n.

Given an instance of Exact 3-cover with $|\{F \in \mathcal{F} \mid x \in F\}| \geq 3(n + 2)$ for each $x \in X$, we construct an instance of Connected Maximum Cut in a split graph as follows. We introduce m vertices $u_1, u_2, \ldots, u_m$, where each u_i corresponds to F_i, and introduce $m - 2n$ vertices $u_{m+1}, u_{m+2}, \ldots, u_{2(m-n)}$. Let $U := \{u_1, u_2, \ldots, u_{2(m-n)}\}$. For

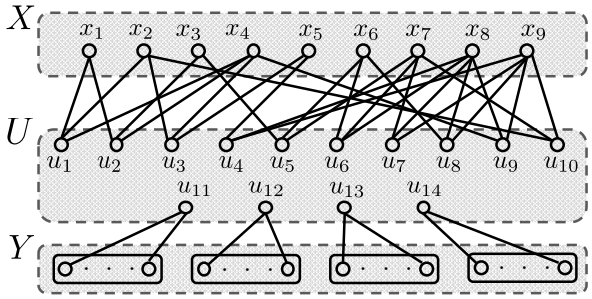


Figure 2 An instance of Connected Maximum Cut on split graphs reduced from an instance of Exact 3-cover where $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\}$ and $\mathcal{F} = \{\{x_1, x_2, x_3\}, \{x_1, x_3, x_4\}, \{x_2, x_4, x_5\}, \{x_5, x_8, x_9\}, \{x_3, x_6, x_7\}, \{x_6, x_7, x_8\}, \{x_7, x_8, x_9\}, \{x_6, x_8, x_9\}, \{x_4, x_8, x_9\}, \{x_2, x_7, x_9\}\}$.

$i = m+1, m+2, \ldots, 2(m-n)$, introduce a vertex set Y_i of size M, where M is a sufficiently large integer compared to n (e.g. $M = 3n+1$). Now, we construct a graph $G = (U \cup X \cup Y, E)$, where $Y := \bigcup_{m+1 \le i \le m-2n} Y_i$, $E_U := \{\{u, u'\} \mid u, u' \in U, u \neq u'\}$, $E_X := \{\{u_i, x_j\} \mid 1 \le i \le m, 1 \le j \le 3n, x_j \in F_i\}$, $E_Y := \{\{u_i, y\} \mid m+1 \le i \le 2(m-n), y \in Y_i\}$, and $E := E_U \cup E_X \cup E_Y$. Then, G is a split graph in which U induces a clique and $X \cup Y$ is an independent set. We now show the following claim.

▷ Claim. The original instance of Exact 3-cover has a solution if and only if the obtained graph G has a connected cut of size at least $(m-n)^2 + 3m - 3n + (m-2n)M$.

Proof. Suppose that the original instance of Exact 3-cover has a solution $\mathcal{F}'$. Then $S := \{u_i \mid F_i \in \mathcal{F}'\} \cup \{u_i \mid m+1 \le i \le 2(m-n)\} \cup X$ is a desired connected cut, because $|\delta(S) \cap E_U| = (m-n)^2$, $|\delta(S) \cap E_X| = \sum_{i=1}^{m} |F_i| - |X| = 3m - 3n$, and $|\delta(S) \cap E_Y| = (m-2n)M$.

Conversely, suppose that the obtained instance of Connected Maximum Cut has a connected cut S such that $|\delta(S)| \ge (m-n)^2 + 3m - 3n + (m-2n)M$. Since $|\delta(S) \cap E_U| \le (m-n)^2$, $|\delta(S) \cap E_X| \le 3m$, and $|\delta(S) \cap E_Y| \le |S \cap \{u_{m+1}, \ldots, u_{2(m-n)}\}| \cdot M$, we obtain $|S \cap \{u_{m+1}, \ldots, u_{2(m-n)}\}| = m - 2n$, that is, $\{u_{m+1}, \ldots, u_{2(m-n)}\} \subseteq S$. Let $t = |S \cap \{u_1, \ldots, u_m\}|$, $X_0 = \{x \in X \mid N(x) \cap S = \emptyset\}$ the vertices in X that has no neighbor in S, $X_{\mathrm{all}} = \{x \in X \mid N(x) \subseteq S\}$ the vertices in X whose neighbor is entirely included in S, and $X_{\mathrm{part}} = X \setminus (X_0 \cup X_{\mathrm{all}})$ all the other vertices in X. Recall that every element in X is contained in at least $3(n+2)$ subsets of $\mathcal{F}$. Then, since $|\delta(S) \cap E_U| = (m-t)(m-2n+t) = (m-n)^2 - (t-n)^2$, $|\delta(S) \cap E_X| \le |E_X| - |X_{\mathrm{part}}| - |\delta(X_0)| \le 3m - (3n - |X_{\mathrm{all}}| - |X_0|) - 3(n+2)|X_0|$, $|\delta(S) \cap E_Y| \le (m-2n)M$, and $|\delta(S)| \ge (m-n)^2 + 3m - 3n + (m-2n)M$, we obtain

$$|X_{\mathrm{all}}| - (3n+5)|X_0| - (t-n)^2 \ge 0. \tag{1}$$

By counting the number of edges between $S \cap \{u_1, u_2, \ldots, u_m\}$ and X, we obtain $3t \ge |\delta(X_{\mathrm{all}})| \ge 3(n+2)|X_{\mathrm{all}}|$, which shows that $t \ge (n+2)|X_{\mathrm{all}}|$. If $|X_{\mathrm{all}}| \ge 1$, then $t \ge (n+2)|X_{\mathrm{all}}| \ge n + 2|X_{\mathrm{all}}|$, and hence $|X_{\mathrm{all}}| - 3(n+5)|X_0| - (t-n)^2 \le |X_{\mathrm{all}}| - (2|X_{\mathrm{all}}|)^2 < 0$, which contradicts (1). Thus, we obtain $|X_{\mathrm{all}}| = 0$, and hence we have $t = n$ and $X_0 = \emptyset$ by (1). Therefore, $\mathcal{F}' := \{F_i \mid 1 \le i \in m,\ u_i \in S\}$ satisfies that $|\mathcal{F}'| = n$ and $\bigcup_{F \in \mathcal{F}'} F = X$. This shows that $\mathcal{F}'$ is a solution of the original instance of Exact 3-cover. ◁

This shows that Exact 3-cover is reduced to Connected Maximum Cut in split graphs, which completes the proof. ◀

▶ **Theorem 6.** Maximum Minimal Cut *is NP-complete on split graphs.*

4 Parameterized Complexity

4.1 Tree-width

In this section, we give FPT algorithms for Connected Maximum Cut and Maximum Minimal Cut parameterized by tree-width. In particular, we design $O^*(c^{\mathrm{tw}})$-time algorithms where c is some constant.

4.1.1 $O^*(\mathrm{tw}^{O(\mathrm{tw})})$-algorithm

We design an $O^*(\mathrm{tw}^{O(\mathrm{tw})})$-algorithm for Maximum Minimal Cut. To do this, we consider a slightly different problem, called Maximum Minimal s-t Cut: Given a graph $G = (V, E)$, an integer k and two vertices $s, t \in V$, determine whether there is a cut (S_1, S_2) of size at least k in G such that $s \in S_1$, $t \in S_2$ and (S_1, S_2) is minimal, that is, both $G[S_1]$ and $G[S_2]$ are connected. If we can solve Maximum Minimal s-t Cut in time $O^*(\mathrm{tw}^{O(\mathrm{tw})})$, we can also solve Maximum Minimal Cut in the same running time up to a polynomial factor in n since it suffices to compute Maximum Minimal s-t Cut for each pair of s and t.

Our algorithm is based on standard dynamic programming on a tree decomposition. This algorithm outputs a maximum minimal cut (S_1, S_2). Basically, the algorithm is almost the same as an $O^*(2^{\mathrm{tw}})$-algorithm for Max Cut in [7] except for keeping the connectivity of a cut. In other words, for each vertex, we label either **1** or **2**, which represent a vertex is assigned to S_1 or S_2. To keep track of the connectivity, for each bag X_i, we consider two partitions $\mathcal{S}_1$ and $\mathcal{S}_2$ of $S_1 \cap X_i$ and $S_2 \cap X_i$, respectively.

▶ **Theorem 7.** *Given a tree decomposition of width* tw *of* G, Maximum Minimal Cut *and* Maximum Minimal s-t Cut *can be solved in time* $O^*(\mathrm{tw}^{O(\mathrm{tw})})$.

The algorithm in Theorem 7 is applicable to Connected Maximum s-t Cut and Connected Maximum Cut as well.

▶ **Theorem 8.** *Give a tree decomposition of width* tw, Connected Maximum s-t Cut *and* Connected Maximum Cut *are solvable in time* $O^*(\mathrm{tw}^{O(\mathrm{tw})})$.

The dynamic programming algorithms in Theorems 7, 8 can be seen as ones for *connectivity problems* such as finding a Hamiltonian cycle, a feedback vertex set, and a Steiner tree. For such problems, we can improve the running time $\mathrm{tw}^{O(\mathrm{tw})}$ to $2^{O(\mathrm{tw})}$ using two techniques called the *rank-based approach* due to Bodlaender et al. [3] and the *cut & count technique* due to Cygan et al. [17]. In the next two subsections, we improve the running time of the algorithms described in this section using these techniques.

4.1.2 Rank-based approach

In this subsection, we provide faster $2^{O(\mathrm{tw})}$-time deterministic algorithms parameterized by tree-width. To show this, we use the rank-based approach proposed by Bodlaender et al. [3]. The key idea of the rank-based approach is to keep track of *small* representative sets of size $2^{O(\mathrm{tw})}$ that capture partial solutions of an optimal solution instead of $\mathrm{tw}^{O(\mathrm{tw})}$ partitions. Indeed, we can compute small representative sets within the claimed running time using reduce algorithm [3].

▶ **Theorem 9.** *Given a tree decomposition of width* tw, *there are* $O^*((1 + 2^{\omega+1})^{\mathrm{tw}})$*-time deterministic algorithms for* Connected Maximum s-t Cut *and* Connected Maximum Cut.

▶ **Theorem 10.** *Given a tree decomposition of width* tw*, there are $O^*(2^{(\omega+2)\mathrm{tw}})$-time deterministic algorithms for* Maximum Minimal *s-t* Cut *and* Maximum Minimal Cut*.*

4.1.3 Cut & Count

In this subsection, we design much faster randomized algorithms by using Cut & Count, which is the framework for solving the connectivity problems faster [17]. In Cut & Count, we count the number of *relaxed* solutions modulo 2 on a tree decomposition and determine whether there exists a connected solution by cancellation tricks.

▶ **Theorem 11.** *Given a tree decomposition of width* tw*, there is a Monte-Carlo algorithm that solves* Maximum Minimal Cut *and* Maximum Minimal *s-t* Cut *in time $O^*(4^{\mathrm{tw}})$. It cannot give false positives and may give false negatives with probability at most 1/2.*

▶ **Theorem 12.** *Given a tree decomposition of width* tw*, there is a Monte-Carlo algorithm that solves* Connected Maximum Cut *and* Connected Maximum *s-t* Cut *in time $O^*(3^{\mathrm{tw}})$. It cannot give false positives and may give false negatives with probability at most 1/2.*

4.2 Clique-width

In this section, we design XP algorithms for both Connected Maximum Cut and Maximum Minimal Cut when parameterized by clique-width. The algorithms are analogous to the dynamic programming algorithm for Maximum Cut given by Fomin et al. [22], but we need to carefully control the connectivity information in partial solutions.

Suppose that the clique-width of G is w. Then, G can be constructed by the four operations: *creation*, *disjoint union*, *joining*, and *relabeling* (see e.g., [14]). This construction naturally defines a tree expressing a sequence of operations. This tree is called a *w-expression tree* of G and used for describing dynamic programming algorithms for many problems based on clique-width. Here, we rather use a different graph parameter and its associated decomposition closely related to clique-width. We believe that this decomposition is more suitable to describe our dynamic programming.

▶ **Definition 13.** *Let $X \subseteq V(G)$. We say that $M \subseteq X$ is a* twin-set *of X if for any $v \in V(G) \setminus X$, either $M \subseteq N(v)$ or $M \cap N(v) = \emptyset$ holds. A twin-set M is called a* twin-class *of X if it is maximal subject to being a twin-set of X. X can be partitioned into twin-classes of X.*

▶ **Definition 14.** *Let w be an integer. We say that $X \subseteq V(G)$ is a w-*module *of G if X can be partitioned into w twin-classes $\{X_1, X_2, \ldots, X_w\}$. A* decomposition tree *of G is a pair of a rooted binary tree T and a bijection ϕ from the set of leaves of T to $V(G)$. For each node v of T, we denote by L_v the set of leaves, each of which is either v or a descendant of v. The* width *of a decomposition tree (T, ϕ) of G is the minimum w such that for every node v in T, the set $\bigcup_{l \in L_v} \phi(l)$ is a w_v-module of G with $w_v \leq w$. The* module-width *of G is the minimum t such that there is a decomposition tree of G of width w.*

Rao [41] proved that clique-width and module-width are linearly related to each other. Let $\mathrm{cw}(G)$ and $\mathrm{mw}(G)$ be the clique-width and the module-width of G, respectively. We note that a similar terminology "modular-width" has been used in many researches, but module-width used in this paper is different from it.

▶ **Theorem 15** ([41])**.** *For every graph G, $\mathrm{mw}(G) \leq \mathrm{cw}(G) \leq 2\mathrm{mw}(G)$.*

Moreover, given a w-expression tree of G, we can in time $O(n^2)$ compute a decomposition tree (T, ϕ) of G of width at most w and $w_v \leq w$ twin-classes of $\bigcup_{l \in L_v} \phi(l)$ for each node v in T [10].

Fix a decomposition tree (T, f) of G whose width is w. Our dynamic programming algorithm runs over the nodes of the decomposition tree in a bottom-up manner. For each node v in T, we let $\{X_1^v, X_2^v, \ldots, X_{w_v}^v\}$ be the twin-classes of $\bigcup_{l \in L_v} \phi(l)$. From now on, we abuse the notation to denote $\bigcup_{l \in L_v} \phi(l)$ simply by L_v. A tuple of $4w_v$ integers $t = (p_1, \overline{p}_1, p_2, \overline{p}_2, \ldots, p_{w_v}, \overline{p}_{w_v}, c_1, \overline{c}_1, c_2, \overline{c}_2, \ldots, c_{w_v}, \overline{c}_{w_v})$ is *valid* for v if it holds that $0 \leq p_i, \overline{p}_i \leq |X_i^v|$ with $p_i + \overline{p}_i = |X_i^v|$ and $c_i, \overline{c}_i \in \{0, 1\}$ for each $1 \leq i \leq w_v$. For a valid tuple t for v, we say that a cut $(S, L_v \setminus S)$ of $G[L_v]$ is *t-legitimate* if for each $1 \leq i \leq w_v$, it satisfies the following conditions:

- $p_i = |S \cap X_i^v|$,
- $\overline{p}_i = |(L_v \setminus S) \cap X_i^v|$,
- $G[S \cap X_i^v]$ is connected if $c_i = 1$, and
- $G[(L_v \setminus S) \cap X_i^v]$ is connected if $\overline{c}_i = 1$.

The size of a t-legitimate cut is defined accordingly. In this section, we allow each side of a cut to be empty and the empty graph is considered to be connected. Our algorithm computes the value $\mathrm{mc}(v, t)$ that is the maximum size of a t-legitimate cut for each valid tuple t and for each node v in the decomposition tree.

Leaves (Base step):

For each valid tuple t for a leaf v, $\mathrm{mc}(v, t) = 0$. Note that there is only one twin-class $X_1^v = \{v\}$ for v in this case.

Internal nodes (Induction step):

Let v be an internal node of T and let a and b be the children of v in T. Consider twin-classes $\mathcal{X}^v = \{X_1^v, X_2^v, \ldots, X_{w_v}^v\}$, $\mathcal{X}^a = \{X_1^a, X_2^a, \ldots, X_{w_a}^a\}$, and $\mathcal{X}^b = \{X_1^b, X_2^b, \ldots, X_{w_b}^b\}$ of L_v, L_a, and L_b, respectively. Note that $\mathcal{X}^a \cup \mathcal{X}^b$ is a partition of L_v.

▶ **Observation 1.** *$\mathcal{X}^v$ is a partition of L_v coarser than $\mathcal{X}^a \cup \mathcal{X}^b$.*

To see this, consider an arbitrary twin-class X_i^a of L_a. By the definition of twin-sets, for every $z \in V(G) \setminus L_a$, either $X_i^a \subseteq N(z)$ or $X_i^a \cap N(z) = \emptyset$ holds. Since $V(G) \setminus L_v \subseteq V(G) \setminus L_a$, X_i^a is also a twin-set of L_v, which implies X_i^a is included in some twin-class X_j^v of L_v. This argument indeed holds for twin-classes of L_b. Therefore, we have the above observation.

The intuition of our recurrence is as follows. By Observation 1, every twin-class of L_v can be obtained by merging some twin-classes of L_a and of L_b. This means that every t_v-legitimate cut of $G[L_v]$ for a valid tuple t_v for v can be obtained from some t_a-legitimate cut and t_b-legitimate cut for valid tuples for a and b, respectively. Moreover, for every pair of twin-classes X_i^a of L_a and X_j^b of L_b, either there are no edges between them or every vertex in X_i^a is adjacent to every vertex in X_j^b as X_i^a is a twin-set of L_v. Therefore, the number of edges in the cutset of a cut $(S, L_v \setminus S)$ between X_i^a and X_j^b depends only on the cardinality of $X_i^a \cap S$ and $X_j^b \cap S$ rather than actual cuts $(S \cap X_i^a, (L_a \setminus S) \cap X_i^a)$ and $(S \cap X_i^b, (L_b \setminus S) \cap X_i^b)$.

Now, we formally describe this idea. Let X^v be a twin-class of L_v. We denote by $I_a(X^v)$ (resp. $I_b(X^v)$) the set of indices i such that X_i^a (resp. X_i^b) is included in X^v and by $\mathcal{X}^a(X^v)$ (resp. $\mathcal{X}^b(X^v)$) the set $\{X_i^a : i \in I_a(X^v)\}$ (resp. $\{X_i^b : i \in I_b(X^v)\}$). For $X^a \in \mathcal{X}^a(X^v)$ and $X^b \in \mathcal{X}^a(X^v)$, we say that X^a is adjacent to X^b if every vertex in X^a is adjacent to every

vertex in X^b and otherwise X^a is not adjacent to X^b. This adjacency relation naturally defines a bipartite graph whose vertex set is $\mathcal{X}^a(X^v) \cup \mathcal{X}^b(X^v)$. We say that a subset of twin-classes of $\mathcal{X}^a(X^v) \cup \mathcal{X}^b(X^v)$ is *non-trivially connected* if it induces a connected bipartite graph with at least twin-classes. Let $S \subseteq X^v$. To make $G[S]$ (and $G[X^v \setminus S]$) connected, the following observation is useful.

▶ **Observation 2.** *Suppose $S \subseteq X^v$ has a non-empty intersection with at least two twin-classes of $\mathcal{X}^a(X^v) \cup \mathcal{X}^b(X^v)$. Then, $G[S]$ is connected if and only if the twin-classes having a non-empty intersection with S are non-trivially connected.*

This observation immediately follows from the fact that every vertex in a twin-class is adjacent to every vertex in an adjacent twin-class and is not adjacent to every vertex in a non-adjacent twin-class.

Let $t_v = (p_1^v, \overline{p}_1^v, \ldots, p_{w_v}^v, \overline{p}_{w_v}^v, c_1^v, \overline{c}_2^v, \ldots, c_{w_v}^v, \overline{c}_{w_v}^v)$ be a valid tuple for v. For notational convenience, we use $\mathbf{p}^v$ to denote $(p_1^v, \overline{p}_1^v, \ldots, p_{w_v}^v, \overline{p}_{w_v}^v)$ and $\mathbf{c}^v$ to denote $(c_1^v, \overline{c}_2^v, \ldots, c_{w_v}^v, \overline{c}_{w_v}^v)$ for each node v in T. For valid tuples $t_a = (\mathbf{p}^a, \mathbf{c}^a)$ for a and $t_b = (\mathbf{p}^b, \mathbf{c}^b)$ for b, we say that t_v *is consistent with the pair* (t_a, t_b) if for each $1 \leq i \leq w_v$,

C1 $p_i^v = \sum_{j \in I_a(X_i^v)} p_j^a + \sum_{j \in I_b(X_i^v)} p_j^b$;

C2 $\overline{p}_i^v = \sum_{j \in I_a(X_i^v)} \overline{p}_j^a + \sum_{j \in I_b(X_i^v)} \overline{p}_j^b$;

C3 if $c_i^v = 1$, either (1) $\{X_j^a : j \in I_a(X^v), p_j^a > 0\} \cup \{X_j^b : j \in I_b(X^v), p_j^b > 0\}$ is non-trivially connected or (2) exactly one of $\{p_j^s : s \in \{a,b\}, 1 \leq j \leq w_s\}$ is positive, say p_j^s, and $c_j^s = 1$;

C4 if $\overline{c}_i^v = 1$, either (1) $\{X_j^a : j \in I_a(X^v), \overline{p}_j^a > 0\} \cup \{X_j^b : j \in I_b(X^v), \overline{p}_j^b > 0\}$ is non-trivially connected or (2) exactly one of $\{\overline{p}_j^s : s \in \{a,b\}, 1 \leq j \leq w_s\}$ is positive, say $\overline{p}_j^s$, and $\overline{c}_j^s = 1$.

▶ **Lemma 16.**

$$\mathrm{mc}(v, t_v) = \max_{t_a, t_b} \left(\mathrm{mc}(a, t_a) + \mathrm{mc}(b, t_b) + \sum_{\substack{X_i^a \in \mathcal{X}^a, X_j^b \in \mathcal{X}^b \\ X_i^a, X_j^b : adjacent}} (p_i^a \overline{p}_j^b + p_j^b \overline{p}_i^a) \right),$$

where the maximum is taken over all consistent pairs (t_a, t_b).

Proof. We first show that the left-hand side is at most the right-hand side. Suppose $(S, L_v \setminus S)$ be a t_v-legitimate cut of $G[L_v]$ whose size is equal to $\mathrm{mc}(v, t_v)$. Let $S_a = S \cap L_a$ and $S_b = S \cap L_b$. We claim that $(S_a, L_a \setminus S_a)$ is a t_a-legitimate cut of $G[L_a]$ for some valid tuple t_a for a. This is obvious since we set $p_i^a = |S_a \cap X_i^a|$, $\overline{p}_i^a = |(L_a \setminus S_a) \cap X_i^a|$, $c_i^a = 1$ if $G[S_a \cap X_i^a]$ is connected, and $c_i^a = 1$ if $G[(L_a \setminus S_a) \cap X_i^a]$ is connected, which yields a valid tuple t_a for a. We also conclude that $(S_b, L_b \setminus S_b)$ is a t_b-legitimate cut of $G[L_b]$ for some valid tuple t_b for b. Moreover, the number of cut edges between twin-class X_i^a of L_a and twin-class X_j^b of L_b is $|S_a \cap X_i^a| \cdot |(L_b \setminus S_b) \cap X_j^b| + |S_b \cap X_j^b| \cdot |(L_b \setminus S_a) \cap X_i^a| = p_i^a \overline{p}_j^b + p_j^b \overline{p}_i^a$ if X_i^a and X_j^b is adjacent, zero otherwise. Therefore, the left-hand side is at most the right-hand side.

To show the converse direction, suppose $(S_a, L_a \setminus S_a)$ is a t_a-legitimate cut of $G[L_a]$ and $(S_b, L_b \setminus S_b)$ is a t_b-legitimate cut of $G[L_b]$, where t_v is consistent with (t_a, t_b) and the sizes of the cuts are $\mathrm{mc}(a, t_a)$ and $\mathrm{mc}(b, t_b)$, respectively. We claim that $(S_a \cup S_b, L_v \setminus (S_a \cup S_b))$ is a t_v-legitimate cut of $G[L_v]$. Since t_v is consistent with (t_a, t_b), for each $1 \leq i \leq w_v$, we have $p_i^v = \sum_{j \in I_a(X_i^v)} p_j^a + \sum_{j \in I_b(X_i^v)} p_j^b = \sum_{1 \leq j \leq w_a} |S_a \cap X_v^i| + \sum_{1 \leq j \leq w_b} |S_b \cap X_v^i| = |(S_a \cup S_b) \cap X_v^i|$. Symmetrically, we have $\overline{p}^i = |(L_v \setminus (S_a \cup S_b)) \cap X_i^v|$. If $c_i^v = 1$, by condition C3 of the

consistency, either (1) $\{X^a_j : j \in I_a(X^v), p^a_j > 0\} \cup \{X^b_j : j \in I_b(X^v), p^b_j > 0\}$ is non-trivially connected or (2) exactly one of $\{p^s_j : s \in \{a, b\}, 1 \leq j \leq w_s\}$ is positive, say p^s_j, and $c^s_j = 1$. If (1) holds, by Observation 2, $G[(S_a \cap S_b) \cap X^i_v]$ is connected. Otherwise, as $c^s_j = 1$, $G[S_s \cap X^i_v] = G[(S_a \cup S_b) \cap X^v_i]$ is also connected. By a symmetric argument, we conclude that $G[(L_v \setminus (S_a \cup S_b)) \cap X^i_v]$ is connected if $\overline{c}^v_i = 1$. Therefore the cut $(S_a \cup S_b, L_v \setminus (S_a \cup S_b))$ is t_v-legitimate. Since the cut edges between two twin-classes of L_a is counted by $\mathrm{mc}(a, t_a)$ and those between two twin-classes of L_v is counted by $\mathrm{mc}(b, t_b)$. Similar to the forward direction, the number of cut edges between a twin-class of L_a and a twin-class of L_b can be counted by the third term in the right-hand side of the equality. Hence, the left-hand side is at least right-hand side. ◀

▶ **Theorem 17.** Connected Maximum Cut *and* Maximum Minimal Cut *can be computed in time* $n^{O(w)}$ *provided that a* w*-expression tree of* G *is given as input.*

Proof. From a w-expression tree of G, we can obtain a decomposition tree (T, ϕ) of width at most w in $O(n^2)$ time using Rao's algorithm [41]. Based on this decomposition, we evaluate the recurrence in Lemma 16 in a bottom-up manner. The number of valid tuples for each node of T is at most $4^w n^w$. For each internal node v and for each valid tuple t_v for v, we can compute $\mathrm{mc}(v, t_v)$ in $(4^w n^w)^2 n^{O(1)}$ time. Overall, the running time of our algorithm is $n^{O(w)}$. Let r be the root of T. For Connected Maximum Cut, by the definition of legitimate cuts, we should take the maximum value among $\mathrm{mc}(r, (i, n-i, 1, j))$ for $1 \leq i < n$ and $j \in \{0, 1\}$. Note that as L_v has only one twin-class, the length of valid tuples is exactly four. For Maximum Minimal Cut, we should take the maximum value among $\mathrm{mc}(r, (i, n-i, 1, 1))$ for $1 \leq i < n$. ◀

Since there is an algorithm that, given a graph G and an integer k, either conclude that the clique-width of G is more than k or find a $(2^{k-1} - 1)$-expression tree of G in time $O(n^3)$ [33, 39, 38], Maximum Minimal Cut and Connected Maximum Cut are XP parameterized by the clique-width of the input graph.

4.3 Twin-cover

Maximum Cut is FPT when parameterized by twin-cover number [23]. In this section, we show that Connected Maximum Cut and Maximum Minimal Cut are also FPT when parameterized by twin-cover number.

▶ **Theorem 18.** Connected Maximum Cut *can be solved in time* $O^*(2^{2^{\mathrm{tc}}+\mathrm{tc}})$.

Proof. We first compute a minimum twin-cover X of $G = (V, E)$ in time $O^*(1.2738^{\mathrm{tc}})$ [23]. Now, we have a twin-cover X of size tc. Recall that $G[V \setminus X]$ consists of vertex disjoint cliques and for each $u, v \in Z$ in a clique Z of $G[V \setminus X]$, $N(u) \cap X = N(v) \cap X$.

We iterate over all possible subsets X' of X and compute the size of a maximum cut $(S, V \setminus S)$ of G with $S \cap X = X'$.

If $X' = \emptyset$, exactly one of the cliques of $G[V \setminus X]$ intersects S as $G[S]$ is connected. Thus, we can compute a maximum cut by finding a maximum cut for each clique of $G[V \setminus X]$, which can be done in polynomial time.

Suppose otherwise that $X' \neq \emptyset$. We define a *type* of each clique Z of $G[V \setminus X]$. The type of Z, denoted by $T(Z)$, is $N(Z) \cap X$. Note that there are at most $2^{\mathrm{tc}} - 1$ types of cliques in $G[V \setminus X]$.

For each type of cliques, we guess that S has an intersection with this type of cliques. There are at most $2^{2^{\mathrm{tc}}-1}$ possible combinations of types of cliques. Let $\mathcal{T}$ be the set of types

in $G[V \setminus X]$. For each guess $\mathcal{T}' \subseteq \mathcal{T}$, we try to find a maximum cut $(S, V \setminus S)$ such that $G[S]$ is connected, $S \cap X = X'$, for each $T \in \mathcal{T}'$, at least one of the cliques of type T has an intersection with S, and for each $T \notin \mathcal{T}'$, every clique of type T has no intersection with S. We can easily check if $G[S]$ will be connected as S contains a vertex of a clique of type $T \in \mathcal{T}'$. Consider a clique Z of type $T(Z) = X'' \subseteq X$. Since every vertex in Z has the same neighborhood in X, we can determine the number of cut edges incident to Z from the cardinality of $S \cap Z$. More specifically, if $|S \cap Z| = p$, the number of cut edges incident to Z is equal to $p(|Z| - p) + p|X'' \cap (X \setminus X')| + (|Z| - p)|X'' \cap X'|$. Moreover, we can independently maximize the number of cut edges incident to Z for each clique Z of $G[V \setminus X]$.

Overall, for each $X' \subseteq X$ and for each set of types $\mathcal{T}'$, we can compute a maximum connected cut with respect to X' and $\mathcal{T}'$ in polynomial time. Therefore, the total running time is bounded by $O^*(2^{2^{\mathrm{tc}}+\mathrm{tc}})$. ◀

▶ **Theorem 19.** Maximum Minimal Cut *can be solved in time* $O^*(2^{\mathrm{tc}}3^{2^{\mathrm{tc}}})$.

4.4 Solution size

In this section, we give FPT algorithms parameterized by the solution size for Connected Maximum Cut and Maximum Minimal Cut. To show this, we use the following theorem.

▶ **Theorem 20** ([2])**.** *The Cartesian product* $C_k \times K_2$ *of a* k*-circuit with* K_2 *is called a* k-prism*. If* G *contains no* k*-prism as a minor,* $\mathrm{tw}(G) = O(k^2)$.

Then we have the following theorem.

▶ **Theorem 21.** Connected Maximum Cut *and* Maximum Minimal Cut *can be solved in time* $O^*(2^{O(k^2)})$ *where* k *is the solution size.*

Proof. We first determine whether the tree-width of G is $O(k^2)$ in time $O^*(2^{O(k)})$ by using the algorithm in [5]. If $\mathrm{tw}(G) = O(k^2)$, the algorithm in [5] outputs a tree decomposition of width $O(k^2)$. Thus, we apply the dynamic programming algorithms based on tree decompositions described in Section 4.1, and the running time is $O^*(2^{O(k^2)})$. Otherwise, we can conclude that G has a minimal cut (and also a connected cut) of size at least k. To see this, consider a k-prism minor of G. Then, we take k "middle edges" corresponding to K_2 in the k-prism minor and add some edges to make these edges form a cutset of some minimal cut of G. The size of such a cut is at least k and hence G has a minimal cut and a connected cut of size at least k. ◀

For Connected Maximum Cut, we can further improve the running time by giving an $O^*(9^k)$-time algorithm.

In [21], Fellows et al. proposed a "Win/Win" algorithm that outputs in linear time either a spanning tree of G having at least k leaves, or a path decomposition of G of width at most $2k$. If G has such a spanning tree, we can construct a cut $(S, V \setminus S)$ of size at least k by taking the internal vertices of the tree for S. Clearly, $G[S]$ is connected, and hence we are done in this case. Otherwise, we have a path decomposition of width at most $2k$. Thus, we can compute Connected Maximum Cut on such a path decomposition by using an $O^*(3^{\mathrm{tw}})$-algorithm in Section 4.1.

▶ **Theorem 22.** *There is a Monte-Carlo algorithm that solves* Connected Maximum Cut *in time* $O^*(9^k)$*. It cannot give false positives and may give false negatives with probability at most 1/2.*

Also, using the rank-based algorithm in Theorem 9, we obtain an $O^*(38.2^k)$-time deterministic algorithm for Connected Maximum Cut. Note that our rank-based algorithm in Theorem 9 runs in time $O^*((1+2^\omega)^{\mathrm{pw}})$ on a path decomposition and $(1+2^\omega)^2 < 38.2$, where $\omega < 2.3727$ is the exponent of matrix multiplication.

▶ **Theorem 23.** *There is an $O^*(38.2^k)$-time deterministic algorithm for* Connected Maximum Cut.

As for kernelization, it is not hard to see that Connected Maximum Cut and Maximum Minimal Cut do not admit a polynomial kernelization unless NP $\subseteq$ coNP/poly since both problems are trivially OR-compositional [4]; at least one of graphs $G_1, G_2, \ldots G_t$ have a connected/minimal cut of size at least k if and only if their disjoint union $G_1 \cup G_2 \cup \cdots \cup G_t$ has.

▶ **Theorem 24.** *Unless* NP $\subseteq$ coNP/poly*,* Maximum Minimal Cut *and* Connected Maximum Cut *admit no polynomial kernel parameterized by the solution size.*

5 Conclusion and Remark

In this paper, we studied two variants of Max Cut, called Connected Maximum Cut and Maximum Minimal Cut. We showed that both problems are NP-complete even on planar bipartite graphs and split graphs. For the parameterized complexity, we gave FPT algorithms parameterized by tree-width, twin-cover number, and the solution size, respectively. Moreover, we designed XP-algorithms parameterized by clique-width.

Finally, we mention our problems on weighted graphs. It is not hard to see that Connected Maximum Cut and Maximum Minimal Cut remain to be FPT with respect to tree-width. However, our results with respect to clique-width and twin-cover number would not be extended to weighted graphs since both problems are NP-hard on 0-1 edge-weighted complete graphs.

References

1. C. Bazgan, L. Brankovic, K. Casel, H. Fernau, K. Jansen, K.-M. Klein, M. Lampis, M. Liedloff, J. Monnot, and V. T. Paschos. The many facets of upper domination. *Theoretical Computer Science*, 717:2–25, 2018.
2. E. Birmelé, J. A. Bondy, and B. A. Reed. *Brambles, Prisms and Grids*, pages 37–44. Birkhäuser Basel, Basel, 2007.
3. H. L. Bodlaender, M. Cygan, S. Kratsch, and J. Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.
4. H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.
5. H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk. A $c^k n$ 5-Approximation Algorithm for Treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016.
6. H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating Treewidth, Pathwidth, Frontsize, and Shortest Elimination Tree. *Journal of Algorithms*, 18(2):238–255, 1995.
7. H. L. Bodlaender and K. Jansen. On the Complexity of the Maximum Cut Problem. *Nordic Journal of Computing*, 7(1):14–31, 2000.
8. N. Boria, F. D. Croce, and V. T. Paschos. On the max min vertex cover problem. *Discrete Applied Mathematics*, 196:62–71, 2015.

9 A. Boyacı, T. Ekim, and M. Shalom. A polynomial-time algorithm for the maximum cardinality cut problem in proper interval graphs. *Information Processing Letters*, 121:29–33, 2017.

10 B.-M. Bui-Xuan, O. Suchý, J. A. Telle, and M. Vatshelle. Feedback vertex set on graphs of low clique-width. *European Journal of Combinatorics*, 34(3):666–679, 2013.

11 R. Carvajal, M. Constantino, M. Goycoolea, J. P. Vielma, and A. Weintraub. Imposing Connectivity Constraints in Forest Planning Models. *Operations Research*, 61(4):824–836, 2013.

12 B. Chaourar. A Linear Time Algorithm for a Variant of the MAX CUT Problem in Series Parallel Graphs. *Advances in Operations Research*, pages 1267108:1–1267108:4, 2017.

13 B. Chaourar. Connected max cut is polynomial for graphs without $K_5 \setminus e$ as a minor. *CoRR*, abs/1903.12641, 2019.

14 B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77–114, 2000.

15 M. Cygan. Deterministic Parameterized Connected Vertex Cover. In *SWAT 2012*, pages 95–106, 2012.

16 M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer International Publishing, 2015.

17 M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving Connectivity Problems Parameterized by Treewidth in Single Exponential Time. In *FOCS 2011*, pages 150–159, 2011.

18 M. Demange. A Note on the Approximation of a Minimum-Weight Maximal Independent Set. *Computational Optimization and Applications*, 14(1):157–169, 1999.

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

20 J. Díaz and M. Kamiński. MAX-CUT and MAX-BISECTION are NP-hard on unit disk graphs. *Theoretical Computer Science*, 377(1):271–276, 2007.

21 M. R. Fellows, D. Lokshtanov, N. Misra, M. Mnich, F. Rosamond, and S. Saurabh. The Complexity Ecology of Parameters: An Illustration Using Bounded Max Leaf Number. *Theory of Computing Systems*, 45(4):822–848, 2009.

22 F. V. Fomin, P. Golovach, D. Lokshtanov, and S. Saurabh. Almost Optimal Lower Bounds for Problems Parameterized by Clique-Width. *SIAM Journal on Computing*, 43(5):1541–1563, 2014.

23 R. Ganian. Improving Vertex Cover as a Graph Parameter. *Discrete Mathematics and Theoretical Computer Science*, 17(2):77–100, 2015.

24 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

25 M. X. Goemans and D. P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

26 V. Grimm, T. Kleinert, F. Liers, M. Schmidt, and G. Zöttl. Optimal price zones of electricity markets: a mixed-integer multilevel model and global solution approaches. *Optimization Methods and Software*, 34(2):406–436, 2019.

27 S. Guha and S. Khuller. Approximation Algorithms for Connected Dominating Sets. *Algorithmica*, 20(4):374–387, 1998.

28 V. Guruswami. Maximum cut on line and total graphs. *Discrete Applied Mathematics*, 92(2):217–221, 1999.

29 F. Hadlock. Finding a Maximum Cut of a Planar Graph in Polynomial Time. *SIAM Journal on Computing*, 4(3):221–225, 1975.

30 D. J. Haglin and S. M. Venkatesan. Approximation and intractability results for the maximum cut problem and its variants. *IEEE Transactions on Computers*, 40(1):110–113, 1991.

31 M. T. Hajiaghayi, G. Kortsarz, R. MacDavid, M. Purohit, and K. Sarpatwar. Approximation Algorithms for Connected Maximum Cut and Related Problems. In *ESA 2015*, pages 693–704, 2015.

32 T. Hanaka, H. L. Bodlaender, T. C. van der Zanden, and H. Ono. On the Maximum Weight Minimal Separator. In *TAMC 2017*, pages 304–318, 2017.

33 P. Hliněný and S. Oum. Finding Branch-Decompositions and Rank-Decompositions. *SIAM Journal on Computing*, 38(3):1012–1032, 2008.

34 R. M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

35 K. Khoshkhah, M. K. Ghadikolaei, J. Monnot, and F. Sikora. Weighted Upper Edge Cover: Complexity and Approximability. In *WALCOM 2019*, pages 235–247, 2019.

36 M. Mahajan and V. Raman. Parameterizing above Guaranteed Values: MaxSat and MaxCut. *Journal of Algorithms*, 31(2):335–354, 1999.

37 G. I. Orlova and Y. G. Dorfman. Finding the maximal cut in a graph. *Engineering Cyvernetics*, 10(3):502–506, 1972.

38 S. Oum. Approximating Rank-width and Clique-width Quickly. *ACM Transactions on Algorithms*, 5(1):10:1–10:20, 2008.

39 S. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006.

40 V. Raman and S. Saurabh. Improved fixed parameter tractable algorithms for two "edge" problems: MAXCUT and MAXDAG. *Information Processing Letters*, 104(2):65–72, 2007.

41 M. Rao. Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics*, 308(24):6157–6165, 2008.

42 N. Robertson and P. D. Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986.

43 S. Saurabh and M. Zehavi. Parameterized Complexity of Multi-Node Hubs. In *IPEC 2018*, volume 115, pages 8:1–8:14, 2019.

44 S. Vicente, V. Kolmogorov, and C. Rother. Graph cut based image segmentation with connectivity priors. In *CVPR 2008*, pages 1–8, 2008.

45 M. Yannakakis and F. Gavril. Edge Dominating Sets in Graphs. *SIAM Journal on Applied Mathematics*, 38(3):364–372, 1980.

46 M. Zehavi. Maximum Minimal Vertex Cover Parameterized by Vertex Cover. *SIAM Journal on Discrete Mathematics*, 31(4):2440–2456, 2017.

Multistage Vertex Cover

Till Fluschnik
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany
till.fluschnik@tu-berlin.de

Rolf Niedermeier
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany
rolf.niedermeier@tu-berlin.de

Valentin Rohm
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany
valentinl.rohm@campus.tu-berlin.de

Philipp Zschoche
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany
zschoche@tu-berlin.de

Abstract

Covering all edges of a graph by a small number of vertices, this is the NP-hard Vertex Cover problem, is among the most fundamental algorithmic tasks. Following a recent trend in studying dynamic and temporal graphs, we initiate the study of Multistage Vertex Cover. Herein, having a series of graphs with same vertex set but over time changing edge sets (known as temporal graph consisting of time layers), the goal is to find for each layer of the temporal graph a small vertex cover *and* to guarantee that the two vertex cover sets between two subsequent layers differ not too much (specified by a given parameter). We show that, different from classic Vertex Cover and some other dynamic or temporal variants of it, Multistage Vertex Cover is computationally hard even in fairly restricted settings. On the positive side, however, we also spot several fixed-parameter tractability results based on some of the most natural parameterizations.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases NP-hardness, dynamic graph problems, temporal graphs, time-evolving networks, W[1]-hardness, fixed-parameter tractability, kernelization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.14

Related Version A full version of the paper: `https://arxiv.org/abs/1906.00659`.

Funding *Till Fluschnik*: Supported by the DFG, project TORE (NI 369/18).

1 Introduction

Vertex Cover (VC) asks, given an undirected graph G and an integer $k \geq 0$, whether at most k vertices can be deleted from G such that the remaining graph contains no edge. VC is NP-hard and it is a formative problem of algorithmics and combinatorial optimization. We study a *time-dependent*, "*multistage*" version, namely a variant of VC on temporal graphs. A *temporal graph* $\mathcal{G}$ is a tuple $(V, \mathcal{E}, \tau)$ consisting of a set V of vertices, a discrete time-horizon τ, and a set of temporal edges $\mathcal{E} \subseteq \binom{V}{2} \times \{1, \dots, \tau\}$. Equivalently, a temporal graph $\mathcal{G}$ can be seen as a vector $(G_1, \dots, G_\tau)$ of static graphs (*layers*), where each graph is defined over the same vertex set V. Then, our specific goal is to find a small vertex cover S_i for each layer G_i such that the sizes of the symmetric differences $S_i \triangle S_{i+1}$ between the vertex covers S_i and S_{i+1} of every two consecutive layers G_i and G_{i+1} are small. Formally, we thus introduce and study the following problem.

Multistage Vertex Cover (MSVC)

Input: A temporal graph $\mathcal{G} = (V, \mathcal{E}, \tau)$ and two integers $k \in \mathbb{N}, \ell \in \mathbb{N}_0$.

Question: Is there a sequence $\mathcal{S} = (S_1, \ldots, S_\tau)$ such that

- **(i)** for all $i \in \{1, \ldots, \tau\}$, it holds that $S_i \subseteq V$ is a size-at-most-k vertex cover for G_i, and
- **(ii)** for all $i \in \{1, \ldots, \tau - 1\}$, it holds that $|S_i \triangle S_{i+1}| \leq \ell$?

Throughout this paper we assume that $0 < k < |V|$ because otherwise we have a trivial instance. In our model, we follow the recently proposed *multistage* [4, 16, 5, 11] view on classical optimization problems on temporal graphs.

In general, the motivation behind a multistage variant of a classical problem such as Vertex cover is that the environment changes over time (here reflected by the changing edge sets in the temporal graph) and a corresponding adaptation of the current solution comes with a cost. In this spirit, the parameter ℓ in the definition of MSVC allows to model that only moderate changes concerning the solution vertex set may be wanted when moving from one layer to the subsequent one. Indeed, in this sense ℓ can be interpreted as a parameter measuring the degree of (non-)conservation [17, 1].

It is immediate that MSVC is NP-hard as it generalizes Vertex Cover ($\tau = 1$). We will study its parameterized complexity regarding the problem-specific parameters k, τ, ℓ, and some of their combinations, as well as restrictions to temporal graph classes [13].

Related Work. The literature on vertex covering is extremely rich, even when focusing on parameterized complexity studies. Indeed, Vertex Cover (VC) can be seen as "drosophila" of parameterized algorithmics. Thus, we only consider VC studies closely related to our setting. First, we mention in passing that VC is studied in dynamic graphs [19, 3] and graph stream models [6]. More importantly for us, Akrida et al. [2] studied a variant of VC on temporal graphs. Their model significantly differs from ours: They want an edge to be covered at least once over every time window of some given size Δ. That is, they define a temporal vertex cover as a set $S \subseteq V \times \{1, \ldots, \tau\}$ such that, for every time window of size Δ and for each edge $e = \{v, w\}$ appearing in a layer contained in the time window, it holds that $(v, t) \in S$ or $(w, t) \in S$ for some t in the time window with $(e, t) \in \mathcal{E}$. For their model, they ask whether such an S of small cardinality exists. Note that if $\Delta > 1$, then for some $t \in \{1, \ldots, \tau\}$ the set $S_t := \{v \mid (v, t) \in S\}$ is not necessarily a vertex cover of layer G_t. For $\Delta = 1$, each S_t must be a vertex cover of G_t. However, in Akrida et al.'s model the size of each S_t as well as the size of the symmetric difference between each S_t and S_{t+1} may strongly vary. They provide several hardness results and algorithms (mostly referring to approximation or exact algorithms, but not to parameterized complexity studies).

A second related line of research, not directly referring to temporal graphs though, studies reconfiguration problems which arise when we wish to find a step-by-step transformation between two feasible solutions of a problem such that all intermediate results are feasible solutions as well [18, 15]. Mouawad et al. [22, 21] studied, among other reconfiguration problems, Vertex Cover Reconfiguration which takes as input a graph G, two vertex covers S and T of size at most k each, and an integer τ. The goal is to determine whether there is a sequence $(S = S_1, \ldots, S_\tau = T)$ such that each S_t is a vertex cover of size at most k. The essential difference to our model is that from one "sequence element" to the next only one vertex may be changed and that the input graph does not change over time. Indeed, there is an easy reduction of this model to ours while the opposite direction is unlikely to hold. This is substantiated by the fact that Mouawad et al. [22] showed that Vertex Cover Reconfiguration is fixed-parameter tractable when parameterized by vertex cover size k while we show W[1]-hardness for the corresponding case of MSVC.

Table 1 Overview on our results. The column headings describe the restrictions on the input and each row corresponds to a parameter. p-NP-hard, PK, and NoPK abbreviate para-NP-hard, polynomial problem kernel, and no problem kernel of polynomial size unless coNP ⊆ NP/ poly. † (Obs. 2.5)

	general layers		tree layers	one-edge layers
	$0 \le \ell < 2k$	$\ell \ge 2k$	$0 \le \ell < 2k$	$0 \le \ell < 2$
	NP-hard		NP-hard (Thm. 3.1(i))	NP-hard (Thm. 3.1(ii))
τ	p-NP-hard (Thm. 3.1)		p-NP-hard (Thm. 3.1)	FPT, PK (Obs. 5.8)
k	XP, W[1]-h., (Thm. 4.1)	FPT†, NoPK (Thm. 5.1)	XP, W[1]-h. (Thm. 4.1)	*open*, NoPK (Thm. 5.1)
$k+\tau$	FPT, PK (Thm. 5.5)		FPT, PK (Thm. 5.5)	FPT, PK (Thm. 5.5)

Finally, there is also a close relation to the research on dynamic parameterized problems [1, 20]. Krithika et al. [20] studied Dynamic Vertex Cover where one is given two graphs on the same vertex set and a vertex cover for one of them together with the guarantee that the cardinality of the symmetric difference between the two edge sets is upper-bounded by a parameter d. The task then is to find a vertex cover for the second graph that is "close enough" (measured by a second parameter) to the vertex cover of the first graph. They show fixed-parameter tractability and a linear kernel with respect to d.

Our Contributions. Our results, focusing on the three perhaps most natural parameters, are summarized in Table 1.[1] We highlight a few specific results. Multistage Vertex Cover remains NP-hard even if every layer consists of only one edge; clearly, the corresponding hardness reduction then exploits an unbounded number τ of time layers. If one only has two layers, however, one of them being a tree and the other being a path, then again Multistage Vertex Cover already becomes NP-hard. MSVC parameterized by solution size k is fixed-parameter tractable if $\ell \ge 2k$, but becomes W[1]-hard if $\ell < 2k$. Considering the tractability results for Dynamic Vertex Cover [20] and Vertex Cover Reconfiguration [22], this hardness is surprising and is our most technical result. Furthermore, in the former case (parameterization by k with $\ell \ge 2k$) MSVC does not admit a problem kernel of polynomial size unless coNP ⊆ NP/ poly. If one considers the combined parameter $k + \tau$, however, then besides fixed-parameter tractability in *all* cases we also obtain polynomial-sized kernels.

2 Preliminaries

We denote by $\mathbb{N}$ and $\mathbb{N}_0$ the natural numbers excluding and including zero, respectively. For two sets A and B, we denote by $A \triangle B := (A \setminus B) \cup (B \setminus A)$ the symmetric difference of A and B, and by $A \uplus B$ the disjoint union of A and B. We use basic notation from graph theory [8] and parameterized algorithmics [7].

Temporal Graphs. A temporal graph $\mathcal{G}$ is a tuple $(V, \mathcal{E}, \tau)$ consisting of the set of vertices V, the set of temporal edges $\mathcal{E}$, and a discrete time-horizon τ. A temporal edge e is an element in $\binom{V}{2} \times \{1, \dots, \tau\}$. Equivalently, a temporal graph $\mathcal{G}$ is a vector of static graphs $(G_1, \dots, G_\tau)$,

[1] Several details and proofs (marked with ⋆) are deferred to the full version of the paper: `https://arxiv.org/abs/1906.00659`.

where each graph is defined over the same vertex set V. We also denote by $V(\mathcal{G})$, $\mathcal{E}(\mathcal{G})$, and $\tau(\mathcal{G})$ the set of vertices, the set of temporal edges, and the discrete time-horizon of $\mathcal{G}$, respectively. The *underlying graph* $G_\downarrow = G_\downarrow(\mathcal{G})$ of a temporal graph $\mathcal{G}$ is the static graph with vertex set $V(\mathcal{G})$ and edge set $\{e \mid \exists t \in \{1, \ldots, \tau(\mathcal{G})\} : (e, t) \in E\}$.

General Observations on MSVC. We state some simple but useful observations on MSVC and its relation to Vertex Cover.

▶ **Observation 2.1** (⋆). *Every instance* $(\mathcal{G}, k, \ell)$ *of* MSVC *with* $k \geq \sum_{i=1}^{\tau(\mathcal{G})} |E(G_i)|$ *is a* yes*-instance.*

▶ **Observation 2.2** (⋆). *Let* $(\mathcal{G}, k, \ell)$ *be an instance of* MSVC. *If* $(\mathcal{G}, k, \ell)$ *is a* yes*-instance, then there is a solution* $\mathcal{S} = (S_1, \ldots, S_\tau)$ *such that* $|S_1| = k$ *and* $k - 1 \leq |S_i| \leq k$ *for all* $i \in \{1, \ldots, \tau\}$.

▶ **Observation 2.3** (⋆). *There is an algorithm that maps any instance* (G, k) *of* Vertex Cover *in* $\tau \cdot |V(G)|^{O(1)}$ *time to an equivalent instance* $(\mathcal{G}, k, \ell)$ *of* MSVC *with* $\ell = 0$, *where* $\mathcal{G}$ *is a sequence of any* τ *subgraphs of* G *such that the underlying graph is* G.

▶ **Observation 2.4** (⋆). *There is a polynomial-time algorithm that maps any instance* $(\mathcal{G}, k, \ell)$ *of* MSVC *with* $\ell = 0$ *to an equivalent instance* $(G_\downarrow(\mathcal{G}), k)$ *of* Vertex Cover.

▶ **Observation 2.5** (⋆). *An instance* $(\mathcal{G}, k, \ell)$ *of* MSVC *with* $\ell \geq 2k$ *and* $\mathcal{G} = (G_1, \ldots, G_\tau)$ *can be decided by deciding each instance of the set* $\{(G_i, k) \mid 1 \leq i \leq \tau\}$ *of* Vertex Cover*-instances.*

3 Hardness On Restricted Inputs

MSVC is NP-hard as it generalizes Vertex Cover ($\tau = 1$). In this section we prove that MSVC remains NP-hard on very restricted inputs.

▶ **Theorem 3.1.** Multistage Vertex Cover *is* NP*-hard even if*

(i) $\tau = 2$, $\ell = 0$, *and the first layer is a path and the second layer is a tree, or*

(ii) *every layer contains only one edge and* $\ell = 1$.

▶ Remark 3.2. Theorem 3.1(i) is tight regarding τ since Vertex Cover (i.e., MSVC with $\tau = 1$) on trees is solvable in polynomial time. Theorem 3.1(ii) is tight regarding ℓ, because in the case of $\ell \neq 1$ either Observation 2.3 or Observation 2.5 is applicable.

Vertex Cover remains NP-complete on cubic Hamiltonian graphs when a Hamiltonian cycle is additionally given in the input [12]—we refer to this problem as Hamiltonian Cubic Vertex Cover (HCVC). To prove Theorem 3.1(i), we give a polynomial-time many-one reduction from HCVC to MSVC with two layers, one being a path, the other being a tree.

▶ **Proposition 3.3** (⋆). *There is a polynomial-time algorithm that maps any instance* $(G = (V, E), k, C)$ *of* HCVC *to an equivalent instance* $(\mathcal{G}, k', \ell')$ *of* MSVC *with* $\tau = 2$ *and the first layer* G_1 *being a path and second layer* G_2 *being a tree.*

In order to prove Theorem 3.1(ii), we give a polynomial-time many-one reduction from Vertex Cover to MSVC.

▶ **Proposition 3.4** (⋆). *There is a polynomial-time algorithm that maps any instance* $(G = (V, E), k)$ *of* Vertex Cover *to an equivalent instance* $(\mathcal{G}, k', \ell')$ *of* MSVC *where* $\ell' = 1$ *and every layer* G_i *contains only one edge.*

4 Parameter Vertex Cover Size

In this section, we study the parameter size k of the vertex cover of each layer for MSVC. Vertex Cover and Vertex Cover Reconfiguration [22] when parameterized by the vertex cover size are fixed-parameter tractable. We prove that this is no longer true for MSVC (unless FPT = W[1]).

▶ **Theorem 4.1.** Multistage Vertex Cover *parameterized by k is in* XP *and* W[1]*-hard.*

We first show the XP-algorithm (Section 4.1) and then prove W[1]-hardness (Section 4.2).

4.1 An XP-Algorithm

In this section, we prove the following.

▶ **Proposition 4.2.** *Every instance $(\mathcal{G}, k, \ell)$ of* Multistage Vertex Cover *can be decided in $O(\tau(\mathcal{G}) \cdot |V(\mathcal{G})|^{2k+1})$ time.*

In a nutshell, we first consider for each layer all subsets of vertices of size at most k that form a vertex cover. Second, we find a sequence of vertex covers for all layers such that the sizes of the symmetric differences for every two consecutive solutions is at most ℓ. We show that the second step can be solved via computing a directed source-sink path in a helper graph that we call *configuration graph*.

▶ **Definition 4.3.** *Given a temporal graph $\mathcal{G}$, the* (k,ℓ)-configuration graph *of $\mathcal{G}$ is the graph $D = (V = V_1 \uplus \cdots \uplus V_\tau \uplus \{s,t\}, A, \gamma)$ equipped with a function $\gamma : V \to \{V' \subseteq V(\mathcal{G}) \mid |V'| \leq k\}$ such that*

- **(i)** *for every $i \in \{1, \ldots, \tau(\mathcal{G})\}$, it holds true that S is a vertex cover of G_i of size at most k if and only if there is a vertex $v \in V_i$ with $\gamma(v) = S$,*
- **(ii)** *there is an arc from v to w, $v, w \in V$, if and only if $v \in V_i$, $w \in V_{i+1}$, and $|\gamma(v) \triangle \gamma(w)| \leq \ell$, and*
- **(iii)** *there is an arc (s, v) for all $v \in V_1$ and an arc (v, t) for all $v \in V_\tau$.*

Note that Mouawad et al. [22] used a similar configuration graph to show fixed-parameter tractability of Vertex Cover Reconfiguration parameterized by the vertex cover size k. In the multistage setting the configuration graph is too large for fixed-parameter tractability regarding k. However, we show an XP-algorithm regarding k to construct the configuration graph.

▶ **Lemma 4.4** (⋆)**.** *The (k,ℓ)-configuration graph of a temporal graph $\mathcal{G}$ with n vertices and time horizon τ*

- **(i)** *can be constructed in $O(\tau \cdot n^{2k+1})$ time, and*
- **(ii)** *contains at most $\tau \cdot 2n^k + 2$ vertices and $(\tau - 1)n^{2k} + 2n^k$ arcs.*

▶ **Lemma 4.5** (⋆)**.** MSVC*-instance $(\mathcal{G}, k, \ell)$ is a* `yes`*-instance if and only if there is an s-t path in the (k,ℓ)-configuration graph D of $\mathcal{G}$.*

We are ready to prove Proposition 4.2.

Proof of Proposition 4.2. First, compute the configuration graph D of the instance $(\mathcal{G} = (V, \mathcal{E}, \tau), k, \ell)$ of Multistage Vertex Cover in $O(\tau \cdot |V|^{2k+1})$ time (Lemma 4.4(i)). Then, find an s-t path in D with a breadth-first search in $O(\tau \cdot |V|^{2k})$ time (Lemma 4.4(ii)). If an s-t path is found, then return `yes`, otherwise return `no` (Lemma 4.5). ◀

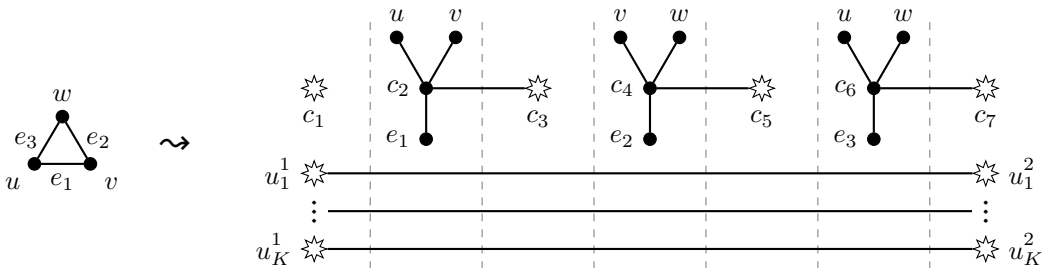


Figure 1 Illustration of Construction 1 on an example graph (left-hand side) and the first seven layers of the obtained graph (right-hand side). Star-shapes illustrate star graphs with $k' + 1$ leaves. Dashed vertical lines separate layers.

▶ Remark 4.6. The reason why the algorithm behind Proposition 4.2 is only an XP-algorithm and not an FPT-algorithm regarding k for Multistage Vertex Cover is because we do not have a better upper bound on the number of vertices in the (k, ℓ)-configuration graph for $\mathcal{G}$ than $O(\tau(\mathcal{G}) \cdot |V(\mathcal{G})|^k)$. This is due to the fact that we check for each subset of $V(\mathcal{G})$ of size k or $k-1$ whether it is a vertex cover in some layer.

This changes if we consider Minimal Multistage Vertex Cover where we additionally demand the i-th set in the solution to be a *minimal* vertex cover for the layer G_i. Here, we can enumerate for each layer G_i all minimal vertex covers of size at most k (and hence all candidates for the i-th set of the solution) with the folklore search-tree algorithm for vertex cover. This leads to $O(2^k \tau(\mathcal{G}))$ many vertices in the (k, ℓ)-configuration graph (for Minimal Multistage Vertex Cover) and thus to fixed-parameter tractability of Minimal Multistage Vertex Cover parameterized by the vertex cover size k.

However, as we show next it is not likely (unless FPT=W[1]) that one can substantially improve the algorithm behind Proposition 4.2.

4.2 Parameterized Intractability

In this section we show that MSVC is W[1]-hard when parameterized by k. This hardness result is established by the following parameterized reduction from the W[1]-complete [9] Clique problem, where, given an undirected graph G and a positive integer k, the question is whether G contains a clique of size k (that is, k vertices that are pairwise adjacent).

▶ **Proposition 4.7.** *There is an algorithm that maps any instance (G, k) of* Clique *in polynomial time to an equivalent instance $(\mathcal{G}, k', \ell)$ of* MSVC *with $k' = 2\binom{k}{2} + k + 1$, $\ell = 2$, and each layer of $\mathcal{G}$ being a tree.*

The proof of Proposition 4.7 is deferred to the end of this section. It is a reduction from Clique where we construct an instance of MSVC from an instance of Clique as follows (see Figure 1 for an illustrative example).

▶ **Construction 1.** Let $(G = (V, E), k)$ be an instance of Clique with $m = |E|$ and $E = \{e_1, \ldots, e_m\}$. Let

$$K = \binom{k}{2}, \qquad k' = 2K + k + 1, \qquad \text{and} \qquad \kappa = K + k + 3.$$

We construct a temporal graph $\mathcal{G} = (V', \mathcal{E}, \tau)$ as follows. Let V' be initially $V \cup E$ (note that E simultaneously describes the edge set of G and a vertex subset of $\mathcal{G}$). We add the

following vertex sets

$$U^t = \{u_j^t \mid j \in \{1, \ldots, K\}\},\ t \in \{1, \ldots, \kappa + 1\} \text{ and } C = \{c_1, \ldots, c_{2m\kappa+1}\}.$$

Let $\mathcal{E}$ be initially empty. We extend the set V' and define $\mathcal{E}$ through the $\tau = 2m\kappa + 1$ layers we construct in the following.

(1) In each layer G_i with i being odd, make c_i the center of a star with $k' + 1$ leaves.
(2) In each layer G_{2mj+1}, $j \in \{0, \ldots, \kappa\}$, make each vertex in U^{j+1} the center of a star with $k' + 1$ leaves.
(3) For each $j \in \{0, \ldots, \kappa - 1\}$, in each layer G_{2mj+i} with $i \in \{1, \ldots, 2m + 1\}$, make u_x^{j+1} adjacent to u_x^{j+2} for each $x \in \{1, \ldots, K\}$.
(4) For each i being even, add the edge $\{c_i, c_{i+1}\}$ to G_i and to G_{i+1}.
(5) For each $j \in \{0, \ldots, \kappa - 1\}$, for each $i \in \{1, \ldots, m\}$, in G_{2mj+2i}, make c_{j2m+2i} adjacent with $e_i = \{v, w\}$, v, and w.

This finishes the construction of $\mathcal{G}$. ⌟

The construction essentially repeats the same gadget (which we call *phase*) κ times where the layer $2m \cdot i + 1$ is simultaneously last layer of phase i and the first layer of phase $i + 1$. In the beginning of phase i, a solution must contain the vertices of U^i. The idea now is that during phase i one has to exchange the vertices of U^i with the vertices of U^{i+1}.

It is not difficult to see that the instance in Construction 1 can be computed in polynomial time. Hence, it remains to prove the equivalence stated in Proposition 4.7. We prepare the proofs of the forward and the backward direction in Sections 4.2.1 and 4.2.2, respectively.

▶ Remark 4.8. We can turn the instance $(\mathcal{G}, k', \ell)$ computed by Construction 1 into an equivalent instance $(\mathcal{G}', k'', \ell)$ where each layer is a tree as follows. Set $k'' = k' + 1$. Add a vertex x to $\mathcal{G}$. In each layer of $\mathcal{G}$, make x a star with $k'' + 1$ vertices and connect x with exactly one vertex of each connected component. Note that in every solution x is contained in a vertex cover for each layer in $\mathcal{G}'$.

4.2.1 Forward direction

The forward direction of Proposition 4.7 is—in a nutshell—as follows: If $V' \cup E'$ with $V' \subseteq V$ and $E' \subseteq E$ correspond to the vertex set and edge set of a clique of size k, then there are K layers in each phase covered by $V' \cup E'$. Hence, having K layers where no vertices from C have to be exchanged, in each phase t we can exchange all vertices from U^t to U^{t+1}. Starting with set $S_1 = U^1 \cup V' \cup E' \cup \{c_1\}$ then yields a solution.

▶ **Lemma 4.9** (⋆). *Let (G, k) be an instance of* Clique *and $(\mathcal{G}, k', \ell)$ be the instance of* Multistage Vertex Cover *resulting from Construction 1. If (G, k) is a* yes*-instance, then $(\mathcal{G}, k', \ell)$ is a* yes*-instance.*

4.2.2 Backward direction

In this section we prepare the proof of the backward direction for the proof of Proposition 4.7. We first show that if an instance of Multistage Vertex Cover computed by Construction 1 is a yes-instance, then it is safe to assume that neither two vertices are deleted from nor added to a vertex cover in a consecutive step (we refer to these solutions as *smooth*, see Definition 4.11). Moreover, a vertex from C is only exchanged with another vertex from C and, at any time, there is exactly one vertex from C contained in the solution (similarly to the constructed solution in Lemma 4.9). We call these solutions *one-centered* (Definition 4.13).

We then prove that there must be a phase t for any one-centered solution that is deleting at least $\binom{k}{2}$ times a vertex from "past" sets $U_{t'}$, $t' \leq t$. This at hand, we prove that such a phase witnesses a clique of size k.

That a solution needs to contain at least one vertex from C at any time follows immediately from the fact that there is either an edge between two vertices in C or there is a vertex in C which is the center of a star with $k' + 1$ leaves.

▶ **Observation 4.10.** *Let $(\mathcal{G}, k', \ell)$ from Construction 1 be a* yes*-instance. Then for each solution $(S_1, \ldots, S_\tau)$ it holds true that $|S_i \cap C| \geq 1$ for all $i \in \{1, \ldots, \tau(\mathcal{G})\}$.*

In the remainder of this section we denote the vertices which are removed from the set S_{i-1} and added to the next set S_i in a solution $S = (\ldots, S_{i-1}, S_i, \ldots)$ by

$$S_{i-1} \diamondminus S_i := (S_{i-1} \setminus S_i, S_i \setminus S_{i-1}).$$

If $S_{i-1} \setminus S_i$ or $S_i \setminus S_{i-1}$ have size one, then we will omit the brackets of the singleton.

▶ **Definition 4.11.** *A solution $\mathcal{S} = (S_1, \ldots, S_\tau)$ for $(\mathcal{G}, k', \ell)$ from Construction 1 is* smooth *if for all $i \in \{2, \ldots, \tau\}$ we have $|S_{i-1} \setminus S_i| \leq 1$ and $|S_{i-1} \setminus S_i| \leq 1$.*

▶ **Observation 4.12.** *Let $(\mathcal{G}, k', \ell)$ from Construction 1 be a* yes*-instance. Then there is a smooth solution $(S_1, \ldots, S_\tau)$.*

Proof. By Observation 2.1, we know that there is a solution $\mathcal{S} = (S_1, \ldots, S_\tau)$ such that $|S_1| = k'$ and $k' - 1 \leq |S_i| \leq k'$ for all $i \in \{1, \ldots, \tau\}$. Hence, for all $i \in \{2, \ldots, \tau\}$ it holds true that $\big||S_i| - |S_{i-1}|\big| \leq 1$. It follows that $|S_{i-1} \setminus S_i| \leq 1$ and $|S_{i-1} \setminus S_i| \leq 1$, and thus, $\mathcal{S}$ is a smooth solution. ◀

Our goal is to prove the existence of the following type of solutions.

▶ **Definition 4.13.** *A smooth solution $\mathcal{S} = (S_1, \ldots, S_\tau)$ for $(\mathcal{G}, k', \ell)$ from Construction 1 is* one-centered *if*

- **(i)** *for all $i \in \{1, \ldots, \tau\}$ we have $|S_i \cap C| = 1$, and*
- **(ii)** *for all $i \in \{2, \ldots, \tau\}$ and $S_{i-1} \diamondminus S_i = (a, b)$ we have that $a \in C \iff b \in C$.*

We now show that in the output instance of Construction 1, there are solutions (if there is any) where $c_1 \in C$ is the only vertex from C in the first set of the solution.

▶ **Lemma 4.14 (⋆).** *Let $(\mathcal{G}, k', \ell)$ from Construction 1 be a* yes*-instance. Then there is a smooth solution $(S_1, \ldots, S_\tau)$ such that $S_1 \cap C = \{c_1\}$.*

Next, we show that there are solutions such that whenever we remove a vertex in C from the vertex cover, then we simultaneously add another vertex from C to the vertex cover. Formally, we prove the following.

▶ **Lemma 4.15 (⋆).** *Let $(\mathcal{G}, k', \ell)$ from Construction 1 be a* yes*-instance. Then there is a smooth solution $(S_1, \ldots, S_\tau)$ such that $S_1 \cap C = \{c_1\}$ and for all i with $S_{i-1} \diamondminus S_i = (a, c)$ and $c \in C$ we also have $a \in C$.*

Combining Observation 4.10 and Lemma 4.15, we can assume that given a yes-instance, there is a solution which is one-centered.

▶ **Corollary 4.16.** *Let $(\mathcal{G}, k', \ell)$ from Construction 1 be a* yes*-instance. Then, there is a solution $\mathcal{S}$ which is one-centered.*

Table 2 Note that $\varepsilon_i - \varepsilon_{i-1} = |F_i^t| - |F_{i-1}^t| - (|Y_i^t| - |Y_{i-1}^t|) - (f_i^t - f_{i-1}^t)$.

$S_{i-1}^t \diamondsuit S_i^t$		$\lvert F_i^t\rvert - \lvert F_{i-1}^t\rvert$	$-(\lvert Y_i^t\rvert - \lvert Y_{i-1}^t\rvert)$	$-(f_i^t - f_{i-1}^t)$	$\varepsilon_i - \varepsilon_{i-1}$
(u,b)	$b \in E$	$\in \{-1,0\}$	$\in \{0,1\}$	1	$\in \{0,1,2\}$
	$b \in \widehat{U}_{\kappa+1}$	$\in \{-1,0\}$	1	0	$\in \{0,1\}$
	$b \in V$, $b = \varnothing$	$\in \{-1,0\}$	1	1	$\in \{1,2\}$
(a,u)	$a \in E$	0	$\in \{1,2\}$	-1	$\in \{0,1\}$
	$a \in V$, $a = \varnothing$	0	1	-1	0
(a,v)	$a \in E$	0	$\in \{1,2\}$	0	$\in \{1,2\}$
	$a \in V$, $a = \varnothing$	0	1	0	1
(a,e)	$a \in V$	0	1	0	1
	$a \in E$, $a = \varnothing$	0	$\in \{0,1\}$	0	$\in \{0,1\}$

In the remainder of this section, for each $t \in \{1,\dots,\kappa+1\}$ let the union of U^i for all $i \le t$ be denoted by

$$\widehat{U}_t = \bigcup_{i=1}^{t} U^i.$$

We introduce further notation regarding a one-centered solution $\mathcal{S} := (S_1^1,\dots,S_{2m+1}^1 = S_1^2,\dots,\dots,S_1^\kappa,\dots,S_{2m+1}^\kappa)$ for $(\mathcal{G},k',\ell)$. Here, S_i^t is the i-th set of phase t and thus the $(2m(t-1)+i)$-th set of $\mathcal{S}$. We define the sets

$$Y_i^t := \{e_j \in S_i^t \cap E \mid 2j \ge i\} \quad \text{and} \quad F_i^t := \{j > i \mid S_{j-1}^t \diamondsuit S_j^t = (u,b) \text{ with } u \in \widehat{U}_t\}.$$

Set Y_i^t is the set of vertices e_j from E in S_i^t such that the corresponding layer for e_j in phase t is not before the layer with index i in phase t. Set F_i^t is the set of indices greater than i of layers from $\mathcal{G}$ in phase t where a vertex from $\widehat{U}_t$ is not carried over to the next layer's vertex cover. We now show that there is a phase t where $|F_1^t| \ge K$.

▶ **Lemma 4.17** (⋆). *Let $\mathcal{S} = (S_1^1,\dots,S_{2m+1}^1 = S_1^2,\dots,\dots,S_1^\kappa,\dots,S_{2m+1}^\kappa)$ be a one-centered solution to $(\mathcal{G},k',\ell)$ from Construction 1 being a* yes*-instance. Then, there is a $t \in \{1,\dots,\kappa\}$ such that $|F_1^t| \ge K$.*

In the remainder of this section the value

$$f_i^t := |S_i^t \cap \widehat{U}_{\kappa+1}| - K$$

describes the number of vertices in $\widehat{U}_{\kappa+1}$ which we could remove from S_i^t such that S_i^t is still a vertex cover for $G_{2m(t-1)+i}$ (the i-th layer of phase t). Observe that $f_i^t \ge 0$ for all $i \in \{1,\dots,2m+1\}$ and $t \in \{1,\dots,\kappa\}$, because we need in each layer exactly K vertices from $\widehat{U}_{\kappa+1}$ in the vertex cover.

We now derive an invariant which must be true in each phase.

▶ **Lemma 4.18** (⋆). *Let $\mathcal{S} = (S_1^1,\dots,S_{2m+1}^1 = S_1^2,\dots,\dots,S_1^\kappa,\dots,S_{2m+1}^\kappa)$ be a one-centered solution to $(\mathcal{G},k',\ell)$ from Construction 1 being a* yes*-instance. Then, for all $t \in \{1,\dots,\kappa\}$ and $i \in \{1,\dots,2m+1\}$, it holds true that $|F_i^t| - |Y_i^t| \le f_i^t$.*

Proof. Define $\varepsilon_i = |F_i^t| - |Y_i^t| - f_i^t$ for all $i \in \{1,\dots,2m+1\}$. We show that $\varepsilon_i - \varepsilon_{i-1} \ge 0$ for all $i \in \{1,\dots,2m+1\}$. Since $\mathcal{S}$ is one-centered, in Table 2 all relevant tuples for $S_{i-1}^t \diamondsuit S_i^t$ are shown.

Now assume towards a contradiction that there is a $j \in \{1, \ldots, 2m+1\}$ such that $\varepsilon_j > 0$. Since $\varepsilon_i - \varepsilon_{i-1} \geq 0$ for all $i \in \{1, \ldots, 2m+1\}$, we have that $\varepsilon_{2m+1} > 0 \iff |F^t_{2m+1}| - |Y^t_{2m+1}| > f^t_{2m+1}$. By definition, we have that $|F^t_{2m+1}| = 0$ and $|Y^t_{2m+1}| = 0$ Moreover, since $\mathcal{S}$ is a solution and each vertex cover needs at least K vertices from $\widehat{U}_\tau$, we have that $f^t_{2m+1} \geq 0$. It follows that $0 = |F^t_{2m+1}| - |Y^t_{2m+1}| > f^t_{2m+1} \geq 0$, yielding a contradiction. ◀

Next, we prove that in a phase t with $|F^t_1| \geq K$, there are at most k vertices from V contained in the union of the vertex covers of phase t.

▶ **Lemma 4.19** (⋆). *Let $\mathcal{S} = (S^1_1, \ldots, S^1_{2m+1} = S^2_1, \ldots, \ldots, S^\kappa_1, \ldots, S^\kappa_{2m+1})$ be a one-centered solution to $(\mathcal{G}, k', \ell)$ from Construction 1 being a* yes*-instance, and let $t \in \{1, \ldots, \kappa\}$ be such that $|F^t_1| \geq K$. Then, $|\bigcup_{i=1}^{2m+1} S^t_i \cap V| \leq k$.*

Proof. From Lemma 4.18, we know that $|Y^t_1| \geq K - f^t_1$. Let $|Y^t_1| = K - f^t_1 + \lambda$ for some $\lambda \in \mathbb{N}_0$, and $\varepsilon_i = |F^t_i| - |Y^t_i| - f^t_i$, for all $i \in \{1, \ldots, 2m+1\}$.

We now show that there are at most λ layers where we exchange a vertex currently in the vertex cover with a vertex in V. Let $i \in \{2, \ldots, 2m+1\}$ such that $S^t_{i-1} \diamondsuit S^t_i = (a, v)$ with $v \in V$. From Table 2 (recall that one-centered solutions are smooth), we know that $\varepsilon_i \geq \varepsilon_{i-1} + 1$.

Assume towards a contradiction that there are $\lambda + 1$ many of these exchanges. Then, there is a $j \in \{1, \ldots, 2m+1\}$ such that

$$\varepsilon_j \geq \varepsilon_1 + \lambda + 1 = |F^t_1| - |Y^t_1| - f^t_1 + \lambda + 1 \geq K - (K - f^t_1 + \lambda) - f^t_1 + \lambda + 1 \geq 1$$
$$\iff |F^t_j| - |Y^t_j| > f^t_j.$$

This contradicts the invariant of Lemma 4.18.

In the beginning of phase t, we have at most $k - \lambda$ vertices from V in the vertex cover, because $|S^t_1 \cap V| \leq K + k - |Y^t_1| - f^t_1 = K + k - (K - f^t_1 + \lambda) - f^t_1 = k - \lambda$. Since there are at most λ many exchanges $S^t_{i-1} \diamondsuit S^t_i = (a, v)$ where $v \in V$ and $i \in \{2, \ldots, 2m+1\}$, we know that the vertex set $\bigcup_{i=1}^{2m+1} S^t_i \cap V$ is of size at most k. ◀

4.2.3 Proof of Proposition 4.7

Proof of Proposition 4.7. Let (G, k) be an instance of Clique and $(\mathcal{G}, k', \ell)$ be the instance of MSVC resulting from Construction 1. Observe that Construction 1 runs in polynomial time. We prove that (G, k) is a yes-instance of Clique if and only if $(\mathcal{G}, k', \ell)$ is a yes-instance of MSVC.

($\Rightarrow$) It follows from Lemma 4.9 that $(\mathcal{G}, k', \ell)$ is a yes-instance if (G, k) is a yes-instance.

($\Leftarrow$) Let $(\mathcal{G}, k', \ell)$ be a yes-instance. From Corollary 4.16 it follows that there is a one-centered solution $\mathcal{S} = (S^1_1, \ldots, S^1_{2m+1} = S^2_1, \ldots, \ldots, S^\kappa_1, \ldots, S^\kappa_{2m+1})$ for $(\mathcal{G}, k', \ell)$. By Lemma 4.17, there is a $t \in \{1, \ldots, \kappa\}$ such that $|F^t_1| \geq K = \binom{k}{2}$. By Lemma 4.19, we know that $|\bigcup_{i=1}^{2m+1} S^t_i \cap V| \leq k$. Now we identify the clique of size k in G. Since $|F^t_1| \geq K$, we know that, by Construction 1, at least K layers are covered by vertices in $V \cup E \cup \widehat{U}_{\kappa+1} \cup \{c^t_{2j+1} \mid j \in \{1, \ldots, m\}\}$ in phase t. Note that each of these layers corresponds to an edge $e = \{v, w\}$ in G and that we need in particular the vertices v and w in the vertex cover. Since we have at most k vertices in $\bigcup_{i=1}^{2m+1} S^t_i \cap V$, these vertices induce a clique of size k in G.

Finally, following Remark 4.8, we can turn each layer into a tree preserving equivalence. The W[1]-hardness of Clique [9] regarding k and that $k' \in O(k^2)$ then finish the proof. ◀

5 On Efficient Data Reduction

In this section, we study the possibility of effective data reduction for MSVC when parameterized by k, τ, and $k+\tau$, that is, the possible existence of problem kernels of polynomial size. We prove that unless coNP $\subseteq$ NP/poly, MSVC admits no problem kernel of size polynomial in k (Section 5.1). Yet, when combining k and τ, we prove a problem kernel of size $O(k^2\tau)$ (Section 5.2). Moreover, we prove a problem kernel of size 5τ when each layer consists of only one edge (Section 5.3). Recall that MSVC is para-NP-hard regarding τ even if each layer is a tree.

5.1 No problem kernel of size polynomial in k

We prove that if

(i) each layer consists only of one edge and $\ell = 1$, or

(ii) if each layer is planar and $\ell \geq 2k$,

then MSVC admits no kernel of size polynomial in k unless coNP $\subseteq$ NP/poly. Recall that MSVC parameterized by k is fixed-parameter tractable in case of (i) (see Observation 2.5), while we left open whether it also holds true in case (ii).

▶ **Theorem 5.1.** *Unless* coNP $\subseteq$ NP/poly, MSVC *admits no polynomial kernel when parameterized by k, even if*

(i) *each layer consists of one edge and $\ell = 1$, or if*

(ii) *each layer is planar and $\ell \geq 2k$.*

We prove Theorem 5.1 using AND-compositions.

▶ **Definition 5.2.** *An* AND-composition *for a parameterized problem L is an algorithm that given p instances $(x_1, k), \ldots, (x_p, k)$ of L, computes in time polynomial in $\sum_{i=1}^{p} |x_i|$ an instance (y, k') of L such that*

(i) *$(y, k') \in L$ if and only if $(x_i, k) \in L$ for all $i \in \{1, \ldots, p\}$, and*

(ii) *k' is polynomially upper-bounded in k.*

Drucker [10] showed that if a parameterized problem whose unparameterized version is NP-hard admits an AND-composition, then coNP $\subseteq$ NP/poly. Note that coNP $\subseteq$ NP/poly implies a collapse of the polynomial-time hierarchy to its third level [23]. In the proof of Theorem 5.1(i), we use the following.

▶ **Construction 2.** Let $(\mathcal{G}_1 = (V, \mathcal{E}_1, \tau_1), k, \ell), \ldots, (\mathcal{G}_p = (V, \mathcal{E}_p, \tau_p), k, \ell)$ be p instances of MSVC where each layer consists of one edge and $\ell = 1$. We construct an instance $(\mathcal{G} = (V, \mathcal{E}, \tau), k, \ell)$ of MSVC as follows. Denote by $(G_1^i, \ldots, G_{\tau_i}^i)$ the sequence of layers of $\mathcal{G}_i$. Initially, let $\mathcal{G}$ be the temporal graph with layer sequence $((G_j^i)_{1 \leq j \leq \tau_i})_{1 \leq i \leq p}$. Next, for each $i \in \{1, \ldots, p-1\}$, insert between $G_{\tau_i}^i$ and G_1^{i+1} the sequence $(H_1^i, H_2^i, \ldots, H_{2k}^i) = (G_{\tau_i}^i, G_1^{i+1}, \ldots, G_1^{i+1})$. This finishes the construction. Note that $\tau = 2k(p-1) + \sum_{i=1}^{p} \tau_i$. ⌟

Construction 2 gives an AND-composition used in the proof of Theorem 5.1(i).

▶ **Proposition 5.3** (⋆). MSVC *where each layer consists of one edge and $\ell = 1$ AND-composes into itself parameterized by k.*

Turning a set of input instances of Vertex Cover on planar graphs (this is equivalent to MSVC with one layer which is a planar graph) into a sequence gives an AND-composition used in the proof of Theorem 5.1(ii).

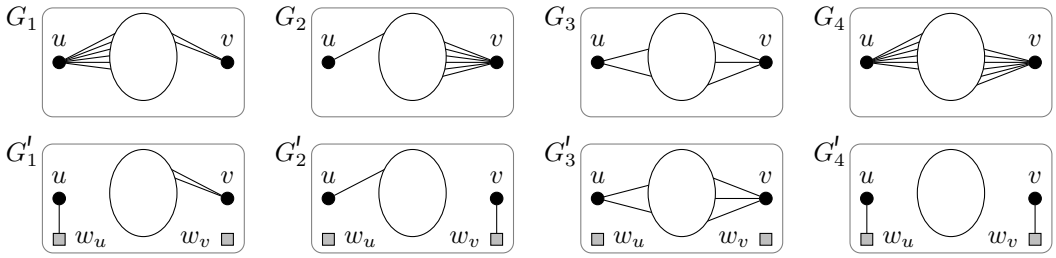


Figure 2 Illustration of Reduction Rule 2, exemplified for two vertices u, v and $k = 5$. The vertices w_v, w_u (gray squares) are introduced by the application of Reduction Rule 2.

▶ **Proposition 5.4** (⋆). MSVC *with one layer being a planar graph AND-composes into* MSVC *parameterized by* k *with* $\ell \geq 2k$ *and each layer being planar.*

Proof of Theorem 5.1. Using Drucker's result for AND-copositions [10], Propositions 5.3 and 5.4 prove Theorem 5.1(i) and (ii), respectively. Recall that MSVC where each layer consists of one edge (Theorem 3.1) and Vertex Cover on planar graphs [14] are NP-hard. ◀

5.2 A problem kernel of size $O(k^2\tau)$

MSVC remains NP-hard for $\tau = 2$, even if each layer is a tree (Theorem 3.1). Moreover, MSVC does not admit a problem kernel of size polynomial in k, even if each layer consists of one edge (Theorem 5.1). Yet, when combining both parameters we obtain a problem kernel of cubic size.

▶ **Theorem 5.5.** *There is an algorithm that maps any instance* $(\mathcal{G} = (V, \mathcal{E}, \tau), k, \ell)$ *of* MSVC *in time* $O(|V|^2\tau)$ *to an instance* $(\mathcal{G}', k, \ell)$ *of* MSVC *with at most* $2k^2\tau$ *vertices and* $k^2\tau$ *temporal edges.*

To prove Theorem 5.5, we apply three polynomial-time data reduction rules. These reduction rules can be understood as temporal variants of the folklore reduction rules for Vertex Cover. Our first reduction rule is immediate.

▶ **Reduction Rule 1** (Isolated vertices). *If there is some vertex* $v \in V$ *such that* $e \cap v = \emptyset$ *for all* $e \in E(G_\downarrow)$*, then delete* v*.*

For Vertex Cover when asking for a vertex cover of size q, there is the well-known reduction rule dealing with high-degree vertices: If there is a vertex v of degree larger than q, then delete v and its incident edges and decrease q by one. For MSVC a high-degree vertex can only appear in some layers, and hence deleting this vertex is in general not correct. However, there is a temporal variant of the high-degree rule as follows.

▶ **Reduction Rule 2** (High degree). *If there exists a vertex* v *such that there is an inclusion-maximal subset* $J \subseteq \{1, \ldots, \tau\}$ *such that* $\deg_{G_i}(v) > k$ *for all* $i \in J$*, then add a vertex* w_v *to* V *and for each* $i \in J$*, remove all edges incident to* v *in* G_i*, and add the edge* $\{v, w_v\}$*.*

See Figure 2 for an illustration. We now show how Reduction Rule 2 can be applied and that it does not turn a `yes`-instance into a `no`-instance or vice versa.

▶ **Lemma 5.6** (⋆). *Reduction Rule 2 is correct and exhaustively applicable in* $O(|V|^2\tau)$ *time.*

Similarly as in the reduction rules for Vertex Cover, we now count the number of edges in each layer: If more than k^2 edges are contained in one layer, then no set of k vertices each of degree at most k can cover more than k^2 edges.

▶ **Reduction Rule 3** (no-instance). *If none of Reduction Rules 1 and 2 is applicable and there is a layer with more than k^2 edges, then output a trivial no-instance.*

We are ready to prove that when none of the Reduction Rules 1 to 3 can be applied, then the instance contains "few" vertices and temporal edges.

▶ **Lemma 5.7** (⋆). *Let $(\mathcal{G}, k, \ell)$ be an instance of* MSVC *such that none of Reduction Rules 1 to 3 is applicable. Then $\mathcal{G}$ consists of at most $2k^2\tau(\mathcal{G})$ vertices and $k^2\tau(\mathcal{G})$ temporal edges.*

We are ready to prove the main result of this section.

Proof of Theorem 5.5. Apply Reduction Rules 1 to 3 exhaustively in $O(|V|^2\tau)$ time to obtain an equivalent instance $(\mathcal{G}', k, \ell)$. Due to Lemma 5.7, $\mathcal{G}'$ consists of at most $2k^2\tau$ vertices and at most $k^2\tau$ temporal edges. ◀

5.3 A problem kernel of size 5τ

MSVC when each layer is a tree does not admit a problem kernel of any size in τ unless P = NP. Yet, when each layer consists of only one edge, then each instance of MSVC contains at most τ edges and, hence, at most 2τ non-isolated vertices. Thus, MSVC admits a straight-forward problem kernel of size linear in τ.

▶ **Observation 5.8.** *Let $(\mathcal{G} = (V, \mathcal{E}, \tau), k, \ell)$ be an instance of* MSVC *where each layer consists of one edge. Then we can compute in $O(|V| \cdot \tau)$ time an instance $(\mathcal{G}', k, \ell)$ of size at most 5τ.*

Proof. Observe that we can immediately output a trivial yes-instance if $k \geq \tau$ (Observation 2.1) or $\ell \geq 2$ (Observation 2.5). Hence, assume that $k \leq \tau - 1$ and $\ell \leq 1$. Apply Reduction Rule 1 exhaustively on $(\mathcal{G}, k, \ell)$ to obtain $(\mathcal{G}', k, \ell)$. Since there are τ edges in $\mathcal{G}$, there are at most 2τ vertices in $\mathcal{G}'$. It follows that the encoding length of $(\mathcal{G}', k, \ell)$ is at most 5τ. ◀

6 Conclusion

We introduced Multistage Vertex Cover, proved it to be NP-hard even on restricted inputs, and studied its parameterized complexity regarding the natural parameters k, ℓ, and τ (each given as input). We leave open whether MSVC parameterized by k is fixed-parameter tractable when each layer consists of only one edge (see Table 1). Moreover, it is open whether MSVC remains NP-hard on two layers each being a path (that is, strengthening Theorem 3.1(i)).

References

1 Faisal N. Abu-Khzam, Judith Egan, Michael R. Fellows, Frances A. Rosamond, and Peter Shaw. On the parameterized complexity of dynamic problems. *Theor. Comput. Sci.*, 607:426–434, 2015.

2 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal Vertex Cover with a Sliding Time Window. In *Proc. of 45th ICALP*, volume 107 of *LIPIcs*, pages 148:1–148:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

3 Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic Parameterized Problems and Algorithms. In *Proc. of 44th ICALP*, volume 80 of *LIPIcs*, pages 41:1–41:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

4 Evripidis Bampis, Bruno Escoffier, Michael Lampis, and Vangelis Th. Paschos. Multistage Matchings. In *Proc. of 16th SWAT*, volume 101 of *LIPIcs*, pages 7:1–7:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

5 Evripidis Bampis, Bruno Escoffier, and Alexandre Teiller. Multistage Knapsack. In *Proc. of 44th MFCS*, volume 138 of *LIPIcs*, pages 22:1–22:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

6 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via Sampling with Applications to Finding Matchings and Related Problems in Dynamic Graph Streams. In *Proc. of 27th SODA*, pages 1326–1344. SIAM, 2016.

7 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

8 Reinhard Diestel. *Graph Theory*, volume 173 of *GTM*. Springer, 5th edition, 2016.

9 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

10 Andrew Drucker. New Limits to Classical and Quantum Instance Compression. *SIAM J. Comput.*, 44(5):1443–1479, 2015.

11 David Eisenstat, Claire Mathieu, and Nicolas Schabanel. Facility Location in Evolving Metrics. In *Proc. of 41st ICALP*, volume 8573 of *LNCS*, pages 459–470. Springer, 2014.

12 Herbert Fleischner, Gert Sabidussi, and Vladimir I. Sarvanov. Maximum independent sets in 3- and 4-regular Hamiltonian graphs. *Discrete Math.*, 310(20):2742–2749, 2010.

13 Till Fluschnik, Hendrik Molter, Rolf Niedermeier, Malte Renken, and Philipp Zschoche. Temporal Graph Classes: A View Through Temporal Separators. *Theor. Comput. Sci.*, 2019. In press.

14 M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some Simplified NP-Complete Graph Problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976.

15 Parikshit Gopalan, Phokion G Kolaitis, Elitza Maneva, and Christos H Papadimitriou. The connectivity of Boolean satisfiability: computational and structural dichotomies. *SIAM J. Comput.*, 38(6):2330–2355, 2009.

16 Anupam Gupta, Kunal Talwar, and Udi Wieder. Changing Bases: Multistage Optimization for Matroids and Matchings. In *Proc. of 41st ICALP*, volume 8572 of *LNCS*, pages 563–575. Springer, 2014.

17 Sepp Hartung and Rolf Niedermeier. Incremental list coloring of graphs, parameterized by conservation. *Theor. Comput. Sci.*, 494:86–98, 2013.

18 Takehiro Ito, Erik D Demaine, Nicholas JA Harvey, Christos H Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theor. Comput. Sci.*, 412(12-14):1054–1065, 2011.

19 Yoichi Iwata and Keigo Oka. Fast Dynamic Graph Algorithms for Parameterized Problems. In *Proc. of 12th SWAT*, volume 8503 of *LNCS*, pages 241–252. Springer, 2014.

20 R. Krithika, Abhishek Sahu, and Prafullkumar Tale. Dynamic Parameterized Problems. *Algorithmica*, 80(9):2637–2655, 2018.

21 Amer Mouawad, Naomi Nishimura, Venkatesh Raman, and Sebastian Siebertz. Vertex cover reconfiguration and beyond. *Algorithms*, 11(2):20, 2018.

22 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.

23 Chee-Keng Yap. Some Consequences of Non-Uniform Conditions on Uniform Classes. *Theor. Comput. Sci.*, 26:287–300, 1983.

Parameterized Complexity of Edge-Coloured and Signed Graph Homomorphism Problems

Florent Foucaud
Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans Cedex 2, France
Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France

Hervé Hocquard
Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France

Dimitri Lajou
Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France

Valia Mitsou
Université Paris-Diderot, IRIF, CNRS, 75205, Paris, France

Théo Pierron
Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France
DIMEA, Masaryk University, 60200 Brno, Czech republic

Abstract

We study the complexity of graph modification problems with respect to homomorphism-based colouring properties of edge-coloured graphs. A homomorphism from an edge-coloured graph G to an edge-coloured graph H is a vertex-mapping from G to H that preserves adjacencies and edge-colours. We consider the property of having a homomorphism to a fixed edge-coloured graph H, which generalises the classic vertex-colourability property. The question we are interested in is the following: given an edge-coloured graph G, can we perform k graph operations so that the resulting graph admits a homomorphism to H? The operations we consider are vertex-deletion, edge-deletion and switching (an operation that permutes the colours of the edges incident to a given vertex). Switching plays an important role in the theory of signed graphs, that are 2-edge-coloured graphs whose colours are the signs $+$ and $-$. We denote the corresponding problems (parameterized by k) by Vertex Deletion-H-Colouring, Edge Deletion-H-Colouring and Switching-H-Colouring. These problems generalise the extensively studied H-Colouring problem (where one has to decide if an input graph admits a homomorphism to a fixed target H). For 2-edge-coloured H, it is known that H-Colouring already captures the complexity of all fixed-target Constraint Satisfaction Problems.

Our main focus is on the case where H is an edge-coloured graph of order at most 2, a case that is already interesting since it includes standard problems such as Vertex Cover, Odd Cycle Transversal and Edge Bipartization. For such a graph H, we give a PTime/NP-complete complexity dichotomy for all three Vertex Deletion-H-Colouring, Edge Deletion-H-Colouring and Switching-H-Colouring problems. Then, we address their parameterized complexity. We show that all Vertex Deletion-H-Colouring and Edge Deletion-H-Colouring problems for such H are FPT. This is in contrast with the fact that already for some H of order 3, unless PTime = NP, none of the three considered problems is in XP, since 3-Colouring is NP-complete. We show that the situation is different for Switching-H-Colouring: there are three 2-edge-coloured graphs H of order 2 for which Switching-H-Colouring is W[1]-hard, and assuming the ETH, admits no algorithm in time $f(k)n^{o(k)}$ for inputs of size n and for any computable function f. For the other cases, Switching-H-Colouring is FPT.

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

Keywords and phrases Graph homomorphism, Graph modification, Edge-coloured graph, Signed graph

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.15

Related Version A full version of the paper is available at `https://arxiv.org/abs/1910.01099`.

Funding This research was financed by the ANR project HOSIGRA (ANR-17-CE40-0022) and the IFCAM project "Applications of graph homomorphisms" (MA/IFCAM/18/39).

1 Introduction

Graph colouring problems such as k-Colouring are among the most fundamental problems in algorithmic graph theory. Problem H-Colouring is a homomorphism-based generalisation of k-Colouring that is extensively studied [8, 14, 18, 25]. Considering a fixed graph H, in H-Colouring one asks whether an input graph G admits a homomorphism (an edge-preserving vertex-mapping) to H. k-Colouring is the same problem as K_k-Colouring, where K_k is the complete graph of order k (the order of a graph is its number of vertices).

We will consider parameterized variants of H-Colouring where H is an edge-coloured graph. We say that a graph is *t-edge-coloured* if its edges are coloured with at most t colours. We allow loops and multiple edges, but multiple edges of the same colour are irrelevant in H. We sometimes give actual colour names to the colours: red, blue, green. For 2-edge-coloured graphs, we will use red and blue as the two edge colours. A standard uncoloured graph can be seen as 1-edge-coloured. For two edge-coloured graphs G and H, a homomorphism from G to H is a vertex-mapping $\varphi : V(G) \to V(H)$ such that, if xy is an edge of colour i in G, then $\varphi(x)\varphi(y)$ is an edge of colour i in H. Whenever such a φ exists, we say that G maps to H, and we write $G \xrightarrow{ec} H$.

The H-Colouring problems are well-studied, see for example [1, 2, 3, 4, 5]. They are special cases of *Constraint Satisfaction Problems* (CSPs). A large set of CSPs can be modeled by homomorphisms of general relational structures to a fixed relational structure H [14]. The corresponding decision problem is noted H-CSP. When H has only binary relations, H can be seen as an edge-coloured graph (a relation corresponds to the set of edges of a given colour) and H-CSP is exactly H-Colouring. The complexity of H-CSP has been the subject of intensive research in the last decades, since Feder and Vardi conjectured in [14] that H-CSP is either PTime or NP-complete – a statement that became known as the Dichotomy Conjecture. The latter conjecture was recently solved in [7, 30] independently; the criterion for H-CSP to be in PTime is based on certain algebraic properties of H. Nevertheless, determining whether a structure H satisfies this criterion is not an easy task (even for targets as simple as oriented trees [8]). Thus, the study of more simple and elegant complexity classifications for relevant special cases is of high importance.

The complexity of H-Colouring when H is uncoloured is well-understood: it is in PTime if H contains a loop or is bipartite; otherwise it is NP-complete [18]. This was one of the early dichotomy results in the area. On the other hand, when H is a 2-edge-coloured graph, it was proved that the class of H-Colouring problems captures the difficulty of the whole class of H-CSP problems [5], and thus the dichotomy classification for this class of problems is expected to be much more intricate.

Our goal is to study generalisations of H-Colouring problems for edge-coloured graphs by enhancing them as *modification problems*. In this setting, given a graph property $\mathcal{P}$ and a graph operation π, the graph modification problem for $\mathcal{P}$ and π asks whether an input graph G can be made to satisfy property $\mathcal{P}$ after applying operation π a given number k of times. This is a classic setting studied extensively both in the realms of classical and parameterized complexity, see for example [9, 22, 23, 28]. In this context, the most studied graph operations are vertex deletion (VD) and edge-deletion (ED), see the seminal papers [23, 28].

For a fixed graph H, let $\mathcal{P}(H)$ denote the property of admitting a homomorphism to H. Certain standard computational problems can be stated as graph modification problems to $\mathcal{P}(H)$. For example, Vertex Cover is the graph modification problem for property $\mathcal{P}(K_1)$ and operation VD. Similarly, Odd Cycle Transversal and Edge Bipartization are the graph modification problems for $\mathcal{P}(K_2)$ and VD, and $\mathcal{P}(K_2)$ and ED, respectively.

When considering edge-coloured graphs with only two edge-colours, another operation of interest is *switching*: to switch at a vertex v is to change the colour of all edges incident with v. (Note that a loop does not change its colour under switching.) This operation is of prime importance in the context of signed graphs. A *signed graph* is a 2-edge-coloured graph in which the two colours are denoted by signs ($+$ and $-$). A graph is called *balanced* if it can be switched to be all-positive. The concepts of signed graphs, balance and switching, were introduced and developed in [17, 29] and have many interesting applications, in particular in social networks and biological dynamical systems (see [19] and the references therein).

The switching operation plays an important role in the study of homomorphisms of signed graphs, a concept defined in [26] which has many connections to deep questions in structural graph theory. In their definition, before mapping the vertices, one may perform any number of switchings. (Note that when switching at a set S of vertices of a signed graph G, the order does not matter: ultimately, only the edges between S and its complement $V(G) \setminus S$ change their sign.) The algorithmic complexity of this problem was studied in [5, 6, 16]. Herein, we will consider edge-coloured graph modification problems for property $\mathcal{P}(H)$ (for fixed edge-coloured graphs H) and for graph operations VD, ED and SW.

A parameterized problem is a decision problem with a parameter of the input. It is *fixed parameter tractable* (FPT) if for any input I with parameter value k, it can be solved in time $O(f(k)|I|^c)$ for a computable function f and integer c. It is in the class XP if it can be solved in time $|I|^{g(k)}$ for a computable function g. It is W[1]-hard if all problems in the class W[1] can be reduced in FPT time to it. For more details, see the books [11, 12]. Let us now formally define the problems of interest to us (the parameter is always k).

Vertex Deletion-(resp. Edge Deletion)-H-Colouring	**Parameter:** k.
Input: An edge-coloured graph G, an integer k.	
Question: Is there a set S of k vertices (resp. edges) of G such that $(G - S) \xrightarrow{ec} H$?	

Switching-H-Colouring	**Parameter:** k.
Input: A 2-edge-coloured graph G, an integer k.	
Question: Is there a set S of k vertices of G such that the 2-edge-coloured graph G' obtained from G by switching at every vertex of S satisfies $G' \xrightarrow{ec} H$?	

In the study of the three above problems, one may assume that H is a *core* (that is, H does not have a homomorphism to a proper subgraph of itself). Indeed, it is well-known that for any subgraph H' of H with $H \xrightarrow{ec} H'$, we have $G \xrightarrow{ec} H$ if and only if $G \xrightarrow{ec} H'$ [3].

Of course, whenever H-Colouring is NP-complete, all three above problems are NP-complete, even when $k = 0$, and so they are not in XP (unless PTime = NP). This is for example the case when H is a monochromatic triangle: then we have 3-Colouring. Thus, from the point of view of parameterized complexity, it is of primary interest to consider these problems for edge-coloured graphs H such that H-Colouring is in PTime. (In that case a simple brute-force algorithm iterating over all k-subsets of vertices of G implies that the three problems are in XP.) For classic graphs, the only cores H for which H-Colouring is in PTime are the three connected graphs with at most one edge (a single vertex with no edge, a single vertex with a loop, two vertices joined by an edge), so in that case the interest of these problems is limited. However, for many interesting families of edge-coloured

graphs H, the problem H-Colouring is in PTime, and the class of such graphs H is not very well-understood, see [2, 3, 4]. Even when H is a 2-edge-coloured cycle, tree or complete graph, there are infinitely many H with H-Colouring NP-complete and infinitely many H where it is in PTime [2].

Recall that when H is a single vertex with no loop, Vertex Deletion-H-Colouring is exactly Vertex Cover. If H has a single edge, Vertex Deletion-H-Colouring and Edge Deletion-H-Colouring are Odd Cycle Transversal and Edge Bipartization, respectively. For H consisting of a single (blue) loop, Switching-H-Colouring for $k = |V(G)|$ consists in checking whether the given 2-edge-coloured graph G is balanced (a problem that is in PTime [5]). More generally, Switching-H-Colouring for 2-edge-coloured graphs H and $k = |V(G)|$ is exactly the problem Signed H-Colouring studied in [5, 6, 16].

Related work. Several works address the parameterized complexity of graph colouring problems. In [25], the vertex-deletion variant of H-List-Colouring is studied. Graph modification problems for Colouring in specific graph classes and for operations VD and ED are considered, for example in [10] (bipartite graphs, split graphs) and [27] (comparability graphs). Graph colouring problems parameterized by structural parameters are considered in [20]. Algorithmic problems relative to the operation of *Seidel switching* have been considered. Given a (classic) graph G, the Seidel switching operation performed at a vertex exchanges all adjacencies and non-adjacencies of v. This can be seen as performing a switching operation in a 2-edge-coloured complete graph, where blue edges are the actual edges of G, and red edges are its non-edges. In [13, 21], the complexity of graph modification problems with respect to the Seidel switching operation and the property of being a member of certain graph classes has been studied. Our work on Switching-H-Colouring problems can be seen as a variation of these problems, generalised to arbitrary 2-edge-coloured graphs.

Our results. We study the classic and parameterized complexities of the three problems Vertex Deletion-H-Colouring, Edge Deletion-H-Colouring and Switching-H-Colouring. Our focus is on t-edge-coloured graphs H of order at most 2 with t an integer ($t = 2$ for Switching-H-Colouring). Despite having just two vertices, H-Colouring for such H is interesting and nontrivial; it is proved to be in PTime by two different nontrivial methods, see [1, 4]. Thus, the three considered problems are in XP for such H. (Recall that for suitable 1-edge-coloured graphs H of order 1 or 2, Vertex Deletion-H-Colouring and Edge Deletion-H-Colouring include Vertex Cover and Odd Cycle Transversal.)

We completely classify the classical complexity of Vertex Deletion-H-Colouring when H is a t-edge-coloured graph of arbitrary order: it is either trivially in PTime or NP-complete. It turns out that all Vertex Deletion-H-Colouring problems are FPT when H has order at most 2. To prove this, we extend a method from [4] and reduce the problem to an FPT variant of 2-Sat.

For Edge Deletion-H-Colouring, a classical complexity dichotomy seems more difficult to obtain, as there are nontrivial PTime cases. We perform such a classification when H is a t-edge-coloured graph of order at most 2. Similar 2-Sat-based arguments as for Vertex Deletion-H-Colouring give a FPT algorithm for Edge Deletion-H-Colouring when H has order at most 2.

For Switching-H-Colouring when H is a 2-edge-coloured graph, the classical dichotomy is again more difficult to obtain. We perform such a classification by using some characteristics of the switch operation and by giving some reductions to well-known NP-complete problems. In contrast to the two previous cases for the parameterized complexity, we show that for three graphs H of order 2, Switching-H-Colouring is already W[1]-hard

Table 1 Overview of our main results, sorted by problem and by type of classification.

Problem	Vertex-Del.-H-Col.	Edge-Del.-H-Col.	Switching-H-Col.
P vs NP	Dichotomy for all graphs (Cor. 7)	Dichotomy when $\|V(H)\| \leq 2$ (Th. 8)	Dichotomy when $\|V(H)\| \leq 2$ (Th. 9)
FPT vs W-hard when $\|V(H)\| \leq 2$	All FPT (Th. 12)	All FPT (Th. 12)	Dichotomy (Th. 13 and 14)

(and cannot be solved in time $f(k)|G|^{o(k)}$ for any function f, assuming the ETH[1]). For all other 2-edge-coloured graphs of order 2, we prove that Switching-H-Colouring is FPT. Table 1 presents a brief overview of our results.

Our paper is structured as follows. In Section 2, we state some definitions and make some preliminary observations in relation with the literature. In Section 3, we study the classical complexity of the three considered problems. We address their parameterized complexity in Section 4. Finally, we conclude in Section 5.

2 Preliminaries and known results

2.1 Some known complexity dichotomies

Recall that whenever H-Colouring is NP-complete, Vertex Deletion-H-Colouring, Edge Deletion-H-Colouring and Switching-H-Colouring are NP-complete (even for $k = 0$), and thus are not in XP, unless PTime = NP. For example, this is the case when H is a monochromatic triangle. When Signed H-Colouring (this is Switching-H-Colouring for $k = |V(G)|$, see [5]) is NP-complete, then Switching-H-Colouring is NP-complete (but could still be in XP or FPT).

On the other hand, when H-Colouring is in PTime, all three problems are in XP for parameter k (by a brute-force algorithm iterating over all k-subsets of vertices of G, performing the operation on these k vertices, and then solving H-Colouring):

▶ **Proposition 1.** *Let H be an edge-coloured graph such that H-Colouring is in PTime. Then,* Vertex Deletion-H-Colouring, Edge Deletion-H-Colouring *and* Switching-H-Colouring *can be solved in time $|G|^{O(k)}$.*

When $k = 0$ and H is 1-coloured, we have the following classic theorem.

▶ **Theorem 2** (Hell and Nešetřil [18])**.** *Let H be a 1-edge-coloured graph. H-Colouring is in PTime if the core of H has at most one edge (H is bipartite or has a loop), and NP-complete otherwise.*

There is no analogue of Theorem 2 for edge-coloured graphs. In fact, it is proved in [5] that a dichotomy classification for H-Colouring restricted to 2-edge-coloured H would imply a dichotomy for all fixed-target CSP problems. Thus, no simple combinatorial classification is expected to exist. In fact, even for trees, cycles or complete graphs, such classifications are not easy to come by [2]. However, some classifications exist for certain classes of graphs H, such as those of order at most 2 [1, 4] or paths [3].

[1] The Exponential Time Hypothesis, ETH, postulates that 3-SAT cannot be solved in time $2^{o(n)}(n+m)^c$, where n and m are the input's number of variables and clauses, and c is any integer [24].

For Switching-H-Colouring with $k = |V(G)|$, (that is, Signed H-Colouring), we have the following (where the *switching core* of a 2-edge-coloured graph is a notion of core where an arbitrary number of switchings can be performed before the self-mapping):

▶ **Theorem 3** (Brewster et al. [5, 6]). *Let H be a signed graph.* Signed H-Colouring *is in PTime if the switching core of H has at most two edges, and NP-complete otherwise.*

Note that 2-edge-coloured graphs where the switching core has at most two edges either have one vertex (with zero loop, one loop or two loops of different colours), or two vertices (with either one edge or two parallel edges of different colours joining them) [5]. (If there are two vertices joined by one edge and a loop at one of the vertices, we can switch at the non-loop vertex if necessary to obtain one edge-colour, and then retract the whole graph to the loop-vertex, so this is not a core.)

2.2 Homomorphism dualities and FPT time

For a t-edge-coloured graph H, we say that H has the *duality property* if there is a set $\mathcal{F}(H)$ of t-edge-coloured graphs such that, for any t-edge-coloured graph G, $G \xrightarrow{ec} H$ if and only if no graph F of $\mathcal{F}(H)$ satisfies $F \xrightarrow{ec} G$. If $\mathcal{F}(H)$ is finite, we say that H has the *finite duality property*. If checking whether any graph F in $\mathcal{F}(H)$ satisfies $F \xrightarrow{ec} G$ (for an input edge-coloured graph G) is in PTime, we say that H has the *polynomial duality property*. This is in particular the case when $\mathcal{F}(H)$ is finite. For such H, H-Colouring is in PTime. This topic is explored in detail for edge-coloured graphs in [1]. By a simple bounded search tree argument, we get the following:

▶ **Proposition 4.** *Let H be an edge-coloured graph with the finite duality property. Then,* Vertex Deletion-H-Colouring, Edge Deletion-H-Colouring *and* Switching-H-Colouring *are FPT.*

Proof. First, we search for all appearances of homomorphic images of graphs in $\mathcal{F}(H)$ (there are at most $f(\mathcal{F}(H))$ such images for some exponential function f), which we call *obstructions*. This takes time at most $f(\mathcal{F}(H))n^{m_v}$, where $m_v = \max\{|V(F)|, F \in \mathcal{F}(H)\}$. Then, we need to get rid of each obstruction. For Vertex Deletion-H-Colouring (resp. Edge Deletion-H-Colouring), we need to delete at least one vertex (resp. edge) in each obstruction, thus we can branch on all m_v (resp. m_v^2) possibilities. For Switching-H-Colouring, we need to switch at least one of the vertices of the obstruction (but then update the list of obstructions, as we may have created a new one). In all cases, this gives a search tree of height k and degree bounded by a function of $\mathcal{F}(H)$, which is FPT. ◀

3 PTime/NP-complete complexity dichotomies

In this section, we prove some results about the classical complexity of Vertex Deletion-H-Colouring, Edge Deletion-H-Colouring and Switching-H-Colouring. We first adapt a general method from [23] to show that Vertex Deletion-H-Colouring is either trivial, or NP-complete in Section 3.1.

For Edge Deletion-H-Colouring and Switching-H-Colouring, we cannot use this technique (in fact there exist nontrivial PTime cases). Thus, we turn our attention to edge-coloured graphs of order 2 (note that for every edge-coloured graph H of order at most 2, H-Colouring is in PTime [1, 4]). Recall that Switching-H-Colouring is defined only on 2-edge-coloured graphs, so our focus is on this case (but for Edge Deletion-H-Colouring our results hold for any number of colours). In Section 3.2, we prove a

dichotomy result for graphs of order at most 2 for the Edge Deletion-H-Colouring problem. The Switching-H-Colouring problem is treated in Section 3.3, where we also prove a dichotomy result.

The twelve 2-edge-coloured graphs of order at most 2 that are cores (up to symmetries of the colours) are depicted in Figure 1. The two colours are red (dashed edges) and blue (solid edges). We use the terminology of [1]: for $\alpha \in \{-, r, b, rb\}$, the 2-edge-coloured graph H^1_α is the graph of order 1 with no loop, a red loop, a blue loop, and both kinds of loops, respectively. Similarly, for $\alpha \in \{-, r, b, rb\}$ and $\beta, \gamma \in \{-, r, b\}$, the graph $H^{2\alpha}_{\beta,\gamma}$ denotes the graph of order 2 with vertex set $\{0, 1\}$. The string α indicates the presence of an edge between 0 and 1: no edge, a red edge, a blue edge and both edges for $-$, r, b and rb, respectively. Similarly, β and γ denote the presence of a loop at vertices 0 and 1, respectively ($-$ for no loop, r for a red loop, b for a blue loop).

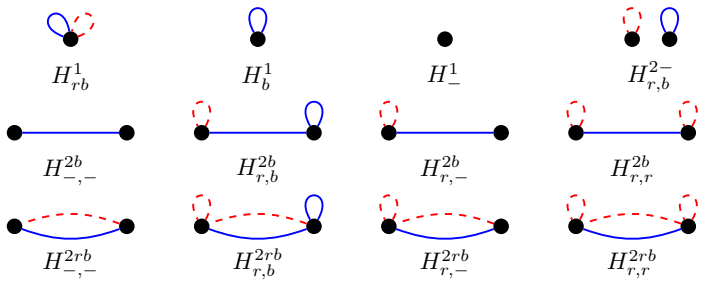


Figure 1 The twelve 2-edge-coloured cores of order at most 2 considered in this paper.

3.1 Dichotomy for Vertex Deletion-H-Colouring

Graph modification problems for operations VD and ED have been studied extensively. For a graph property $\mathcal{P}$, we denote by Vertex Deletion-$\mathcal{P}$ the graph modification problem for property $\mathcal{P}$ and operation VD. Lewis and Yannakakis [23] defined a non-trivial property $\mathcal{P}$ on graphs as a property true for infinitely many graphs and false for infinitely many graphs. They showed the following general result:

▶ **Theorem 5** (Lewis and Yannakakis [23]). *The* Vertex Deletion-$\mathcal{P}$ *problem for nontrivial graph-properties $\mathcal{P}$ that are hereditary on induced subgraphs is NP-hard.*

By modifying the proof of Theorem 5, we can prove the two following results (the proof is omitted due to space restrictions and is included in the full version of the manuscript).

▶ **Theorem 6.** *Let $\mathcal{P}$ be a nontrivial property of loopless edge-coloured graphs that is hereditary for induced subgraphs and true for all independent sets. Then,* Vertex Deletion-$\mathcal{P}$ *is NP-hard.*

For a t-edge-coloured graph, the only case where the property of mapping to H is trivial (in this case, always true) is when H has a vertex with all t kinds of loops attached (in which case the core of H is that vertex). Thus we obtain the following dichotomy.

▶ **Corollary 7.** *Let H be a t-edge-coloured graph.* Vertex Deletion-H-Colouring *is in PTime if H contains a vertex having all t kinds of loops, and NP-complete otherwise.*

3.2 Dichotomy for Edge Deletion-H-Colouring when H has order 2

No analogue of Theorem 5 for operation ED exists nor is expected to exist [28]. We thus restrict our attention to the case of edge-coloured graphs H of order at most 2. For this case we classify the complexity of Edge Deletion-H-Colouring. Since multiple edges of the same colour are irrelevant, if H has order 2, for each edge-colour there are three possible edges.

▶ **Theorem 8.** *Let H be an edge-coloured core of order at most 2. If each colour of H contains only loops or contains all three possible edges, then* Edge Deletion-H-Colouring *is in PTime; otherwise it is NP-complete.*

Proof. The NP-completeness proofs are by reductions from Vertex Cover, based on vertex- and edge-gadgets constructed using obstructions to the corresponding homomorphisms from [1]. They are available in the full version of the manuscript. We now present the PTime part.

First note that if colour i has all three possible edges in H, we can simply ignore this colour by removing it from H and G without decreasing the parameter, as it does not provide any constraint on the homomorphisms.

We can therefore suppose that H contains only loops. If two colours induce the same subgraph of H, then we can identify these two colours in both G and H as they give the same constraints.

If G has colours that H does not have, then remove each edge with this colour and decrease the parameter for each removed edge. If it goes below zero then we reject.

We can now assume that H has only loops and G has the same colours as H. We are left with only a few cases, as H is a core (there is no vertex whose set of loops is included in the set of loops of the other).

- H has a single loop. Then, $G \xrightarrow{ec} H$ as G has the same colours as H.
- H contains two non-incident loops with different colours and two non-incident loops of a third colour. Up to symmetry, suppose that H has one blue loop and one green loop on the first vertex and has one red loop and one green loop on the second vertex. We will reduce to the problem where we have removed the green loops. We construct G' from G by replacing each green edge by a blue edge and a red edge. We claim that Edge Deletion-H-Colouring with parameter k and input G is true if and only if Edge Deletion-$H^{2-}_{r,b}$-Colouring with parameter k plus the number of green edges of G on input G' is true. (See full version of the article for details.) Using this method we can reduce the problem to Edge Deletion-$H^{2-}_{r,b}$-Colouring, which is our last case.
- H contains two non-incident loops with different colours; then $H = H^{2-}_{r,b}$. Indeed if there were any other kind of loop, then we would be in the previous case or we could identify two colours. Note that a 2-edge-coloured graph maps to $H^{2-}_{r,b}$ if and only if it has no red edge incident to a blue edge. Thus, solving Edge Deletion-$H^{2-}_{r,b}$-Colouring amounts to splitting G into disconnecting red and blue connected components. This can be done by constructing the following bipartite graph: put a vertex for each edge of G; two are adjacent if the corresponding edges in G are adjacent and of different colours. Solving Edge Deletion-$H^{2-}_{r,b}$-Colouring is the same as solving Vertex Cover on this bipartite graph, which is PTime.

There is no other case as otherwise the set of loops of one vertex would be included in the set of loops of the other. ◀

3.3 Dichotomy for Switching-H-Colouring when H has order 2

▶ **Theorem 9.** *Let H be a 2-edge-coloured graph from Figure 1. If H is one of $H^{2b}_{r,b}$, $H^{2b}_{r,-}$, $H^{2rb}_{r,b}$, $H^{2rb}_{r,-}$ or $H^{2rb}_{r,r}$, then* Switching-H-Colouring *is NP-complete. Otherwise, it is in PTime.*

Proof. We begin with the PTime cases.

- Every 2-edge-coloured graph maps to H^1_{rb}, thus Switching-H^1_{rb}-Colouring is trivially in PTime.
- No graph with an edge can be mapped to H^1_- (regardless of switchings).
- For H^1_b, we need to test if the graph can be switched to an all-blue graph in less than k switchings. There are only two sets of switchings that achieve this signature (one is the complement of the other). It is in PTime to test if the graph can be switched to an all-blue graph (see [5, Proposition 2.1]). Doing that also gives us one of the two switching sets; we then need to check if its size is at most k or at least $|V(G)| - k$. So, Switching-H^1_b-Colouring is in PTime.
- For $H^{2-}_{r,b}$, we just apply the algorithm for H^1_b and H^1_r to each connected component, one of the two must accept for each of them.
- For $H^{2br}_{-,-}$, a graph G is a YES-instance if and only if G (without considering edge-colours) is bipartite, which is PTime testable.
- For $H^{2b}_{-,-}$ a graph G is a YES-instance if and only if it is bipartite and maps to H^1_b. We just need to check the two properties, which are both PTime.
- For $H^{2b}_{r,r}$, a graph G maps to $H^{2b}_{r,r}$ if and only if it has no cycles with an odd number of blue edges [1]. This property is preserved under the switching operation. Thus, switching the graph does not impact the nature of the instance. It is thus in PTime (we can test with $k = 0$) since $H^{2b}_{r,r}$-Colouring is in PTime [1, 4].

We now consider the NP-complete cases. For every H, Switching-H-Colouring clearly lies in NP. NP-hardness follows from the above-stated Theorem 3 (proved in [5, 6]) in all but one case: indeed, the switching cores of $H^{2b}_{r,b}$, $H^{2rb}_{r,b}$, $H^{2rb}_{r,-}$ and $H^{2rb}_{r,r}$ have at least three edges, and thus when H is one of these, Switching-H-Colouring is NP-complete (even with $k = |V(G)|$).

The last case is $H^{2b}_{r,-}$. We give a reduction from Vertex Cover to Switching-$H^{2b}_{r,-}$-Colouring. Given instance G of Vertex Cover, we construct an all-red copy G' of G, and we attach to each vertex v of G a blue edge vv', with a red loop on v' (see Figure 2).

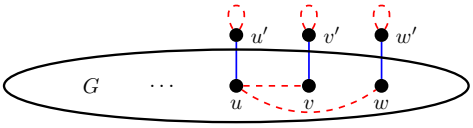


Figure 2 Reduction from Vertex Cover to Switching-$H^{2b}_{r,-}$-Colouring.

Denote by x the vertex of $H^{2b}_{r,-}$ with a loop, and by y the other one. Assume that G has a vertex cover C of size at most k. Denote by G'' the graph obtained from G' by switching at the vertices of C. We map every vertex v' to x, every vertex of C to x and the remaining ones to y. Since C is a vertex cover, each red edge of G'' is either a loop on some vertex v', an edge vv' with $v \in C$ or an edge uv with $u, v \in C$. In each case, both endpoints are mapped on x. The blue edges of G'' are then either vv' with $v \notin C$ or uv with $u \in C$ and $v \notin C$. In both cases, the two endpoints are mapped to different vertices of $H^{2b}_{r,-}$; thus, $G'' \xrightarrow{ec} H^{2b}_{r,-}$.

Conversely, assume that we can switch G' at vertices from a set S such that the resulting graph G'' maps to $H^{2b}_{r,-}$. Let C be the set of vertices v of G such that v or v' lies in S. Note that C has size at most $|S|$. We claim that C is a vertex cover of G. Assume that there is an edge uv in G with $u, v \notin C$. By construction, $u, u', v, v' \notin S$, so uu', vv' are blue in G'', and uv is red. Thus, u, v have to be mapped to x, and u', v' to y, a contradiction since u' has a incident red loop in G''. Therefore C is a vertex cover of G. ◀

4 Parameterized complexity results

4.1 Vertex Deletion-H-Colouring and Edge Deletion-H-Colouring

For many edge-coloured graphs H of order at most 2, we can show that Vertex Deletion-H-Colouring and Edge Deletion-H-Colouring are FPT by giving ad-hoc reductions to Vertex Cover, Odd Cycle Transversal or a combination of both. However, a more powerful method is to generalise a technique from [4] used to prove that H-Colouring is in PTime by reduction to 2-Sat (see also [2]):

▶ **Theorem 10** (Brewster et al. [4]). *Let H be an edge-coloured graph of order at most 2. Then, for each instance G of H-Colouring, there exists a PTime computable 2-Sat formula $F(G)$ that is satisfiable if and only if $G \xrightarrow{ec} H$. Thus, H-Colouring is in PTime.*

The formula $F(G)$ from Theorem 10 contains a variable x_v for each vertex v of G, and for each edge uv, a set of clauses that depends on H. The idea is to see the two vertices of H as "true" and "false", and for each edge uv of a certain colour, to express the possible assignments of x_u and x_v based on the edges of that colour that are present in H.

We will show how to generalise this idea to Vertex Deletion-H-Colouring and Edge Deletion-H-Colouring. We will need the following parameterized variant of 2-Sat:

Variable Deletion Almost 2-Sat	**Parameter:** k.
Input: A 2-CNF Boolean formula F, an integer k.	
Question: Is there a set of k variables that can be deleted from F (together with the clauses containing them) so that the resulting formula is satisfiable?	

Variable Deletion Almost 2-Sat and another similar variant, Clause Deletion Almost 2-Sat (where instead of k variables, k clauses may be deleted), are known to be FPT (see [11, Chapter 3.4]). We need to introduce a more general variant, that we call Group Deletion Almost 2-Sat, defined as follows.

Group Deletion Almost 2-Sat	**Parameter:** k.
Input: A 2-CNF Boolean formula F, an integer k, and a partition of the clauses of F into groups such that each group has a variable which is present in all of its clauses.	
Question: Is there a set of k groups of clauses that can be deleted from F so that the resulting formula is satisfiable?	

By a generalisation of [11, Exercise 3.21] for Clause Deletion Almost 2-Sat, we obtain the following complexity result for Group Deletion Almost 2-Sat. Its proof is included in the full version of the paper.

▶ **Proposition 11.** Group Deletion Almost 2-Sat *is FPT.*

We are now able to prove the following theorem.

▶ **Theorem 12.** *For every edge-coloured graph H of order at most 2,* Vertex Deletion-H-Colouring *and* Edge Deletion-H-Colouring *are FPT.*

Proof. For an instance G, k of Vertex Deletion-H-Colouring or Edge Deletion-H-Colouring, we consider the formula $F(G)$ from Theorem 10 (see [4]). In $F(G)$, to each vertex of G corresponds a variable x_v. Deleting v from G when mapping G to H has the same effect as deleting x_v when satisfying $F(G)$. Thus, this is an FPT reduction from Vertex Deletion-H-Colouring to Variable Deletion Almost 2-Sat.

Moreover, each edge uv of G corresponds to one or two clauses of $F(G)$. This naturally defines the groups of Group Deletion Almost 2-Sat by grouping the clauses corresponding to the same edge. Removing an edge is equivalent to remove its corresponding group. To finish, we have to make sure that we can have one variable common to all the clauses of each group. This is the case in the reduction in [4] for every case except when $E_i(H)$ (the set of edges of colour i in H) is just a loop. Assume without loss of generality that the loop is on vertex 1 (the other loop can be treated the same way). Suppose uv has colour i in G; then uv must be mapped to the loop on vertex 1. The original reduction added the clauses $(x_u)(x_v)$; we modify this part and add instead the clauses $(c + x_u)(c + x_v)(\bar{c})$ where c is a new variable. This is now a valid and equivalent instance of Group Deletion Almost 2-Sat, which is FPT by Proposition 11. ◀

4.2 Switching-H-Colouring: FPT cases

We now consider the parameterized complexity of Switching-H-Colouring. By Theorem 9, there are five 2-edge-coloured graphs H of order at most 2 with Switching-H-Colouring NP-complete. We first show that two of them are FPT:

▶ **Theorem 13.** Switching-$H^{2b}_{r,b}$-Colouring *and* Switching-$H^{2b}_{r,-}$-Colouring *are FPT.*

Proof. The graph $H^{2b}_{r,b}$ has the finite duality property by [1], which implies FPT time for Switching-$H^{2b}_{r,b}$-Colouring by a simple bounded search tree algorithm (Proposition 4).

For the graph $H^{2b}_{r,-}$, the duality set $\mathcal{F}(H)$ discovered in [1] is composed of paths of the form $RB^{2p-1}R$ (where R is a red edge, B a blue edge and $p \geq 1$ is an integer) and of cycles with an odd number of blue edges. As seen before, if the graph G has such a cycle then switching will not remove it, thus we can reject.

If the graph has a $RB^{2p-1}R$ path and is a positive instance, then we claim that we need to switch one of the four vertices of the red edges. Indeed, if we switch only at the vertices inside the blue path (those not incident with one of the red edges) then the parity of the number of blue edges will not change and we will still have some maximal odd blue subpath, the two edges next to the extremities being red. Thus we would still have a $RB^{2q-1}R$ path.

Thus, since we need to switch at one of these four vertices, we branch on this configuration using the classic bounded search tree technique. This is an FPT algorithm. ◀

4.3 Switching-H-Colouring: W[1]-hard cases

The remaining cases, $H^{2rb}_{r,b}$, $H^{2rb}_{r,-}$ and $H^{2rb}_{r,r}$, yield W[1]-hard Switching-H-Colouring problems, even for input graphs of large girth (recall that the *girth* of a graph is the smallest length of one of its cycles, and by the girth of an edge-coloured graph we mean the girth of its underlying uncoloured graph):

▶ **Theorem 14.** *Let $x \in \{r, b, -\}$. Then for any integer $q \geq 3$, the problem* Switching-$H^{2br}_{r,x}$-Colouring *is W[1]-hard, even for graphs G' with girth at least q. Under the same conditions,* Switching-$H^{2br}_{r,x}$-Colouring *cannot be solved in time $f(k)|G|^{o(k)}$ for any function f, assuming the ETH.*

We will prove Theorem 14 by three reductions from Multicoloured Independent Set, which is W[1]-complete [15]:

Multicoloured Independent Set	**Parameter:** k.
Input: A graph G, an integer k and a partition of $V(G)$ into k sets $V_1,\dots,V_k$.	
Question: Is there a set S of exactly k vertices of G, such that each V_i contains exactly one element of S, that forms an independent set of G?	

Our three reductions (one for each possible choice of x) follow the same pattern. In Section 4.3.1, we describe this idea, together with the required properties of the gadgets. In Sections 4.3.2, 4.3.3 and 4.3.4, we show how to construct the gadgets. Since the reduction preserves the parameter and is actually polynomial, the ETH-based lower bound follows.

4.3.1 Generic reduction

Let (G,k) be an instance of Multicoloured Independent Set, and denote by $V_1,\dots,V_k$ the partition of G. We begin by replacing each V_i by a *partition gadget* G_i. This gadget must have $|V_i|$ special vertices, in order to associate a vertex of G_i to each vertex of V_i. Moreover, G_i must satisfy the following:

(P1) We do not have $G_i \xrightarrow{ec} H^{2rb}_{r,x}$.

(P2) If we switch G_i at exactly one vertex v, then the obtained graph maps to $H^{2rb}_{r,x}$ (without switching) if and only if v is one of the special vertices of G_i.

(P3) G_i has girth at least q.

Let uv be an edge of G. Recall that u and v can be seen as vertices of G'. We then add an *edge gadget* G_{uv} between u and v. This gadget must satisfy the following:

(E1) Let H be the graph obtained from G_{uv} by switching at a subset S of $\{u,v\}$. Then, $H \xrightarrow{ec} H^{2rb}_{r,x}$ if $S \neq \{u,v\}$.

(E2) Assume that $u \in V_i$ and $v \in V_j$ and let H be the graph obtained from $G_{uv} \cup G_i \cup G_j$ by switching u and v. Then, we do *not* have $H \xrightarrow{ec} H^{2rb}_{r,x}$.

(E3) G_e has girth at least q.

(E4) In G_e, u and v are at distance at least q.

Let G' be the graph obtained from G by replacing each V_i by a partition gadget G_i, and each edge uv by an edge gadget G_{uv} such that for every $u \in V_i$ and v such that uv is an edge, we identify the special vertex u in G_i with the special vertex u in G_{uv}. (Note in particular that every vertex of G is present in G'.)

We say that a set S of vertices of G is *valid* if, when seen in G', it contains at most one special vertex in each edge gadget. We need a last condition about G':

(SP) If, after switching a valid set in G', the obtained graph does not map to $H^{2rb}_{r,x}$, then this is because a partition gadget or an edge gadget does not map to $H^{2rb}_{r,x}$ (that is, each minimal obstruction is entirely contained in an edge gadget or a partition gadget).

With this Property (SP), we can prove that $(G,k) \mapsto (G',k)$ is a valid reduction.

▶ **Proposition 15.** *(G',k) is a positive instance of* Switching-$H^{2rb}_{r,x}$-Colouring *if and only if (G,k) is a positive instance of* Multicoloured Independent Set.

Proof. Assume we can switch at most k vertices of G' such that the obtained graph maps to $H^{2rb}_{r,x}$. Let S be the set of those vertices. We claim that S is a valid set of G'. First note that, due to $(P1)$, S must contain at least one vertex in each V_i. This enforces $|S| = k$, thus S contains exactly one vertex v_i in each V_i. By $(P2)$, each of these v_i has to be one of the special vertices of G_i. This means that S contains only vertices that are present in G.

We claim that S induces an independent set in G. Assume by contradiction that there is an edge uv in G with $u, v \in S$. Then, by construction, there is an edge gadget whose special vertices are u and v, such that the edge gadget and the two partition gadgets associated with u and v map to $H^{2rb}_{r,x}$ when we switch only at u and v, contradicting $(E2)$. (Note that S does not contain any other vertex of the edge gadget nor any other vertex of the partition gadgets.) Therefore, G has an independent set of size k containing exactly one vertex in each set V_i.

Conversely, assume that G has an independent set S intersecting each V_i at one vertex. Then, we denote by H the graph obtained by switching all vertices of S in G'. By construction, this is a valid set, hence by (SP) every obstruction for mapping to $H^{2rb}_{r,x}$ in H is actually contained in some gadget. However, it cannot be contained in a partition gadget due to $(P2)$, nor in an edge gadget due to $(E1)$. Therefore, we have $H \xrightarrow{ec} H^{2rb}_{r,x}$. ◀

Observe moreover that, due to $(P3)$, $(E3)$ and $(E4)$, G' has girth at least q. Thus to prove Theorem 14 it suffices to construct the gadgets.

4.3.2 Gadgets for $H^{2rb}_{r,r}$

(a) Partition gadget for $V_i = \{x_0, x_1, x_2, x_3\}$. **(b)** Edge gadget for uv.

Figure 3 Partition and edge gadgets in the $H^{2rb}_{r,r}$-reduction when $q = 3$.

We now describe the gadgets for Switching-$H^{2rb}_{r,r}$-Colouring. Note that for every graph G, we have $G \xrightarrow{ec} H^{2rb}_{r,r}$ if and only if it does not contain an all-blue odd cycle.

The partition gadget G_i is an all-blue cycle of length $2q$ if q and $|V_i|$ have the same parity (resp. $2q + 2$ if they do not have the same parity) with a chord of order $|V_i|$ between two antipodal vertices. The special vertices are those on the chord (see Figure 3a).

Property $(P3)$ directly follows from the construction. Moreover, since G_i contains an all-blue odd cycle, we have $(P1)$. If we switch G_i at exactly one vertex, then either this vertex is a special vertex and the obtained graph does not have any all-blue odd cycle (and thus maps to $H^{2rb}_{r,r}$), or it is not a special vertex and there is still an all-blue odd cycle. Therefore, property $(P2)$ also holds.

We now consider the edge gadget. It is formed by an all-blue odd cycle of length $2q + 1$ where two vertices u, v at distance q have been switched (see Figure 3b). These vertices are the special vertices of the gadget. By construction, properties $(E3)$ and $(E4)$ hold. Moreover, consider a set $S \subset \{u, v\}$. The only way for switching the vertices of S to yield a graph containing an all-blue odd cycle is to switch both u and v. This proves $(E1)$. If we switch at both special vertices then we do not have $G_{uv} \xrightarrow{ec} H^{2rb}_{r,r}$, which implies $(E2)$.

It remains to prove Property (SP). Let S be a valid set, and let H be the graph obtained from G' when switching all vertices of S. Assume that H contains an all-blue odd cycle. Since S is valid set, at most one vertex has been switched in each edge gadget. Therefore, no all-blue odd cycle of H can contain an edge from an edge gadget. It is thus contained in some partition gadget, ensuring that (SP) holds.

Figure 4 The edge gadget for uv in the $H^{2rb}_{r,-}$-reduction when $q = 6$.

4.3.3 Gadgets for $H^{2rb}_{r,-}$

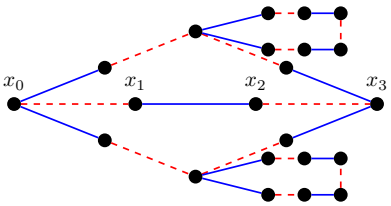
We now describe the gadgets for Switching-$H^{2rb}_{r,-}$-Colouring. Note that for every graph G, we have $G \xrightarrow{ec} H^{2rb}_{r,-}$ if and only if it does not contain a *bad* walk, i.e. a walk v_0, v_1, …, v_{2j}, v_0, v_{2j+2}, …, v_{2p-1}, v_0 such that all edges $v_{2i}v_{2i+1}$ are blue [1].

The partition gadget G_i is the same as in the previous case (see Figure 3a).

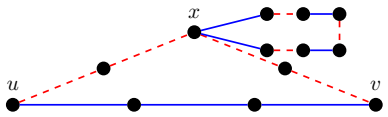
The edge gadget is an odd path of length at least q, whose edges are all blue except for the two first and two last ones (see Figure 4).

The proofs of validity for this case can be found in the full version of the paper.

4.3.4 Gadgets for $H^{2rb}_{r,b}$



(a) Partition gadget for $V_i = \{x_0, x_1, x_2, x_3\}$.



(b) Edge gadget for uv. The vertex x is where the two alternating cycles were identified.

Figure 5 Partition and edge gadgets in the $H^{2rb}_{r,b}$-reduction when $q = 3$.

We now describe the gadgets for Switching-$H^{2rb}_{r,b}$-Colouring. Note that for every graph G, we have $G \xrightarrow{ec} H^{2rb}_{r,b}$ if and only if it does not contain another type of *bad* walks, i.e. an alternating walk v_0, v_1, …, v_{2j}, v_0, v_{2j+2}, …, v_{2p-1}, v_0 for some integers j and p [1].

The partition gadget G_i is defined by gluing two obstructions with large girth along a path of length $|V_i|$ (see Figure 5a). More precisely, consider an alternating odd cycle C of size $|V_i| + q$ (or $|V_i| + q + 1$). Note that C contains a vertex u adjacent to two red edges. We attach an alternating odd cycle C' of length q (or $q + 1$) to u, such that the edges of C' adjacent to u are blue. To obtain G_i, we take two copies of this obstruction, and glue their respective largest cycle along a path of length $|V_i|$. The vertices of this path are the special vertices of G_i.

The edge gadget is formed by identifying the vertices with monochromatic neighbourhood of two alternating odd cycles of length $2q + 1$, in such a way that the common vertex has two blue edges in one cycle and two red edges in the other one. To obtain the edge gadget, we switch this graph at two vertices u, v in the same cycle, at distance q from each other (see Figure 5b).

The proofs of validity for this case can be found in the full version of the paper.

5 Conclusion and perspectives

We have introduced Vertex Deletion-H-Colouring, Edge Deletion-H-Colouring and Switching-H-Colouring and characterised their complexity for some small H. The full complexity landscape still needs to be determined. We have fully classified the classic

complexity of Vertex Deletion-H-Colouring problems. It remains to do the same for Edge Deletion-H-Colouring and Switching-H-Colouring.

We proved that both Vertex Deletion-H-Colouring and Edge Deletion-H-Colouring are FPT when H has order at most 2. However, if H has order 3, for example if H is a monochromatic triangle, we obtain 3-Colouring, which is not in XP. Switching-H-Colouring seems particularly interesting, since we obtained an FPT/W[1]-hard dichotomy when H has order at most 2 (in which case the problem is always in XP). But again for some H of order 3, Switching-H-Colouring is not in XP. It would be very interesting to obtain FPT/W[1]/XP trichotomies for Vertex Deletion-H-Colouring, Edge Deletion-H-Colouring and Switching-H-Colouring.

Finally, we note that it could be interesting to study analogues of Vertex Deletion-H-Colouring and Edge Deletion-H-Colouring for arbitrary fixed-template CSP problems. Up to our knowledge this has not been done.

References

1 Z. Bawar, R. C. Brewster, and D. A. Marcotte. Homomorphism duality in edge-coloured graphs. *Annales des sciences mathématiques du Québec*, 29(1):21–34, 2005.

2 R. C. Brewster. *Vertex colourings of edge-coloured graphs*. PhD thesis, Simon Fraser University, Canada, 1993.

3 R. C. Brewster. The complexity of colouring symmetric relational systems. *Discrete Applied Mathematics*, 49(1):95–105, 1994.

4 R. C. Brewster, R. Dedić, F. Huard, and J. Queen. The recognition of bound quivers using edge-coloured homomorphisms. *Discrete Mathematics*, 297:13–25, 2005.

5 R. C. Brewster, F. Foucaud, P. Hell, and R. Naserasr. The complexity of signed and edge-coloured graph homomorphisms. *Discrete Mathematics*, 340(2):223–235, 2017.

6 R. C. Brewster and M. H. Siggers. A complexity dichotomy for signed H-colouring. *Discrete Mathematics*, 341(10):2768–2773, 2018.

7 A. A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *IEEE Computer Society*, pages 319–330, FOCS 2017, 2017. Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science.

8 J. Bulín. On the complexity of H-coloring for special oriented trees. *European Journal of Combinatorics*, 69:54–75, 2018.

9 L. Cai. Fixed parameter tractability of graph modification problem for hereditary properties. *Information Processing Letters*, 58:171–176, 1996.

10 L. Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127:415–429, 2003.

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

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

13 A. Ehrenfeucht, J. Hage, T. Harju, and G. Rozenberg. Complexity issues in switching of graphs. In *Lecture Notes in Computer Science*, pages 59–70, Proceedings of the International Workshop on Theory and Application of Graph Transformations, TAGT'98, 1764, 2000.

14 T. Feder and M. Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: a study through datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1998.

15 M. R. Fellows, D. Hermelin, F. Rosamond, and S. Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 40(1):53–61, 2009.

16 F. Foucaud and R. Naserasr. The complexity of homomorphisms of signed graphs and signed constraint satisfaction. In *Lecture Notes in Computer Science*, pages 526–537, Proceedings of the 11th Latin American Symposium on Theoretical Informatics 2014, LATIN'14. 8392, 2014.

17 F. Harary. On the notion of balance of a signed graph. *Michigan Mathematical Journal*, 2(2):143–146, 1953-1954.

18 P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory Series B*, 48(1):92–110, 1990.

19 F. Hüffner, N. Betzler, and R. Niedermeier. Separator-based data reduction for signed graph balancing. *Journal of Combinatorial Optimization*, 20(4):335–360, 2010.

20 L. Jaffke and B. M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In *Lecture Notes in Computer Science*, pages 345–356, Proceedings of the 10th International Conference on Algorithms and Complexity, CIAC'17. 10236, 2017.

21 E. Jelínková, O. Suchý, P. Hliněný, and J. Kratochvíl. Parameterized problems related to Seidel's switching. *Discrete Mathematics and Theoretical Computer Science*, 13(2):19–42, 2011.

22 S. Khot and V. Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002.

23 J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

24 D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–71, 2011.

25 D. Marx. Parameterized coloring problems on chordal graphs. *Theoretical Computer Science*, 351(3):407–424, 2006.

26 R. Naserasr, E. Rollová, and É. Sopena. Homomorphisms of signed graphs. *Journal of Graph Theory*, 79(3):178–212, 2015.

27 Y. Takenaga and K. Higashide. Vertex coloring of comparability $+ke$ and $-ke$ graphs. In *Lecture Notes in Computer Science*, pages 102–112, Proceedings of the 32nd International Worksop on Graph-Theoretic Concepts in Computer Science, WG'06. 4271, 2006.

28 M. Yannakakis. Edge-deletion problems. *SIAM Journal on Computing*, 10(2):297–309, 1981.

29 T. Zaslavsky. Signed graphs. *Discrete Applied Mathematics*, 4(1):47–74, 1982.

30 D. Zhuk. A proof of CSP dichotomy conjecture. In *IEEE Computer Society*, pages 331–342, FOCS 2017, 2017. Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science.

On the Fine-Grained Complexity of Least Weight Subsequence in Multitrees and Bounded Treewidth DAGs

Jiawei Gao[1]
University of California, San Diego, CA, USA
jiawei@cs.ucsd.edu

Abstract

This paper introduces a new technique that generalizes previously known fine-grained reductions from linear structures to graphs. Least Weight Subsequence (LWS) [30] is a class of highly sequential optimization problems with form $F(j) = \min_{i<j}[F(i) + c_{i,j}]$. They can be solved in quadratic time using dynamic programming, but it is not known whether these problems can be solved faster than $n^{2-o(1)}$ time. Surprisingly, each such problem is subquadratic time reducible to a highly parallel, non-dynamic programming problem [36]. In other words, if a "static" problem is faster than quadratic time, so is an LWS problem. For many instances of LWS, the sequential versions are equivalent to their static versions by subquadratic time reductions. The previous result applies to LWS on linear structures, and this paper extends this result to LWS on paths in sparse graphs, the Least Weight Subpath (LWSP) problems. When the graph is a multitree (i.e. a DAG where any pair of vertices can have at most one path) or when the graph is a DAG whose underlying undirected graph has constant treewidth, we show that LWSP on this graph is still subquadratically reducible to their corresponding static problems. For many instances, the graph versions are still equivalent to their static versions.

Moreover, this paper shows that if we can decide a property of form $\exists x \exists y P(x, y)$ in subquadratic time, where P is a quickly checkable property on a pair of elements, then on these classes of graphs, we can also in subquadratic time decide whether there exists a pair x, y in the transitive closure of the graph that also satisfy $P(x, y)$.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases fine-grained complexity, dynamic programming, graph reachability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.16

Related Version A full version of the paper is available on ECCC : `https://eccc.weizmann.ac.il/report/2019/045/`.

Funding Work supported by a Simons Investigator Award from the Simons Foundation.

Acknowledgements I sincerely thank Russell Impagliazzo for his guidance and advice on this paper. I would like to thank Marco Carmosino and Jessica Sorrell for helpful comments. Also I would like to thank the anonymous reviewers for comments on an earlier version of this paper.

1 Introduction

1.1 Extending one-dimensional dynamic programming to graphs

Least Weight Subsequence (LWS) [30] is a type of dynamic programming problems: select a set of elements from a linearly ordered set so that the total cost incurred by the adjacent pairs of selected elements is optimized. It is defined as follows: Given elements $x_0, \ldots, x_n$,

[1] Now at Google.

and an $n \times n$ matrix C of costs $c_{i,j}$ for all pairs of indices $i < j$, compute F on all elements, defined by

$$F(j) = \begin{cases} 0, \text{ for } j = 1 \\ \min_{0 \le i < j}[F(i) + c_{i,j}], \text{ for } j = 2, \ldots, n \end{cases}$$

$F(j)$ is the optimal cost value from the first element up to the j-th element. We use the notation LWS_C to define the LWS problem with cost matrix C. The Airplane Refueling problem [30] is a well known example of LWS: Given the locations of airports on a line, find a subset of the airports for an airplane to add fuel, that minimizes the total cost. The cost of flying from the i-th to the j-th airport without stopping is defined by $c_{i,j}$. Other LWS examples include finding a longest chain satisfying a certain property, such as Longest Increasing Subsequence [25] and Longest Subset Chain [36]; breaking a linear structure into blocks, such as Pretty Printing [34]; variations of Subset Sum such as special versions of the Coin Change problem and the Knapsack problem[36]. These problems have $O(n^2)$ time algorithms using dynamic programming, and in many special cases it can be improved: when the cost satisfies the quadrangle inequality or some other properties, there are near linear time algorithms [50, 46, 26]. But for the general LWS, it is not known whether these problems can be solved faster than $n^{2-o(1)}$ time.

A general approach to understanding the fine-grained complexity of these problems was initiated in [36]. Many LWS problems have succinct representations of $c_{i,j}$. Usually C is defined implicitly by the data associated to each element, and the size of the data on each element is relatively small compared to n. Taking problems defined in [36] as examples, in $\mathsf{LowRankLWS}$, $c_{i,j} = \langle \mu_i, \sigma_j \rangle$, where μ_i and σ_j are boolean vectors of length $d \ll n$ associated to each element that are given by the input. The $\mathsf{ChainLWS}$ problem has costs $c_1, \ldots, c_n$ defined by a boolean relation P so that $c_{i,j}$ equals c_j if $P(i,j)$ is true, and ∞ otherwise. P is computable by data associated to element i and element j. (For example, in $\mathsf{LongestSubsetChain}$, $P(i,j)$ is true iff set S_i is contained in set S_j, where S_i and S_j are sets associated to elements i and j respectively.) So the goal of the problem becomes finding a longest chain of elements so that adjacent elements that are to be selected satisfy property P. When C can be represented succinctly, we can ask whether there exist subquadratic time algorithms for these problems, or try to find subquadratic time reductions between problems. [36] showed that in many LWS_C problems where C can be succinctly described in the input, the problem is subquadratic time reducible to a corresponding problem, which is called a $\mathsf{StaticLWS}_C$ problem. The problem $\mathsf{StaticLWS}_C$ is: given elements $x_1, \ldots, x_n$, a cost matrix C, and values $F(i)$ on all $i \in \{1, \ldots, n/2\}$, compute $F(j) = \min_{i \in \{n/2+1, \ldots, n\}}[F(i) + c_{i,j}]$ for all $j \in \{n+1, \ldots, 2n\}$. It is a parallel, batch version (with many values of j rather than a single one) of the LWS update rule applied sequentially one index at a time in the standard DP algorithm. The reduction from LWS_C to $\mathsf{StaticLWS}_C$ implies that a highly sequential problem can be reducible to a highly parallel one. If a $\mathsf{StaticLWS}_C$ problem can be solved faster than quadratic time, so can the corresponding LWS_C problem. Apart from one-directional reductions from general LWS_C to $\mathsf{StaticLWS}_C$, [36] also proved subquadratic time equivalence between some concrete problems ($\mathsf{LowRankLWS}$ is equivalent to $\mathsf{MinInnerProduct}$, $\mathsf{NestedBoxes}$ is equivalent to $\mathsf{VectorDomination}$, $\mathsf{LongestSubsetChain}$ is equivalent to $\mathsf{OrthogonalVectors}$, and $\mathsf{ChainLWS}$, which is a generalization of $\mathsf{NestedBoxes}$ and $\mathsf{LongestSubsetChain}$, is equivalent to $\mathsf{Selection}$, a generalization of $\mathsf{VectorDomination}$ and $\mathsf{OrthogonalVectors}$).

Some of the LWS problems can be naturally extended from lines to graphs. For example, on a road map, we wish to find a path for a vehicle, along which we wish to find a sequence of cities where the vehicle can rest and add fuel so that the total cost is minimized. The cost

of traveling between cities x and y without stopping is defined by cost $c_{x,y}$. Connections between cities could be a general graph, not just a line. Works about algorithms for special LWS problems on special classes of graphs include [11, 43, 24, 38].

Using a similar approach as [36], this paper extends the Least Weight Subsequence problems to the Least Weight Subpath (LWSP_C) problem whose objective is to find a least weight subsequence on a path of a given DAG $G = (V, E)$. Let there be a set V_0 containing vertices that can be the starting point of a subsequence in a path. The optimum value on each vertex is defined by:

$$F(v) = \begin{cases} \min(0, \min_{u \rightsquigarrow v}[F(u) + c_{u,v}]), & \text{for } v \in V_0 \\ \min_{u \rightsquigarrow v}[F(u) + c_{u,v}], & \text{for } v \notin v_0 \end{cases}$$

where $u \rightsquigarrow v$ means v is reachable from u. The goal of LWSP_C is to compute $F(v)$ for all vertices $v \in V$. Examples of LWSP_C problems will be given in Appendix B. LWSP_C can be solved in time $O(|V| \cdot |E|)$ by doing reversed depth/breadth first search from each vertex, and update the F value on the vertex accordingly. It is not known whether it has faster algorithms, even for Longest Increasing Subsequence, which is an LWS_C instance solvable in $O(n \log n)$ time on linear structures. If C is succinctly describable in similar ways as LowRankLWS, NestedBoxes, SubsetChain or ChainLWS, we wish to study if there are subquadratic time algorithms or subquadratic time reductions between problems.

For the cost matrix C, we consider that every vertex has some additional data so that $c_{x,y}$ can be computed by the data contained in x and y. Let the size of additional data associated to each vertex v be its weighted size $w(v)$. The weight of a vertex can be defined in different ways according to the problems. For example, in LowRankLWS, the weighted size of an element can be defined as the dimension of its associated vector; and in SubsetChain, the weighted size of an element is the size of its corresponding subset. We use $m = |E|$ as the number of graph edges. Let n be the number of vertices. We study the case where the graph is sparse, i.e. $m = n^{1+o(1)}$. Let the total weighted size of all vertices be N. For LWS_C and other problems without graphs, we use N as the input size. For LWSP_C and other problems on graphs, we use $M = \max(m, N)$ as the size of the input.

In this paper we will see that if we can improve the algorithm for $\mathsf{StaticLWS}_C$ to $N^{2-o(1)}$, then on some classes of graphs we can solve LWSP_C faster than $M^{2-o(1)}$ time.

1.2 Fine-grained complexity preliminaries

Fine-grained complexity studies the exact-time reductions between problems, and the completeness of problems in classes under exact-time reductions. These reductions have established conditional lower bounds for many interesting problems. The Orthogonal Vectors problem (OV) is a well-studied problem solvable in quadratic time. If the *Strong Exponential Time Hypothesis (SETH)* [31, 32] is true, then OV does not have truly subquadratic time algorithms [47]. The problem OV is defined as follows: Given n boolean vectors of dimension $d = \omega(\log n)$, and decide whether there is a pair of vectors whose inner product is zero. The best algorithm is in time $n^{2-\Omega(1/\log(d/\log n))}$ [7, 23]. The *Moderate-dimension OV conjecture (MDOVC)* states that for all $\epsilon > 0$, there are no $O(n^{2-\epsilon}\mathsf{poly}(d))$ time algorithms that solve OV with vector dimension d. If this conjecture is true, then many interesting problems would get lower bounds, including dynamic programming problems such as Longest Common Subsequence [2, 20], Edit Distance [14, 5], Fréchet distance [18, 21, 22], Local Alignment [9], CFG Parsing and RNA Folding [1], Regular Expression Matching [15, 19] , and also many graph problems [42, 8, 16]. There are also conditional hardness results about graph problems based on the hardness of All Pair Shortest Path [49, 4, 10, 39] and 3SUM [6, 35].

The *fine-grained reduction* was introduced in [49], which can preserve polynomial saving factors in the running time between problems. The statements for fine-grained complexity are usually like this: if there is some $\epsilon_2 > 0$ such that problem Π_2 of input size n is in $\mathsf{TIME}((T_2(n))^{1-\epsilon_2})$, then problem Π_1 of input size n is in $\mathsf{TIME}((T_1(n))^{1-\epsilon_1})$ for some ϵ_1. If T_1 and T_2 are both $O(n^2)$ then this reduction is called a subquadratic reduction. Furthermore, the *exact-complexity reduction* is a more strict version that can preserve sub-polynomial savings factors between problems. We use $(\Pi_1, T_1(n)) \leq_{\text{EC}} (\Pi_2, T_2(n))$ to denote that there is a reduction from problem Π_1 to problem Π_2 so that if problem Π_2 is in $\mathsf{TIME}(T_2(n))$, then problem Π_1 is in $\mathsf{TIME}(T_1(n))$.

1.3 Introducing reachability to first-order model checking

Similar to extending LWS_C to paths in graphs, introducing transitive closure to first-order logic also which makes parallel problems become sequential. The first-order property (or first-order model checking) problem is to decide whether an input structure satisfies a fixed first-order logic formula φ. Although model checking for input formulas is PSPACE-complete [44, 45], when φ is fixed by the problem, it is solvable in polynomial time. We consider the class of problems where each problem is the model checking for a fixed formula φ. The sparse version of OV [27] is one of these problems, defined by the formula $\exists u \exists v \forall i \in [d](\neg One(u,i) \vee (\neg One(v,i)))$, where relation $One(u,i)$ is true iff the i-th coordinate of vector u is one.

If φ has k quantifiers ($k \geq 2$), then on input structures of n elements and m tuples of relations, it can be solved in time $O(n^{k-2}m)$ [28]. On dense graphs where $k \geq 9$, it can be solved in time $O(n^{k-3+\omega})$, where ω is the matrix multiplication exponent [48]. Here we study the case where the input structure is sparse, i.e. $m = n^{1+o(1)}$, and ask whether a three-quantifier first-order formula can be model checked in time faster than $m^{2-o(1)}$. The *first-order property conjecture (FOPC)* states that there exists integer $k \geq 2$, so that first-order model checking for $(k+1)$-quantifier formulas cannot be solved in time $O(m^{k-\epsilon})$ for any $\epsilon > 0$. This conjecture is equivalent to MDOVC, since OV is proven to be a complete problem in the class of first-order model checking problems; in other words, any model checking problem of 3 quantifier formulas on sparse graphs is subquadratic time reducible to OV [28]. This means from improved algorithms for OV we can get improved algorithms for first-order model checking.

The first-order property problems are highly parallelizable. If we introduce the transitive closure (TC) operation on the relations, then these problems will become sequential. The transitive closure of a binary relation E can be considered as the reachability relation by edges of E in a graph. In a sparse structure, the TC of a relation may be dense. So it can be considered as a dense relation succinctly described in the input. In finite model theory, adding transitive closure significantly adds to the expressive power of first-order logic (First discovered by Fagin in 1974 according to [37], and then re-discovered by [12].) In fine-grained complexity, adding arbitrary transitive closure operations on the formulas strictly increases the hardness of the model checking problem. More precisely, [27] shows that SETH on constant depth circuits, which is a weaker conjecture than the SETH (which concerns k-CNF-SAT), implies the model checking for two-quantifier first-order formulas with transitive closure operations cannot be solved in time $O(m^{2-\epsilon})$ for any $\epsilon > 0$. This means this problem may stay hard even if the SETH on k-CNF-SAT is refuted.

However, we will see that for a class of three-quantifier formulas with transitive closure, model checking is no harder than OV under subquadratic time reductions.

We define problem $\mathsf{Selection}_P$ to be the decision problem for whether an input structure satisfies $(\exists x \in X)(\exists y \in Y)P(x,y)$. $P(x,y)$ is a fixed property specified by the problem that

can be decided in time $O(w(x) + w(y))$, where weighted size $w(x)$ is the size of additional data on element x. For example, OV is $\mathsf{Selection}_P$ where $P(x, y)$ iff x and y are a pair of orthogonal vectors. In this case $w(x)$ is defined as the length of vector x. (If we work on the sparse version of OV, the weighted size $w(x)$ is defined by the Hamming weight of x.)

On a directed graph $G = (V, E)$, we define Path_P to be the problem of deciding whether $(\exists x \in V)(\exists y \in V)[\mathrm{TC}_E(x, y) \wedge P(x, y)]$, where TC_E is the transitive closure of relation E and $P(x, y)$ is a property on x, y fixed by the problem. That is, whether there exist two vertices x,y not only satisfying property P but also y is reachable from x by edges in E. We will give an example of Path_P in Appendix B. Also, we define $\mathsf{ListPath}_P$ to be the problem of listing all $x \in V$ such that $(\exists y \in V)[\mathrm{TC}_E(x, y) \wedge P(x, y)]$.

Considering the model checking problems, we let $PathFO_3$ and $ListPathFO_3$ denote the class of Path_P and $\mathsf{ListPath}_P$ such that P is of form $\exists z\psi(x, y, z)$ or $\forall z\psi(x, y, z)$, where ψ is a quantifier-free formula in first-order logic. Later we will see that problems in $PathFO_3$ and $ListPathFO_3$ are no harder than OV. In these model checking problems, the weighted size of an element is the number of tuples in the input structure that the element is contained in.

Trivially, $\mathsf{Selection}_P$ on input size (N_1, N_2) can be decided in time $O(N_1N_2)$, where N_1 is the total weighted size of elements in X, and N_2 is the total weighted size of elements in Y. Path_P and $\mathsf{ListPath}_P$ on input size M and total vertex weighted size N are solvable time $O(MN)$ by depth/breadth first search from each vertex, where M is defined to be the maximum of N and the number of edges m. This paper will show that on some graphs, if $\mathsf{Selection}_P$ is in truly subquadratic time, so is Path_P and $\mathsf{ListPath}_P$. Interestingly, by applying the same reduction techniques from Path_P to $\mathsf{Selection}_P$, we can get a similar reduction from a dynamic programming problem on a graph to a static problem.

1.4 Main results

This paper works on two classes of graphs, both having some similarities to trees. The first class is where the graph G is a multitree. A *multitree* is a directed acyclic graph where the set of vertices reachable from any vertex form a tree. Or equivalently a DAG is a multitree if and only if on all pairs of vertices u, v, there is at most one path from u to v. In different contexts, multitrees are also called *strongly unambiguous graphs*, *mangroves* or *diamond-free posets* [29]. These graphs can be used to model computational paths in nondeterministic algorithms where there is at most one path connecting any two states [13]. The butterfly network, which is a widely-used model of the network topology in parallel computing, is an example of multitrees. We also work on multitrees of strongly connected component, which is a graph that when each strongly connected components are replaced by a single vertex, the graph becomes a multitree.

The second class of graphs is when we treat G as undirected by replacing all directed edges by undirected edges, the underlying graph has constant treewidth. *Treewidth* [40, 41] is an important parameter of graphs that describes how similar they are to trees. [2] On these classes of graphs, we have the following theorems.

▶ **Theorem 1** (Reductions between decision problems.). *Let $t(M) \geq 2^{\Omega(\sqrt{\log M})}$, and let the graph $G = (V, E)$ satisfy one of the following conditions:*

- *G is a multitree, or*
- *G is a multitree of strongly connected components, or*
- *The underlying undirected graph of G has constant treewidth,*

[2] Here we consider the undirected treewidth, where both the graph and the decomposition tree are undirected. It is different from *directed treewidth* defined for directed graphs by [33].

then, the following statements are true:

- *If* $\mathsf{Selection}_P$ *is in time* $N_1N_2/t(\min(N_1,N_2))$*, then* Path_P *is in time* $M^2/t(\mathsf{poly}M)$.[3]
- *If* Path_P *is in time* $M^2/t(M)$*, then* $\mathsf{ListPath}_P$ *is in time* $M^2/t(\mathsf{poly}M)$.
- *When* $P(x,y)$ *is of form* $\exists z\psi(x,y,z)$ *or* $\forall z\psi(x,y,z)$ *where* ψ *is a quantifier-free first-order formula,* $\mathsf{Selection}_P$ *is in time* $N_1N_2/t(\min(N_1,N_2))$ *iff* Path_P *is in time* $M^2/t(\mathsf{poly}M)$ *iff* $\mathsf{ListPath}_P$ *is in time* $M^2/t(\mathsf{poly}M)$.

This theorem implies that OV is hard for classes $PathFO_3$ and $ListPathFO_3$. By the improved algorithm for OV [7, 23], we get improved algorithms for $PathFO_3$ and $ListPathFO_3$:

▶ **Corollary 2** (Improved algorithms.). *Let the graph G be a multitree, or multitree of strongly connected components, or a DAG whose underlying undirected graph has constant treewidth. Then $PathFO_3$ and $ListPathFO_3$ are in time $M^2/2^{\Omega(\sqrt{\log M})}$.*

Next, we consider the dynamic programming problems. If the cost matrix C in LWSP_C is succinctly describable, we get the following reduction from LWSP_C to $\mathsf{StaticLWS}_C$.

▶ **Theorem 3** (Reductions between optimization problems.). *On a multitree graph, or a DAG whose underlying undirected graph has constant treewidth, let $t(N) \geq 2^{\Omega(\sqrt{\log N})}$, then,*

1. *if* $\mathsf{StaticLWS}_C$ *of input size N is in time $N^2/t(N)$, then* LWSP_C *on input size M is in time $M^2/t(\mathsf{poly}(M))$.*
2. *if* LWSP_C *is in time $M^2/t(M)$, then* LWS_C *is in time $N^2/t(\mathsf{poly}(N))$.*

If there is a reduction from a concrete $\mathsf{StaticLWS}_C$ problem to its corresponding LWS_C problem (e.g. there are reductions from MinInnerProduct to LowRankLWS, from VectorDomination to NestedBoxes and from OV to LongestSubsetChain [36]), then the corresponding LWS_C, $\mathsf{StaticLWS}_C$ and LWSP_C problems are subquadratic-time equivalent. From the algorithm for OV [23] and SparseOV [28], we get improved algorithm for problem LongestSubsetChain:

▶ **Corollary 4** (Improved algorithm). *On a multitree or a DAG whose underlying undirected graph has constant treewidth,* LongestSubsetChain *is in time $M^2/2^{\Omega(\sqrt{\log M})}$.*

The reduction uses a technique that decomposes multitrees into sub-structures where it is easy to decide whether vertices are reachable. So we also get reachability oracles using subquadratic space, that can answer reachability queries in sublinear time.

▶ **Theorem 5** (Reachability oracle). *On a multitree of strongly connected components, there exists a reachability oracle with subquadratic preprocessing time and space that has sublinear query time. On a multitree, the preprocessing time and space is $O(m^{5/3})$, and the query time is $O(m^{2/3})$.*

1.5 Organization

In Section 2 we prove the first part of Theorem 1, by reduction from Path_P to $\mathsf{Selection}_P$ on multitrees. The case for bounded treewidth DAGs will be presented in the full version. Section 3 proves Theorem 3 by presenting a reduction from LWSP_C to $\mathsf{StaticLWS}_C$, and the proof of correctness will be in the full version. Section 4 discusses about open problems.

[3] This reduction also applies to optimization versions of these two problems. Let Path_F be a problem to compute $\min_{x,y\in V, x\rightsquigarrow y} F(x,y)$ and $\mathsf{Selection}_F$ be a problem to compute $\min_{x\in X, y\in Y} F(x,y)$, where F is a function on x,y, instead of a boolean property. Then the same technique gives us a reduction from Path_F to $\mathsf{Selection}_F$.

Appendix A lists the definitions of problems, and Appendix B shows some concrete problems as examples.

Due to space restrictions, several proofs had to be deferred to the full version, including the rest of Theorem 1, the subquadratic equivalence of $\mathsf{Selection}_P$, Path_P and $\mathsf{ListPath}_P$ when P is a first-order property, and the reachability oracle for multitrees.

2 From sequential problems to parallel problems, on multitrees

We will prove the first part of Theorem 1 by showing that if $t(M) \geq 2^{\Omega(\sqrt{\log M})}$, then $(\mathsf{Path}_P, M^2/t(\mathsf{poly}M)) \leq_{\mathrm{EC}} (\mathsf{Selection}_P, N_1N_2/t(\min(N_1, N_2)))$. This section gives the reduction for multitrees and multitrees of strongly connected components. For constant treewidth graphs, the reduction will be shown in the full version.

2.1 The recursive algorithm

The algorithm uses a divide-and-conquer strategy. We will consider each strongly connected component as a single vertex, whose weighted size equals the total weighted size of the component. In the following algorithm, whenever querying $\mathsf{Selection}_P$ or exhaustively enumerating pairs of reachable vertices and testing P on them, we can extract all the vertices from a strongly connected component. Thus we will be working on a multitree, instead of a multitree of strongly connected components. Testing P on a pair of vertices (or strongly connected components) of total weighted sizes N_1, N_2 is in time $O(N_1N_2)$.

Let $\mathsf{CutPath}_P$ be a variation of Path_P. It is the property testing problem for $(\exists x \in S)(\exists y \in T)[TC_E(x, y) \wedge \varphi(x, y)]$, where (S, T) is a cut in the graph, such that all the edges between S and T are directed from S to T. $\mathsf{CutPath}_P$ on input size M and total vertex weighted size N can be solved in time $O(MN)$ if $P(x, y)$ is decidable in time $O(w(x) + w(y))$: start from each vertex and do depth/breadth first search, and on each pair of reachable vertices decide if P is satisfied.

▶ **Lemma 6.** *For $t(M) \geq 2^{\Omega(\sqrt{\log M})}$, if $\mathsf{Selection}_P(N, N)$ is in time $N^2/t(N)$ and $\mathsf{CutPath}_P(M)$ is in time $M^2/t(M)$, then $\mathsf{Path}_P(M)$ is in time $M^2/t(\mathsf{poly}(M))$.*

Proof. Let γ be a constant satisfying $0 < \gamma \leq 1/4$. Let $T_\Pi(M)$ be the running time of problem Π on a structure of total weighted size M. We show that there exists a constant c where $0 < c < 1$ so that if $T_{\mathsf{Path}_P}(M')$ is at most $M'^2/t(M'^c)$ for all $M' < M$, then $T_{\mathsf{Path}_P}(M) \leq M^2/t(M^c)$. We run the recursive algorithm as shown in Algorithm 1. The intuition is to divide the graph into a cut S, T, recursively compute Path_P on S and T, and deal with paths from S to T.

It would be good if the difference of total weighted sizes between S and T is at most M^γ. Otherwise, it means by the topological order, there is a vertex of weighted size at least M^γ in the middle, adding it to either S or T would make the size difference between S and T exceed M^γ. In this case, we use letter x to denote the vertex. We will deal with x separately. We temporarily set aside the time of recursively running $\mathsf{Selection}_P$ on x (when x is shrunk from a strongly connected component) in all the recursive calls, and consider the rest of the running time.

Algorithm 1 $\mathsf{Path}_P(G)$ on a DAG

```
// Reducing Path_P to Selection_P and CutPath_P
if G has only one vertex then return false.
Let M be the weighted size of the problem.
Topological sort all vertices.
Keep adding vertices to S by topological order, until the total weighted size of S
  exceeds M/2. Let the rest of vertices be T.
if |S| − |T| > M^γ then
    Let x be the last vertex added to S. Remove x from S.
Run Path_P on the subgraph induced by S.
Run CutPath_P(S, T).
if x exists then
    Run CutPath_P(S, x).
    If x is originally a strongly connected component, run Selection_P on it.
    Run CutPath_P(x, T)
Run Path_P on the subgraph induced by T.
if any one of the above three calls returns true then return true.
```

Let M_S and M_T be the sizes of sets S and T respectively. Without loss of generality, assume $M_S \geq M_T$, and let $\Delta = M_S - M_T$, which is at most M^γ. Then we have

$$\begin{aligned} T_{\mathsf{Path}_P}(M) &= T_{\mathsf{Path}_P}(M_S) + T_{\mathsf{Path}_P}(M_T) + 3T_{\mathsf{CutPath}_P}(M) + O(M) \\ &= T_{\mathsf{Path}_P}(M_T + \Delta) + T_{\mathsf{Path}_P}(M_T) + 3T_{\mathsf{CutPath}_P}(M) + O(M) \\ &\leq 2T_{\mathsf{Path}_P}(M/2 + \Delta) + 3T_{\mathsf{CutPath}_P}(M) + O(M) \\ &= 2(M/2+\Delta)^2/t((M/2+\Delta)^c) + 3M^2/t(M) + O(M). \end{aligned}$$

Because $t(M) < M$ and is monotonically growing, The term $3M^2/t(M) + O(M)$ is bounded by $4M^2/t(M) \leq 16(M/2)^2/t(M) \leq 16(M/2+\Delta)^2/t((M/2+\Delta)^c)$. Thus the above formula is bounded $18(M/2+\Delta)^2/t((M/2+\Delta)^c)$. By picking small enough constant γ and c, this sum is less than $M^2/t(M^c)$.

For the time of running $\mathsf{Selection}_P$ on x where x is originally a strongly connected component, we consider all recursive calls of Path_P. Let the size of each such x be M_i. The total time would be $\sum_i M_i^2/t(M_i) < (\sum_i M_i^2)/t(M^\gamma)$. Because $\sum_i M_i \leq M$, the sum is at most $M^2/t(M^\gamma)$, a value subquadratic to M, with M being the input size of the outermost call of Path_P. ◀

2.2 A special case that can be exhaustively searched

The following lemma shows that if no vertex has both a lot of ancestors and a lot of descendants, then the total number of reachable pairs of vertices is subquadratic to m. This lemma holds for any DAG, not just for multitrees. We will use this lemma in the next subsection to show that in a subgraph where all vertices have few ancestors and descendants, we can test property P on all pairs of reachable vertices by brute force. Actually, we will use a weighted version of this lemma, which will be proved in the full version.

▶ **Lemma 7.** *If in a DAG $G = (V, E)$ of m edges, every vertex has either at most n_1 ancestors or at most n_2 descendants, then there are at most $(m \cdot n_1 \cdot n_2)$ pairs of vertices s, t such that s can reach t.*

In a DAG $G = (V, E)$ of m edges, let S, T be two disjoint sets of vertices where edges between S and T only direct from S to T. If every vertex has either at most n_1 ancestors in S or at most n_2 descendants in T, then there are at most $(m \cdot n_1 \cdot n_2)$ pairs of vertices $s \in S$ and $t \in T$ such that s can reach t.

Proof. We define the ancestors of an edge $e \in E$ to be the ancestors (or ancestors in S) of its incoming vertex, and its descendants to be the descendants (or descendants in T) of its outgoing vertex. Let the number of its ancestors and descendants be denoted by $anc(e)$ and $des(e)$ respectively.

For each edge e, it belongs to exactly one of the following three types:

Type A: If $anc(e) \le n_1$ but $des(e) > n_2$, then let $count(e)$ be $anc(e)$.

Type B: If $des(e) \le n_2$ but $anc(e) > n_1$, then let $count(e)$ be $des(e)$.

Type C: If $anc(e) \le n_1$ and $des(e) \le n_2$, then let $count(e)$ be $anc(e) \cdot des(e)$.

$\sum_{e \in E} count(e) \le m \cdot n_1 \cdot n_2$ because the $count$ value on each edge is bounded by $n_1 \cdot n_2$. We will prove that this value upper bounds the number of reachable pairs of vertices.

For each pair of reachable vertices (u, v) (or (u, v) s.t. $u \in S$ and $v \in T$), let $(e_1, \ldots, e_p)$ be the path from u to v. Along the path, anc does not decrease, and dec does not increase. A path belongs to exactly one of the following three types:

Type a: Along the path $anc(e_1) \le anc(e_2) \le \cdots \le anc(e_p) \le n_1$, and $des(e_1) \ge des(e_2) \ge \cdots \ge des(e_p) > n_2$. That is, all the edges are Type A.

Type b: Along the path $des(e_p) \le des(e_{p-1}) \le \cdots \le des(e_1) \le n_2$, and $anc(e_p) \ge anc(e_{p-1}) \ge \cdots \ge anc(e_1) > n_1$. That is, all the edges are Type B.

Type c: Along the path there is some edge e_i so that $anc(e_i) \le n_1$ and $des(e_i) \le n_2$. That is, it has at least one Type C edge.

There will not be other cases, for otherwise if a Type A edge directly connects to a Type B edge without a Type C edge in the middle, then the vertex joining these two edges would have more than n_1 ancestors and more than n_2 descendants.

If a path from u to v is Type a, then its last edge e_p is Type A. If it is Type b, then its first edge e_1 is Type B. If it is Type c, then there is some edge e_i in the path that is Type C. This means:

1. For each Type A edge e, $count(e)$ is at least the number of all Type a pairs (u, v) whose path has e as its last edge.
2. For each Type B edge e, $count(e)$ is at least the number of all Type b pairs (u, v) whose path has e as its first edge.
3. For each Type C edge e, $count(e)$ is at least the number of all Type c pairs (u, v) whose path contains e.

Therefore each path is counted at least once by the $count(e)$ of some edge e. ◀

2.3 Subroutine: reachability across a cut

Now we will show the reduction from $\mathsf{CutPath}_P$ to $\mathsf{Selection}_P$. The high level idea of $\mathsf{CutPath}_P$ is that we think of the reachability relation on $S \times T$ as an $|S| \times |T|$ boolean matrix whose one-entries correspond to reachable pairs of vertices. If we could partition the matrix into all-one combinatorial rectangles, then we can decide all entries within these rectangles by a query to $\mathsf{Selection}_P$, because in the same rectangle, all pairs are reachable.

▷ Claim 8. Consider the reachability matrix of on sets S and T. Let M_S and M_T be the sizes of S and T. If there is a way to partition the matrix into non-overlapping combinatorial rectangles $(S_1, T_1), \ldots, (S_k, T_k)$ of sizes $(r_1, c_1), \ldots, (r_k, c_k)$, and if there is some t so that computing each subproblem of size (r_i, c_i) takes time $r_i \cdot c_i / t(\min(r_i, c_i))$, and all $r_i \ge \ell$, and all $c_i \ge \ell$ for a threshold value ℓ, then all the computation takes total time $O(M_S \cdot M_T / t(\ell))$.

Algorithm 2 $\mathsf{CutPath}_P(S,T)$ on a multitree

```
1  Compute the total weighted size of ancestors anc(v) and descendants des(v) for all
     vertices.
2  Insert all vertices with at least M^α ancestors and M^α descendants into linked list L.
3  while there exists a vertex v ∈ L do
       // we call v a pivot vertex
4      Let A be the set of ancestors of v in S.
5      Let B be the set of descendants of v in T.
6      Add v to A if v ∈ S, otherwise add v to B.
7      Run Selection_P on (A, B). If it returns true then return true.
8      for each a ∈ A do
9          let des(a) = des(a) − |B|.
10         if des(a) < M^α and a ∈ L then remove a from L.
11     for each b ∈ B do
12         let anc(b) = anc(b) − |A|.
13         if anc(b) < M^α and b ∈ L then remove b from L.
14     Remove v from the graph.
15 for each edge (s, t) crossing the cut(S, T) do
16     Let A be the set of ancestors of s (including s) in S.
17     Let B be the set of descendants of t (including t) in T.
18     On all pairs of vertices (a, b) where a ∈ A, b ∈ B, check property P. If P is true
         on any pair of (a, b) then return true.
```

Proof. Let the minimum of all r_i be r_{min} and the minimum of all c_i be c_{min}. Then the factor of time saved for computing each combinatorial rectangle is at least $t(\min(r_{min}, c_{min}))$, greater than $t(\ell)$. So the time spent on all rectangles is at most $O((\sum_{i=1}^{t} c_i)(\sum_{i=1}^{t} r_i)/t(\ell))$, also we have $(\sum_{i=1}^{t} c_i)(\sum_{i=1}^{t} r_i) \le M_S \cdot M_T$ because the rectangles are contained inside the matrix of size $M_S \cdot M_T$ and they do not overlap. So the total time is $O(M_S \cdot M_T/t(\ell))$. ◁

The algorithm $\mathsf{CutPath}_P(S,T)$ is shown in Algorithm 2. It tries to cover the one-entries of the reachability matrix by combinatorial rectangles as many as possible. Finally, for the one-entries not covered, we go through them by exhaustive search, which takes less than quadratic time.

In the beginning, we can compute the total weighted size of ancestors (or descendants) of all vertices in the DAG in $O(M)$ time by going through all vertices by topological order (or reversed topological order).

In each query to $\mathsf{Selection}_P(A,B)$, all vertices in A can reach all vertices in B, because they all go through v. For any pair of reachable vertices $s \in S, t \in T$, if they go through any pivot vertex, then the pair is queried to $\mathsf{Selection}_P$. Otherwise it is left to the end, and checked by exhaustive search on all pairs of reachable vertices.

The calls to $\mathsf{Selection}_P$ correspond to non-overlapping all-one combinatorial rectangles in the reachability matrix. This is because the graph G is a multitree. For each call to $\mathsf{Selection}_P$, the rectangle size is at least $M^\alpha \times M^\alpha$. Thus the total time for all the $\mathsf{Selection}_P$ calls is $O(M^2/t(M^\alpha))$ by Claim 8.

Each time we remove a pivot vertex v, there will be no more paths from set A to set B, for otherwise there would be two distinct paths connecting the same pair of vertices. Thus,

removing a v decreases the total number of weighted-pairs[4] of reachable vertices by at least $M^\alpha \times M^\alpha$. There are $M \times M$ weighted-pairs of vertices, so the total weight (and thus the total number) of pivot vertices like v is at most $(M \times M)/(M^\alpha \times M^\alpha) = M^{2-2\alpha}$.

Each time we find a pivot vertex v, we update the total weighted size of descendants for all its ancestors, and update the total weighted size of ancestors for all its descendants. Because it has at least M^α ancestors and M^α descendants, the value decrease on each affected vertex is at least M^α. So each vertex has decreased its ancestors/descendants values for at most $M/M^\alpha = M^{1-\alpha}$ times. In other words, each vertex can be an ancestor/descendant of at most $M^{1-\alpha}$ pivot vertices. The total time to deal with all ancestors/descendants of all pivot vertices in the while loop is in $O(M \cdot M^{1-\alpha}) = O(M^{2-\alpha})$.

Finally, after the while loop, there are no vertices with both more than M^α ancestors and M^α descendants. In this case, by a weighted version of Lemma 7 (See the full version), the number of weighted-pairs of reachable vertices is bounded by $M \cdot M^\alpha \cdot M^\alpha = M^{1+2\alpha}$. So the total time to deal with these paths is $O(M^{1+2\alpha})$.

Thus the total running time is $O(M^2/t(M^\alpha) + M^{2-\alpha} + M^{1+2\alpha})$. By choosing α and γ to be appropriate constants, we get subquadratic running time.

If $t(M) = M^\epsilon$, then by choosing $\alpha = 1/(2+\epsilon)$, we get running time $M^{2-\epsilon/(2+\epsilon)}$.

3 Application to Least Weight Subpath

In this section we will prove Theorem 3. The reduction from LWSP_C to $\mathsf{StaticLWS}_C$ uses the same structure as the reduction from Path_P to $\mathsf{Selection}_P$ in the proof of Theorem 1 shown in Section 2. Because in LWSP we only consider DAGs, there are no strongly connected components in the graph.

Process $\mathsf{LWSP}_C(G, F_0)$ computes values of F on initial values F_0 defined on all vertices of G. On a given LWSP_C problem, we will reduce it to an asymmetric variation of $\mathsf{StaticLWS}_C$. Process $\mathsf{StaticLWS}_C(A, B, F_A)$ computes all the values of function F_B defined on domain B, given all the values of F_A defined on domain A, such that $F_B(b) = \min_{a\in A}[F_A(s) + c_{a,b}]$. Let N_A and N_B be the total weighted size of A and B respectively. It is easy to see that if $\mathsf{StaticLWS}_C$ on $|N_A| = |N_B|$ is in time $N_A^2/t(N_A)$, then $\mathsf{StaticLWS}_C$ on general A, B is in time $O(N_A \cdot N_B/t(\min(N_A, N_B)))$.

We also define process $\mathsf{CutLWSP}_C(S, T, F_S)$, which computes all the values of F_T defined on domain T, given all the values of F_S on domain S, where $F_T(t) = \min_{s\in S, s\rightsquigarrow t}[F_S(s) + c_{s,t}]$.

The reduction algorithm is adapted from the reduction from Path_P to $\mathsf{Selection}_P$. LWSP_C is analogous to Path_P, $\mathsf{StaticLWS}_C$ is analogous to $\mathsf{Selection}_P$, and $\mathsf{CutLWSP}_C$ is analogous to $\mathsf{CutPath}_P$. In Path_P, we divide the graph into two halves, recursively call Path_P on the subgraphs, and use $\mathsf{CutPath}_P$ to deal with paths from one side of the graph to the other side. Similarly in LWSP_C, we divide the graph into two halves, recursively compute function F on the source side of the graph, then based on these values we call $\mathsf{CutPath}_P$ to compute the initial values of function F on the sink side of the graph, and finally we recursively call LWSP_C on the sink side of the graph. In $\mathsf{CutPath}_P$, we first identify large all-one rectangles in the reachability matrix, and then use $\mathsf{Selection}_P$ to solve them, and finally we go through all reachable pairs of vertices that are not covered by these rectangles. Similarly, in LWSP_C, we will use the similar method to identify large all-one rectangles in the reachability matrix and use $\mathsf{StaticLWS}_C$ to solve them, and finally we go through all reachable pairs of vertices and update F on each of them.

[4] The number of weighted-pairs is defined to be the sum of $w(u) \cdot w(v)$ for all pairs of reachable vertices $u \rightsquigarrow v$.

Algorithm 3 $\mathsf{LWSP}_C(G=(V,E,V_0),F_0)$ on a DAG

if G has only one vertex v **then**
  if $v \in V_0$ **then**
    return $\min(0, F_0(v))$.
  return F_0 on v.
Let M be the weighted size of the problem.
Topological sort all vertices.
Keep adding vertices to S by topological order, until the total weighted size of S exceeds $M/2$. Let the rest of vertices be T.
if $|S| - |T| > M^\gamma$ **then**
  Let x be the last vertex added to S. Remove x from S.
Compute F on domain S, by $F \leftarrow \mathsf{LWSP}_C(G_S, F_0)$, where G_S is the subgraph of G induced by S.
Let $F_T \leftarrow \mathsf{CutLWSP}_C(S, T, F)$.
For each vertex $t \in T$, let $F_0(t) \leftarrow \min(F_0(t), F_T(t))$.
if x exists **then**
  Compute $F_x \leftarrow \mathsf{CutLWSP}_C(S, x, F)$ for vertex x.
  Compute F on vertex x by $F(x) \leftarrow \min(F_0(x), F_x(x))$.
  Let $F'_T \leftarrow \mathsf{CutLWSP}_C(x, T, F)$.
  For each vertex $t \in T$, let $F_0(t) \leftarrow \min(F_0(t), F'_T(t))$.
Compute F on domain T, by $F \leftarrow \mathsf{LWSP}_C(G_T, F_0)$, where G_T is the subgraph of G induced by T.
return F on domain V.

The algorithm LWSP_C is similar as Path_P (Algorithm 1), and is defined in Algorithm 3. Initially, we let $F(v) \leftarrow 0$ for all $v \in V_0$, and let $F(v) \leftarrow +\infty$ for all $v \notin V_0$. We run $\mathsf{LWSP}_C(G, F_0)$ on the whole graph.

The algorithm $\mathsf{CutLWSP}_C(S, T, F_S)$ is adapted from $\mathsf{CutPath}_P$ (Algorithm 2), with the following changes:

1. In the beginning, $F_T(t)$ is initialized to ∞ for all $t \in T$.
2. Each query to $\mathsf{Selection}_P(A, B)$ in $\mathsf{CutPath}_P$ is replaced by
   a. Compute F_B on domain B by $\mathsf{StaticLWS}_C(A, B, F_S)$.
   b. For each vertex b in B, let $F_T(b)$ be the minimum of the original $F_T(b)$ and $F_B(b)$.
3. Whenever processing a pair of vertices s, t such that s is can reach t in either the preprocessing phase or the final exhaustive search phase, we let $F_T(t) \leftarrow F_S(s) + c_{s,t}$ if $F_S(s) + c_{s,t} < F_T(t)$.
4. In the end, the process returns F_T, the target function on domain T.

The proof of correctness will be shown in the full version. The time complexity of this reduction algorithm follows from the argument of Section 2.

4 Open problems

One open problem is to study Path_P and LWSP_C on general DAGs. Also, we would like to consider the case where the graph is not sparse, where we can use $O(MN)$ as the baseline time complexity instead of $O(M^2)$.

It would also be desirable to study the fine-grained complexity of the DAG versions of other quadratic time solvable dynamic programming problems, e.g. the Longest Common Subsequence problem.

References

1. Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Fine-grained complexity of analyzing compressed data: Quantifying improvements over decompress-and-solve. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 192–203. IEEE, 2017.
2. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 59–78. IEEE, 2015.
3. Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 41–57. SIAM, 2019.
4. Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1681–1697. SIAM, 2014.
5. Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 375–388. ACM, 2016.
6. Amir Abboud and Kevin Lewi. Exact weight subgraphs and the k-sum conjecture. In *International Colloquium on Automata, Languages, and Programming*, pages 1–12. Springer, 2013.
7. Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 218–230. SIAM, 2015.
8. Amir Abboud, Virginia Vassilevska Williams, and Joshua Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete Algorithms*, pages 377–391. SIAM, 2016.
9. Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *International Colloquium on Automata, Languages, and Programming*, pages 39–51. Springer, 2014.
10. Udit Agarwal and Vijaya Ramachandran. Fine-grained Complexity for Sparse Graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 239–252, New York, NY, USA, 2018. ACM. doi:10.1145/3188745.3188888.
11. Alok Aggarwal, Baruch Schieber, and Takeshi Tokuyama. Finding a minimum-weight k-link path in graphs with the concave Monge property and applications. *Discrete & Computational Geometry*, 12(3):263–280, 1994.
12. Alfred V Aho and Jeffrey D Ullman. Universality of data retrieval languages. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 110–119. ACM, 1979.
13. Eric Allender and Klaus-Jörn Lange. $StUSPACE(\log n) \subseteq DSPACE(\log^2 n/\log\log n)$. In *International Symposium on Algorithms and Computation*, pages 193–202. Springer, 1996.
14. Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 51–58. ACM, 2015.

15 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 457–466. IEEE, 2016.

16 Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Towards tight approximation bounds for graph diameter and eccentricities. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 267–280. ACM, 2018.

17 Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

18 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 661–670. IEEE, 2014.

19 Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 307–318. IEEE, 2017.

20 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 79–97. IEEE, 2015.

21 Karl Bringmann and Marvin Künnemann. Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds. *International Journal of Computational Geometry & Applications*, 27(01n02):85–119, 2017.

22 Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete Fréchet distance. *Journal of Computational Geometry*, 7(2):46–76, 2015.

23 Timothy M Chan and Ryan Williams. Deterministic APSP, Orthogonal Vectors, and More: Quickly derandomizing Razborov-Smolensky. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1246–1255. SIAM, 2016.

24 S.C. Chen, J.Y. Wu, G.S. Huang, and R.C.T. Lee. Finding a Longest Increasing Subsequence on a Galled Tree. In *the 28th Workshop on Combinatorial Mathematics and Computation Theory, Penghu, Taiwan*, 2011.

25 Michael L Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975.

26 Zvi Galil and Kunsoo Park. A linear-time algorithm for concave one-dimensional dynamic programming. *Information Processing Letters*, 33(6):309–311, 1990. doi:10.1016/0020-0190(90)90215-J.

27 Jiawei Gao and Russell Impagliazzo. The Fine-Grained Complexity of Strengthenings of First-Order Logic. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:9, 2019. URL: https://eccc.weizmann.ac.il/report/2019/009.

28 Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams. Completeness for First-order Properties on Sparse Structures with Algorithmic Applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 2162–2181, 2017.

29 Jerrold R Griggs, Wei-Tian Li, and Linyuan Lu. Diamond-free families. *Journal of Combinatorial Theory, Series A*, 119(2):310–322, 2012.

30 Daniel S Hirschberg and Lawrence L Larmore. The least weight subsequence problem. *SIAM Journal on Computing*, 16(4):628–638, 1987.

31 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

32 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

33 Thor Johnson, Neil Robertson, Paul D Seymour, and Robin Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.

34 Donald E Knuth and Michael F Plass. Breaking paragraphs into lines. *Software: Practice and Experience*, 11(11):1119–1184, 1981.

35 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1272–1287. SIAM, 2016.

36 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the Fine-Grained Complexity of One-Dimensional Dynamic Programming. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:15, 2017.

37 Leonid Libkin. *Elements of finite model theory*. Springer Science & Business Media, 2013.

38 Guan-Yu Lin, Jia jie Liu, and Yue-Li Wang. Finding a Longest Increasing Subsequence from the Paths in a Complete Bipartite Graph. In *Proceedings of the 29th Workshop on Combinatorial Mathematics and Computation Theory*, 2012.

39 Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1236–1252. Society for Industrial and Applied Mathematics, 2018.

40 Neil Robertson and Paul D Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

41 Neil Robertson and P.D Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

42 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 515–524. ACM, 2013.

43 Baruch Schieber. Computing a minimum weightk-link path in graphs with the concave monge property. *Journal of Algorithms*, 29(2):204–222, 1998.

44 Larry Joseph Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, Massachusetts Institute of Technology, 1974.

45 Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146. ACM, 1982.

46 Robert Wilber. The concave least-weight subsequence problem revisited. *Journal of Algorithms*, 9(3):418–425, 1988.

47 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

48 Ryan Williams. Faster decision of first-order graph properties. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, page 80. ACM, 2014.

49 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 645–654. IEEE, 2010.

50 F Frances Yao. Efficient dynamic programming using quadrangle inequalities. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 429–435. ACM, 1980.

A List of problem definitions and class definitions

Here we list the main problems studied in this paper.

LWS$_C$: Given elements $x_1, \ldots, x_n$ and value $F(0) = 0$, compute $F(j) = \min_{0 \le i < j}[F(i) + c_{i,j}]$ for all $j \in \{1, \ldots, n\}$.

StaticLWS$_C$: Given elements $x_1, \ldots, x_{2n}$ and values of $F(i)$ on all $i \in \{1, \ldots, n\}$, compute $F(j) = \min_{i \in \{1,\ldots,n\}}[F(i) + c_{i,j}]$ for all $j \in \{n+1, \ldots, 2n\}$.

LWSP$_C$: Given graph $G = (V, E)$ and starting vertex set $V_0 \subseteq V$, compute on each $v \in V$, the value of $F(v)$, where

$$F(v) = \begin{cases} \min(0, \min_{u \rightsquigarrow v}[F(u) + c_{u,v}]), & \text{for } v \in V_0 \\ \min_{u \rightsquigarrow v}[F(u) + c_{u,v}], & \text{for } v \notin v_0 \end{cases}$$

CutLWSP$_C$: On DAG G with a cut (S, T) where edges are only directed from S to T, given the values of function F_S on S, for all $t \in T$ compute $F_T(t) = \min_{s \in S, s \rightsquigarrow t}[F_S(s) + c_{s,t}]$.

Selection$_P$: On two sets X, Y, decide whether $(\exists x \in X)(\exists y \in Y)P(x, y)$.

Path$_P$: On graph $G = (V, E)$, decide whether $(\exists x \in V)(\exists y \in V)[\mathrm{TC}_E(x, y) \wedge P(x, y)]$.

ListPath$_P$: On graph $G = (V, E)$, for all $x \in V$, decide whether $(\exists y \in V)[\mathrm{TC}_E(x, y) \wedge P(x, y)]$.

CutPath$_P$: On graph $G = (V, E)$ with cut (S, T) where edges only direct from S to T, decide whether $(\exists x \in S)(\exists y \in T)[\mathrm{TC}_E(x, y) \wedge P(x, y)]$.

***PathFO$_3$*:** class of Path$_P$ problems such that P is of form $\exists z \psi(x, y, z)$ or $\forall z \psi(x, y, z)$, where ψ is a quantifier-free logical formula.

***ListPathFO$_3$*:** class of ListPath$_P$ problems such that P is of form $\exists z \psi(x, y, z)$ or $\forall z \psi(x, y, z)$, where ψ is a quantifier-free logical formula.

B Problem examples

We give a list of problems that can be considered as instances of LWSP$_C$ or Path$_P$.

Trip Planning (LWSP version of Airplane Refueling) On a DAG where vertices represent cities and edges are roads, we wish to find a path for a vehicle, along which we wish to find a sequence of cities where the vehicle can rest and add fuel so that the cost is minimized. The cost of traveling between cities x and y is defined by cost $c_{x,y}$. $c_{x,y}$ can be defined in multiple ways, e.g. $c_{x,y}$ is $cost(y)$ if $dist(x, y) \leq M$ and ∞ otherwise. $dist(x, y)$ is the distance between x, y that can be computed by the positions of x, y. M is the maximal distance the vehicle can travel without resting. $cost(y)$ is the cost for resting at position y.

Longest Subset Chain on graphs (LWSP version of Longest Subset Chain) On a DAG where each vertex corresponds to a set, we want to find a longest chain in a path of the graph such that each set is a subset of its successor. Here $c_{x,y}$ is -1 if S_x is a subset of S_y, and ∞ otherwise.

Multi-currency Coin Change (LWSP version of Coin Change) Consider there are two different currencies, so there are two sets of coins. We need to find a way to get value V_1 for currency #1 and value V_2 for currency #2, so that the total weight of coins is minimized. Each pair of values $v_1 \in \{0, \ldots, V_1\}$ and $v_2 \in \{0, \ldots, V_2\}$ can be considered as a vertex. We connect vertex (v_1, v_2) to (v_1', v_2') iff $v_1' = v_1 + 1$ or $v_2' = v_2' + 1$. The whole graph is a grid, and we wish to find a subsequence of a path from $(0, 0)$ to (V_1, V_2) so that the cost is minimized. The cost is defined by $C_{(v_1,v_2),(v_1',v_2)} = w_{1,v_1'-v_1}$ and $C_{(v_1,v_2),(v_1,v_2')} = w_{2,v_2'-v_2}$, where $w_{i,j}$ is the weight of a coin of value j from currency $\#i$.

Pretty Printing with alternative expressions (LWSP version of Pretty Printing) The Pretty Printing problem is to break a paragraph into lines, so that each line have roughly the same length. If a line is too long or too short, then there is some cost depending on the line length. The goal of the problem is to minimize the cost.

For some text, it is hard to print prettily. For example, if there are long formulas in the text, then sometimes its line gets too wide, but if we move the formula into the next line, the original line has too few words. One solution for this issue is to use alternate

wording for the sentence, to rephrase a part of a sentence to its synonym. These sentences have different lengths, and formulas in some of them will be displayed better than others. These different ways can be considered as different paths in a graph, and we wish to find one sentence that has the minimal Pretty Printing cost.

A Path$_P$ instance Say we have a set of words, and we want to find a word chain (a chain of words so that the last letter of the previous word is the same as the first letter of the next word) so that the first word and the last word satisfy some properties, e.g. they do not have similar meanings, they have the same length, they don't have the same letters on the same positions, etc. Each word corresponds to a vertex in the graph. For words that can be consecutive in a word chain, we add an edge to the words.

Resolving Infeasibility of Linear Systems: A Parameterized Approach

Alexander Göke
Universität Bonn, Bonn, Germany
Technische Universität Hamburg, Hamburg, Germany
alexander.goeke@uni-bonn.de

Lydia Mirabel Mendoza Cadena
Eötvös Loránd University, Budapest, Hungary
lmmendoza@ciencias.unam.mx

Matthias Mnich
Universität Bonn, Bonn, Germany
Technische Universität Hamburg, Hamburg, Germany
mmnich@uni-bonn.de

Abstract

Deciding feasibility of large systems of linear equations and inequalities is one of the most fundamental algorithmic tasks. However, due to inaccuracies of the data or modeling errors, in practical applications one often faces linear systems that are *infeasible*.

Extensive theoretical and practical methods have been proposed for post-infeasibility analysis of linear systems. This generally amounts to detecting a feasibility blocker of small size k, which is a set of equations and inequalities whose removal or perturbation from the large system of size m yields a feasible system. This motivates a parameterized approach towards post-infeasibility analysis, where we aim to find a feasibility blocker of size at most k in fixed-parameter time $f(k) \cdot m^{\mathcal{O}(1)}$.

On the one hand, we establish parameterized intractability (W[1]-hardness) results even in very restricted settings. On the other hand, we develop fixed-parameter algorithms parameterized by the number of perturbed inequalities and the number of positive/negative right-hand sides. Our algorithms capture the case of Directed Feedback Arc Set, a fundamental parameterized problem whose fixed-parameter tractability was shown by Chen et al. (STOC 2008).

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

Keywords and phrases Infeasible subsystems, linear programming, fixed-parameter algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.17

Funding *Alexander Göke*: Supported by DAAD with funds of the Bundesministerium für Bildung und Forschung (BMBF) and by DFG project MN 59/1-1. *Matthias Mnich*: Supported by DAAD with funds of the Bundesministerium für Bildung und Forschung (BMBF) and by DFG project MN 59/4-1.

SPONSORED BY THE Federal Ministry of Education and Research

Acknowledgements We thank an anonymous reviewer of an earlier version for a suggestion on run time improvements.

1 Introduction

Solving systems of linear equations and inequalities constitutes an algorithmic task of fundamental importance. The data that is used in these systems though may be subject to inaccuracies and uncertainties, and therefore may lead to systems which are infeasible. Another source of infeasibility may be modeling errors, or simply incompatibility of constraints. Infeasibility itself allows for little conclusions; for a large system of millions of inequalities, infeasibility may stem from a very small subset of data. A natural question is therefore to

detect the smallest number of changes which must be made to a given system in order to make it feasible. The analysis of infeasible linear systems has been extensively investigated [1, 2, 4, 11, 12, 13, 28, 31]; we refer to the book by Chinneck [13] for an overview.

Formally, the Minimum Feasibility Blocker (MinFB) problem takes as input a system $\mathcal{S}$ of linear inequalities $Ax \leq b$ and asks for a smallest subset $\mathcal{I}$ such that $\mathcal{S} \setminus \mathcal{I}$ is *feasible*. As $\mathcal{I}$ "blocks" the feasibility of $\mathcal{S}$, we refer to $\mathcal{I}$ as a *feasibility blocker*; we avoid calling $\mathcal{I}$ a "solution", to avoid confusion with the solution of the linear system $\mathcal{S} \setminus \mathcal{I}$. Further note that instead of removing the set $\mathcal{I}$ of inequalities, we can equivalently perturb the right-hand sides $b_{\mathcal{I}}$ of the inequalities in $\mathcal{I}$; that is, we can increase the b-values of inequalities in $\mathcal{I}$ to a value such that the perturbed system becomes feasible. Of course, when talking about feasibility, we have to specify over which field, and our choice here is the field $\mathbb{Q}$. Over this field, MinFB is NP-hard [33]; as the feasibility of a linear system can be tested in polynomial time (e.g., by the ellipsoid method [24]) the MinFB problem is NP-complete. Thus, there is a simple XP-algorithm testing every possible feasibility blocker of size at most k. Due to its importance, the MinFB problem has been thoroughly investigated from several different viewpoints, including approximation algorithms [1, 28], polyhedral combinatorics [31], heuristics [11], mixed-integer programming [15], and hardness of approximation [2].

Here we take a new perspective on the MinFB problem, based on parameterized complexity. In parameterized complexity, the problem input of size n is additionally equipped with one or more integer parameters k and one measures the problem complexity in terms of both n and k. The goal is to solve such instances by *fixed-parameter algorithms*, which run in time $f(k) \cdot n^{\mathcal{O}(1)}$ for some computable function f. The motivation is that fixed-parameter algorithms can be practical for small parameter values k even for inputs of large size n, provided that the function f exhibits moderate growth. This contrasts them with algorithms that require time $n^{f(k)}$, which cannot presumed to be practical for large input sizes n. To show that such impractical run times are best possible, a common approach is to show the problem to be W[1]-hard; a standard hypothesis in parameterized complexity is that no W[1]-hard problem admits a fixed-parameter algorithm. For background on parameterized complexity, we refer to the book by Cygan et al. [18].

For the MinFB problem, arguably the most natural parameter is the minimum size k of a feasibility blocker $\mathcal{I}$. The motivation for this choice of parameter is that in applications, we are interested in *small* feasibility blockers $\mathcal{I}$; e.g., Chakravarti [9] argues that a feasibility blocker "with too large a cardinality may be hard to comprehend and may not be very useful for post-infeasibility analysis." Guillemot [25] explicitly posed the question of resolving the parameterized complexity of MinFB; he conjectured that the problem is fixed-parameter tractable parameterized by the size of a minimum feasibility blocker for matrices with at most 2 non-zero entries per row.

Another motivation for our approach comes from the fact that MinFB captures one of the most important problems in parameterized complexity, namely Directed Feedback Arc Set (DFAS): given a digraph G, decide if G admits a directed feedback arc set of size at most k, which is a set F such that $G - F$ is an acyclic digraph (DAG). It was a long-standing open question whether DFAS admits a fixed-parameter algorithm parameterized by the size k of the smallest directed feedback arc set, until Chen et al. [10] gave an algorithm with run time $4^k k! n^{\mathcal{O}(1)}$. The currently fastest algorithm for DFAS runs in time $4^k k! k \cdot \mathcal{O}(n+m)$, and is due to Lokshtanov et al. [29]. It is not difficult to give a parameter-preserving reduction from DFAS to MinFB: for every arc (u, v) of the digraph G that serves as input to DFAS we add the inequality $x_u - x_v \leq -1$ to the linear system $Ax \leq b$. Directed feedback arc sets F

of G are then mapped to feasibility blockers of the same size by removing the constraints corresponding to arcs in F, and vice-versa. Note that the constraint matrix A arising this way is totally unimodular, and each row has exactly two non-zero entries, one $+1$ and one -1. Totally unimodular matrices A whose every row has at most two non-zero entries, one $+1$ and one -1, are known as *difference constraints*; testing feasibility of systems of difference constraints has been investigated extensively [32, 35] due to their practical relevance most notably in temporal reasoning. It is therefore interesting to know whether the more general MinFB problem also admits a fixed-parameter algorithm for parameter k, even for special cases like totally unimodular matrices A (where testing feasibility is easy).

Another case of interest for MinFB is when the constraint matrix A has bounded treewidth, where the treewidth of A is defined as the treewidth of the bipartite graph that originates from assigning one vertex to every row and every column of A and connecting any two vertices by an edge whose corresponding entry in A is non-zero. Fomin et al. [21] gave a fast algorithm for MinFB with constraint matrices of bounded treewidth for the setting $k = 0$, i.e. checking for feasibility without deleting any constraints. At the same time, Bonamy et al. [8] showed that DFAS – a special case of MinFB – is fixed-parameter tractable parameterized by the treewidth of the underlying undirected graph of the input digraph[1]. So the questions arise whether Fomin et al.'s algorithm can be extended to arbitrary values of k, or whether Bonamy et al.'s algorithm can be extended from DFAS to MinFB.

One of the main currently unresolved questions around DFAS is whether it admits a *polynomial compression*. That is, one seeks an algorithm that, given any directed graph G and integer k, in polynomial time computes an instance I of a decision problem Π whose size is bounded by some polynomial $p(k)$, such that G admits a feedback arc set of size at most k if and only if I is a "yes"-instance of Π. The question for a polynomial compression has been stated numerous times as an open problem [5, 19, 17, 30]; from the algorithms by Chen et al. [10] and Lokshtanov et al. [29] only an *exponential* bound on the size of I follows. On the other hand, parameterized complexity provides tools such as cross-composition to rule out the existence of such polynomials $p(k)$ modulo the non-collapse of the polynomial hierarchy; we refer to Bodlaender et al. [6] for background. Given the elusiveness of this problem, we approach the (non-)existence of polynomial compression for DFAS from the angle of the more general MinFB problem.

1.1 Our results

We first show that the MinFB problem is strictly more general than DFAS, even for totally unimodular matrices, assuming that $\mathsf{FPT} \neq \mathsf{W}[1]$.

▶ **Theorem 1.** *The* MinFB *problem is* $\mathsf{W}[1]$*-hard parameterized by the minimum size k of a feasibility blocker, even for difference constraints and right-hand sides $b \in \{\pm 1\}^m$.*

Theorem 1 therefore disproves (assuming $\mathsf{FPT} \neq \mathsf{W}[1]$) the conjecture of Guillemot [25] that finding the minimum number of unsatisfied equations or inequalities is fixed-parameter tractable for linear systems with at most two variables per equation or inequality.

Given this strong parameterized intractability result, we resort to identifying tractable fragments (or classes of instances) of MinFB. In particular, we look for algorithms which

[1] The algorithm of Bonamy et al. [8] is stated for the vertex deletion problem, but it can be modified to work for DFAS as well. Note that the standard reduction from DFAS to the vertex deletion problem which preserves the solution size does not necessarily result in a digraph whose underlying undirected graph has bounded treewidth even if the DFAS instance has this property.

solve fragments of MinFB which capture the fundamental DFAS problem. As relevant parameters, we identify the number b_+ of positive entries in the right-hand side vector b, as well as the number b_- of negative entries in b. This choice comes from the fact that the case of $b_+ = 0$ generalizes the DFAS problem, whereas the case of $b_- = 0$ is always feasible as the all-0 vector is a trivial solution for a system of the form $Ax \leq b$.

Our positive algorithmic results for these parameters are as follows:

▶ **Theorem 2.** *There is an algorithm that solves* MinFB *for systems $\mathcal{S}$ of m difference constraints over n variables and right-hand sides $b \in \{\pm 1\}^m$ in time $2^{\mathcal{O}(k^3+b_++k\log b_+)} \cdot n^{\mathcal{O}(1)}$.*

▶ **Theorem 3.** *There is an algorithm that solves* MinFB *for systems $\mathcal{S}$ of m difference constraints over n variables and right-hand sides $b \in \{\pm 1\}^m$ in time $(k+1)(b_-)^{k+1}\mathcal{O}(nm)$.*

Armed with these fixed-parameter algorithms for parameters $k + b_+$ and $k + b_-$, it is time to consider the question of polynomial compressions for those tractable fragments of MinFB. Such polynomial compression would be particularly interesting, as it could be a step towards obtaining a polynomial compression for DFAS (where $b_+ = 0$); so a polynomial compression for MinFB for parameter k or $k + b_+$, even for node-arc incidence matrices, would imply a polynomial compression for DFAS (as the reduction from DFAS to MinFB does not increase the parameter).

Interestingly, we can actually rule out a polynomial compression for MinFB parameterized by $k + b_-$:

▶ **Theorem 4.** *Assuming* NP $\not\subseteq$ coNP/*poly,* MinFB *does not admit a polynomial compression when parameterized by $k + b_-$ even for systems A of difference constraints and right-hand sides $b \in \{\pm 1\}^m$.*

The most intriguing open question arising from this result is whether our hardness result can be strengthened to rule out a polynomial compression for MinFB parameterized by $k + b_+$.

As mentioned, Fomin et al. [21] give an algorithm that solves MinFB for constraint matrices of bounded treewidth for $k = 0$. And Bonamy et al. [8] give an algorithm that solves the special case of MinFB known as DFAS for constraint matrices of bounded treewidth. Here we show that, somewhat surprisingly, MinFB is NP-hard even for constraint matrices of *constant* pathwidth (which are a subclass of matrices with constant treewidth).

▶ **Theorem 5.** *The* MinFB *problem is* NP*-hard even for constraint matrices of pathwidth 6.*

Due to space constraints, proofs of statements marked by ($\star$) are deferred to the full version of this paper.

1.2 Related work

In fundamental work, Arora, Babai, Stern and Sweedyk [2] considered the problem of removing a smallest set of *equations* to make a given system of linear equations feasible over $\mathbb{Q}$. They gave strong inapproximability results, showing that finding *any* constant-factor approximation is NP-hard. Berman and Karpinski [4] gave the first (randomized) polynomial-time algorithm with sublinear approximation ratio for this problem.

Giannopolous, Knauer and Rote [22] considered the "dual" of MinFB from a parameterized point of view: namely, in MaxFS we ask for a largest subsystem of an n-dimensional linear system $\mathcal{S}$ which is feasible over $\mathbb{Q}$. They showed that deciding whether a feasible subsystem of at least ℓ inequalities in $\mathcal{S}$ exists is W[1]-hard parameterized by $n + \ell$, even when $\mathcal{S}$ consists of equations only.

For systems of equations over *finite* fields, finding minimum feasibility blockers has been considered from a parameterized perspective. In particular, over the binary field $\mathbb{F}_2$, Crowston et al. [16] prove W[1]-hardness even if each equation has exactly three variables and every variable appears in exactly three equations; they further give a fixed-parameter algorithm for the case where each equation has at most two variables.

2 Preliminaries

Throughout, we work with finite and loop-less directed graphs G, whose vertex set we denote by $V(G)$ and arc set by $A(G)$. For a vertex $v \in V(G)$, denote by $\delta^-(v)$ the incoming arcs of v, i.e., arcs of the form (w, v) for some $w \in V(G)$. Likewise, denote by $\delta^+(v)$ the outgoing arcs of v. A *walk* W in G is a sequence of vertices $W = (v_0, v_1, \ldots, v_\ell)$ such that $(v_i, v_{i+1}) \in A(G)$ for $i = 0, \ldots, \ell - 1$. We call ℓ the *length* of the walk. A walk is *closed* if $v_0 = v_\ell$. If all vertices are distinct we call the walk a *path*, if all except v_0 and v_ℓ are distinct we call it a cycle. For two walks W, R where the last vertex of W equals the first vertex of R, let $W \circ R$ be the *concatenation* of W and R, which is the sequence of all vertices in W followed by all vertices in R except the first. Our directed graphs G often come with arc weights $w : A(G) \to \mathbb{Q}$; the *weight* of a cycle C in G is then equal to the sum of its arc weights. In that spirit, we call a cycle *negative* (*non-negative*, *positive*) if its weight is *negative* (*non-negative*, *positive*). A *shortest* path or cycle is a path or cycle of minimum length. Note that "shortest" does not refer to the weight of a cycle.

For our hardness results we will use two different kinds of hardness. The first one is W[1]-hardness which under the standard assumption $\mathsf{W}[1] \neq \mathsf{FPT}$ implies that there is no fixed-parameter tractable algorithm for problems of this type. The other hardness considers compression: A *polynomial compression* of a language L into a language Q is a polynomial-time computable mapping $\Phi : \Sigma^* \times \mathbb{N} \to \Sigma^*$, $\Phi((x, k)) \mapsto y$ such that $((x, k) \in L \Leftrightarrow y \in Q)$ and $|y| \leq k^{\mathcal{O}(1)}$ for all $(x, k) \in \Sigma^* \times \mathbb{N}$. Many natural parameterized problems do not admit polynomial compressions, under the hypothesis that $\mathsf{NP} \subsetneq \mathsf{coNP}/poly$.

Both types of hardness can be transferred to other problems by "polynomial parameter transformations", which were first proposed by Bodlaender et al. [7].

▶ **Definition 6.** *Let Σ be an alphabet. A* polynomial parameter transformation (PPT) *from a parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ to a parameterized problem $\Pi' \subseteq \Sigma^* \times \mathbb{N}$ is a polynomial-time computable mapping $\Phi : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}, (x, k) \mapsto (x', k')$, such that $k' = k^{\mathcal{O}(1)}$, and $(x, k) \in \Pi \Leftrightarrow (x', k') \in \Pi'$ for all $(x, k) \in \Sigma^* \times \mathbb{N}$. Two parameterized problems are* parameter-equivalent *if there are PPTs in both directions and the transformations additionally fulfil that $k' = k$.*

Note that polynomial parameter transformations are transitive. Further, a PPT from Π to Π' together with a polynomial compression for Π' yields a polynomial compression for Π. This can be used to rule out polynomial compressions:

▶ **Proposition 7** ([27]). *Let Π, Π' be parameterized problems. If there is a polynomial parameter transformation from Π to Π' and Π admits no polynomial compression, then neither does Π'.*

Next, we formally define the MinFB problem. Here and throughout the rest of the paper, we denote by $a_{i,\bullet}$ the i^{th} row of the matrix A.

Minimum Feasibility Blocker (MinFB)	Parameter: k
Input:	A coefficient matrix $A \in \mathbb{Q}^{m \times n}$, a right-hand side vector $b \in \mathbb{Q}^m$ and an integer k.
Task:	Find a set $\mathcal{I} \subseteq \{1, \ldots, m\}$ of size at most k such that $(a_{i,\bullet} \cdot x \leq b_i)_{i \in \{1,\ldots,m\} \setminus \mathcal{I}}$ is feasible for some $x \in \mathbb{Q}^n$.

By multiplying rows of A and the corresponding entries of b with (-1) the MinFB problem also covers in a parameter equivalent way the case where some inequalities are of the type $a_{i,\bullet}x \geq b_i$ if there is no sign restriction on the entries of A and b.

Furthermore, equations of the form $a_{i,\bullet} \cdot x = b_i$ can be written as the two inequalities $a_{i,\bullet} \cdot x \leq b_i$ and $-a_{i,\bullet} \cdot x \leq -b_i$ (equivalent to $a_{i,\bullet} \cdot x \geq b_i$). In a feasible solution x^* ignoring such an equation, at most one of the above inequalities is violated. Thus, the MinFB problem with equations can be reduced to the formulation of the MinFB as formulated above without changing the parameter. Conversely though, MinFB as presented above in general cannot be expressed by the MinFB problem having only equations. However, if we allow additionally inequalities with only one variable or require all variables to be non-negative, those are representable by adding slack variables and (if necessary) splitting variables into two non-negative parts x_i^+ and x_i^-.

For a graph H, a *tree decomposition* (*path decompostion*) is a pair $(T, \mathcal{B})$ where T is a tree (path) and $\mathcal{B}$ a collection of bags $B_v \subseteq V(H)$, each bag corresponding to some node $v \in V(T)$. The bags have the property that for any edge of H both its endpoints appear in some common bag in $\mathcal{B}$, and for each vertex $v \in V(H)$ the bags containing v form a subtree of T. The *width* of $(T, \mathcal{B})$ is defined as the largest bag size of $\mathcal{B}$ minus one. The *treewidth* (*pathwidth*) of H is the minimum width over all tree (path) decompositions of H.

3 Parameterized Intractability of Minimum Feasibility Blocker

In this section we show W[1]-hardness of the MinFB problem by giving a reduction from the Bounded Edge Directed (s,t)-Cut problem. This problem takes as input a digraph G, vertices s, t, and integers $k, \ell \in \mathbb{N}$, and asks for a set $X \subseteq E(G)$ of size at most k such that $G - X$ contains no s-t-paths of length at most ℓ. Golovach and Thilikos [23] proved its parameterized intractability for parameter k:

▶ **Proposition 8** ([23]). Bounded Edge Directed (s,t)-Cut *is* W[1]*-hard when parameterized in k even for the special case where G is a DAG.*

The Bounded Edge Directed (s,t)-Cut was also considered by Fluschnik et al. [20]. They showed that Bounded Edge Directed (s,t)-Cut does not admit a kernel of size polynomial in k and ℓ, assuming NP $\not\subseteq$ coNP/*poly*, even for acyclic input digraphs. In fact, their construction allows for a stronger result, ruling out a polynomial compression:

▶ **Proposition 9** ([20]). *Assuming* NP $\not\subseteq$ coNP/*poly*, Bounded Edge Directed (s,t)-Cut *does not admit a polynomial compression in $k + \ell$ even when G is a DAG.*

To get to the MinFB problem, we first consider as an intermediate step the Directed Small Cycle Transversal problem: given a directed graph G and integers k, ℓ, the task is to find a set $X \subseteq E(G)$ of size at most k such that $G - X$ contains no cycles of length at most ℓ. Fluschnik et al. [20] showed that Directed Small Cycle Transversal does not admit a kernel of size polynomial in k and ℓ, unless NP $\subseteq$ coNP/*poly*. Again their result can be strengthened to not admitting a polynomial compression. They further observed that

Directed Small Cycle Tranversal admits a simple branching algorithm that runs in time $\mathcal{O}(\ell^k \cdot n \cdot (n+m))$. Here we argue that a dependence on both parameters k and ℓ is necessary for fixed-parameter tractability:

▶ **Lemma 10.** *There is a polynomial parameter transformation from* Bounded Edge Directed (s,t)-Cut *in DAGs with parameter k (parameter ℓ, parameter $k+\ell$) to* Directed Small Cycle Transversal *with parameter k (resp. ℓ, resp. $k+\ell$).*

This even holds if Directed Small Cycle Transversal *is restricted to instances where there are vertices $s,t \in V(G)$ such that all cycles of G consist of an s-t-path and an arc of the form (t,s).*

Proof. Let (G,s,t,k,ℓ) be a Bounded Edge Directed (s,t)-Cut instance where G is a DAG. As G is a DAG, it admits a topological ordering $v_1,\ldots,v_{|V(G)|}$ of its vertices, so that there are no arcs (v_i,v_j) for $j<i$. Without loss of generality, let $v_1=s$ and $v_{|V(G)|}=t$, as vertices before s or after t in a topological ordering are never part of any s-t-path.

We now create a digraph G' from G by adding $k+1$ parallel arcs $a_1,\ldots,a_{k+1}$ from t to s. Then every cycle in G' consists of an s-t-path and an arc a_i, as G was acyclic. Set $\ell'=\ell+1$. Then (G',k,ℓ') is an instance of Directed Small Cycle Transversal. The above transformation can be done in polynomial time and the parameter increases by at most one (depending on whether ℓ is part of the parameter).

It remains to show that (G',k,ℓ') is a "yes"-instance of Directed Small Cycle Transversal if and only if (G,s,t,k,ℓ) is a "yes"-instance of Bounded Edge Directed (s,t)-Cut.

For the forward direction, let X' be a solution to (G',k,ℓ'). Consider $X = X' \setminus \{a_1,\ldots,a_{k+1}\}$. For sake of contradiction, suppose that $G-X$ contains an s-t-path P of length at most ℓ. As $|X'|\leq k$, it can not contain all arcs a_i. Without loss of generality, $a_1 \notin X'$. Then P followed by a_1 is a cycle in $G'-X'$ of length at most $\ell+1=\ell'$ – a contradiction to X' being a solution to (G',k,ℓ').

For the reverse direction, let X be a solution to (G,s,t,k,ℓ). Then X is also a solution to (G',k,ℓ') by the following argument. Suppose, for sake of contradiction, that $G'-X$ contains a cycle C of length at most ℓ'. By the structure of G', C consists of an s-t-path P in $G-X$ and an arc a_i. Then $|P|=|C|-1\leq\ell$, contradicting that X is a solution to (G,k,ℓ). ◀

Now we introduce another cycle deletion problem, this time on arc-weighted digraphs.

Negative Directed Feedback Arc Set (Negative DFAS) Parameter: k
Input: A digraph G, a weight function $w : A(G) \to \mathbb{Q}$ and an integer k.
Task: Find a set $X \subseteq A(G)$ of size at most k such that $G-X$ has no negative cycles.

▶ **Lemma 11.** *There is a PPT from* Directed Small Cycle Transversal *on instances where every cycle uses an arc of type (t,s) when parameterized by k (by $k+\ell$) to* Negative DFAS *parameterized by k (resp. $k+w_-$, where w_- is the number of arcs with negative weight); this even holds in the case where $w : A(G) \to \{\pm 1\}$.*

Proof. We start with a Directed Small Cycle Transversal instance (G,k,ℓ) as described in the lemma. Let $a_1,\ldots,a_p$ be the arcs of the form (t,s). For any $p \geq k+1$ there is always an arc which survives the deletion of some arc set of at most k elements. So we can assume $p \leq k+1$ as deleting superfluous arcs does not change the solution. Set $A_{+1} = A(G) \setminus \{a_1,\ldots,a_p\}$. Now replace the a_i by mutually disjoint (except for s and t)

paths P_i of length ℓ. Call the resulting directed graph G' and let $A_{-1} = \cup_{i=1}^{p} A(P_i)$. Finally, define $w(a) = 1$ for $a \in A_{+1}$ and $w(a) = -1$ for $a \in A_{-1}$. As $A(G') = A_{-1} \uplus A_{+1}$, the function $w : A(G') \to \{-1, +1\}$ is well defined.

The instance (G', w, k) has the required form. Also the transformation can be made in polynomial time. As k remains unchanged and $w_- = |A_{-1}| = \ell \cdot p \leq \ell \cdot (k+1)$ is bounded by a polynomial in $k + \ell$, the parameter restrictions of PPTs are fulfilled. It remains to prove that (G', w, k) is a "yes"-instance of Negative DFAS if and only if (G, k, ℓ) is a "yes"-instance of Directed Small Cycle Transversal.

For the forward direction, let X' be a solution of (G', w, k). Let X be the set where every arc of X' which is part of some P_i is replaced by a_i (and duplicates are removed). Clearly, $|X| \leq |X'| \leq k$. Suppose there is a cycle C of length at most ℓ in $G - X$. Then C contains a unique arc a_j. Let C' be the cycle in G' resulting from the replacement of a_j by P_j in C. Then C' is also in $G' - X'$ by choice of X and contains at most $\ell - 1$ arcs in A_{+1} and ℓ arcs in A_{-1}. This yields the contradiction $w(C') = |A(C') \cap A_{+1}| - |A(C') \cap A_{-1}| \leq \ell - 1 - \ell = -1$.

For the reverse direction, let X be a solution of (G, k, ℓ). Let X' the set X where every arc a_i is replaced by the first arc of P_i. By definition of G', $X' \subseteq A(G')$ and $|X'| = |X| \leq k$ holds. Now suppose there is a cycle C' in $G' - X'$ with $w(C') < 0$. As the paths P_i are mutually disjoint (except the end vertices) every inner vertex of each P_i has in-degree and out-degree one. Thus, if there is an arc of some P_i inside C' the whole path P_i is. Replace each such P_i by a_i to obtain a cycle C. By construction of G' and X', this cycle C is in $G - X$. Each cycle in G and therefore also C contains exactly one arc of type a_i. Therefore, C' contains exactly one path P_i and from $0 > w(C') = |A(C') \cap A_{+1}| - |A(C') \cap A_{-1}| = |A(C') \cap A_{+1}| - \ell$ we get that $|A(C') \cap A_{+1}| < \ell$. As $|A(C') \cap A_{+1}|$ is integral we can sharpen the bound to $|A(C') \cap A_{+1}| \leq \ell - 1$. By $A(C) = (A(C') \cap A_{+1}) \uplus \{a_i\}$, we get that $|A(C)| = |A(C') \cap A_{+1}| + 1 \leq \ell$ – a contradiction. ◀

▶ **Theorem 12.** *The* Negative DFAS *problem and the* MinFB *problem for difference constraints are parameter-equivalent. Additionally, the equivalence can be constructed such that there is a one-to-one correspondence between constraints and arc weights with* $b_a = w(a)$.

Proof. Let (G, w, k) be a Negative DFAS problem instance, and let $n = |V(G)|$ and $m = |A(G)|$. Fix an arbitrary order $v_1, \ldots, v_n$ of the vertices of G and $\alpha_1, \ldots, \alpha_m$ of the arcs of G. As matrix A we take the incidence matrix of G which is defined as matrix $A = (a_{i,j}) \in \mathbb{R}^{m \times n}$ with entries $a_{i,j} = +1$ for $\alpha_i \in \delta^-(v_j)$, $a_{i,j} = -1$ for $\alpha_i \in \delta^+(v_j)$, and $a_{i,j} = 0$ otherwise. Furthermore, let $b_i = w(\alpha_i)$. The resulting tuple (A, b, k) is an instance of the MinFB problem with A being a matrix of difference constraints.

The construction is bijective by the following reverse construction: Define a directed graph on n vertices $v_1, \ldots, v_n$, then for every constraint $a_{i,\bullet} \cdot x \leq b_i$ add an arc as follows: Let j^- be the unique index with $a_{i,j^-} = -1$, and let j^+ be the unique index with $a_{i,j^+} = +1$. Add an arc $\alpha = (v_{j^+}, v_{j^-})$ with weight $w(\alpha) = b_i$ to the current digraph. Let G be the resulting digraph after all arcs are added. Then (G, w, k) is the constructed Negative DFAS instance. It is easy to verify that this indeed reverses the first construction.

Now we want to compare solutions of both problems. Intuitively, deleted constraints and arcs have an one to one correspondence, but we will formally prove the equivalence here.

For this we need the notion of "feasible potentials". A *feasible potential* (with respect to G and w) is a function $\pi : V(G) \to \mathbb{Q}$ such that, for every arc $a = (x, y) \in A(G)$, the following inequality holds: $w(a) - \pi(x) + \pi(y) \geq 0$. It is well-known that a weighted digraph has a feasible potential if and only if it has no cycle of negative weight (see, for example, the book of Schrijver [34, Theorem 8.2]).

In the following, for each $X \subseteq A(G)$ denote by $X_{\mathcal{I}}$ the corresponding indices of the constraints and vice-versa. Then the following equivalences hold:

$(G - X, w)$ contains no negative cycles with respect to w.
$\Leftrightarrow$ $(G - X, w)$ has a feasible potential $\pi : V(G) \to \mathbb{Q}$.
$\Leftrightarrow$ There is some $\pi : V(G) \to \mathbb{Q}$ such that $\pi(u) \leq \pi(v) + w(\alpha)$ for all $\alpha = (u, v) \in A(G) \setminus X$.
$\Leftrightarrow$ There is some $x \in \mathbb{Q}^{V(G)}$ such that $x_u - x_v \leq w(\alpha)$ for all $\alpha = (u, v) \in A(G) \setminus X$.
$\Leftrightarrow$ There is some $x \in \mathbb{Q}^n$ such that $a_{i,\bullet} \cdot x \leq b_i$ for all $i \in \{1, \ldots, m\} \setminus X_{\mathcal{I}}$.

Furthermore, as X and $X_{\mathcal{I}}$ have the same cardinality, the last statement is equivalent to the statement that X is a solution to (G, w, k) if and only if $X_{\mathcal{I}}$ is a solution to (A, b, k). ◀

Concatenating all reductions above, we obtain the following corollary:

▶ **Corollary 13.** *There is a PPT from* Bounded Edge Directed (s,t)-Cut *parameterized by k (in $k + \ell$) to* MinFB *parameterized by k (in $k + b_-$). This even holds for instances of* MinFB *where A is a system of difference constraints and $b \in \{\pm 1\}^m$.*

This corollary yields our two hardness results:

Proof of Theorem 1. With Proposition 7 we can use the W[1]-hardness of Bounded Edge Directed (s,t)-Cut from Proposition 8 and the PPT from Corollary 13 to get the W[1]-hardness of MinFB. Also the structure of the instance follows from this. ◀

Proof of Theorem 4. This follows by combining Proposition 9 and Corollary 13 with the help of Proposition 7. The structure of the MinFB instance follows as in Theorem 1. ◀

4 Fixed-parameter Algorithms for Systems of Difference Constraints

In this section we develop fixed-parameter algorithm for MinFB for constraint matrices A of difference constraints and right-hand sides $b \in \{\pm 1\}^m$. Our first algorithm takes as parameters the number k of constraints that must be deleted from A to make the system feasible and the number w_- of negative entries in the b-vector; our second algorithm takes as parameters k and the number w_+ of positive entries in b. The naming convention for w_+, w_- stems from Theorem 12 and the 1-to-1 correspondence between b and w.

Recall that the DFAS problem corresponds to the case when $w_+ = 0$, as in this case all arcs have negative weight. In fact, our algorithm makes oracle calls to an algorithm for the more general problem Subset DFAS, in which we are given a digraph G, an arc set $U \subseteq A(G)$ and an integer k, and seek a set $X \subseteq A(G)$ of at most k arcs that intersects each cycle containing some arc of U. The Subset DFAS problem was shown to be fixed-parameter tractable for parameter k by Chitnis et al. [14].

▶ **Proposition 14** ([14]). Subset DFAS *is solvable in time $2^{\mathcal{O}(k^3)} \cdot n^{\mathcal{O}(1)}$.*

For both algorithms we need a subroutine finding a shortest negative cycle C in a given digraph. Recall that *shortest* is defined in terms of number of arcs. Negative cycles can be found with the Moore-Bellman-Ford algorithm (cf. Bang-Jensen and Gutin [3, Sect. 2.3.4]), which runs in time $\mathcal{O}(nm)$. That algorithm can be modified to find a shortest negative cycle of length at most ℓ in time $\mathcal{O}(\ell nm)$; such modification is well-known or at least an easy exercise.

▶ **Lemma 15** ($\star$). *There is an algorithm that, given a digraph G, arc weights $w : A(G) \to \mathbb{Q}$ and an integer $\ell \in \mathbb{N}$, in time $\mathcal{O}(\ell nm)$ either finds a shortest negative cycle C of length at most ℓ in G, or decides that none exists.*

Algorithm 1 SimpleNegativeCycleDeletion.

Input : A digraph G, arc weights $w : A(G) \to \mathbb{Q}$ and $k \in \mathbb{N}$.
Output : A set $S \subseteq A(G)$ of at most k arcs such that $G - S$ has no negative cycles or `false` if no such set exists.

```
1 for every k_-, k_+ ∈ Z_{≥0} with k_- + k_+ = k do
2     for every subset S_- of w^{-1}(-1) ⊆ A(G) with |S_-| = k_- do
3         S_0 = S_-.
4         while G - S_0 contains a negative cycle C and |S_0| ≤ k do
5             Branch on adding an arc a ∈ A(C) with w(a) = 1 to S_0.
6         if |S_0| ≤ k then
7             return S_0.
8 return false.
```

4.1 Few negative right-hand sides

We are now ready to give our fixed-parameter algorithm for MinFB for constraint matrices A of difference constraints and right-hand sides $b \in \{\pm 1\}^m$, parameterized by the size k of the deletion set and the number b_- of negative entries in b. For the rest of this subsection we will use Theorem 12 and only work on the Negative DFAS problem with $w : A(G) \to \{\pm 1\}$. We give a simple algorithm for the Negative DFAS problem with $w : A(G) \to \{\pm 1\}$, parameterized by k and b_-. Pseudocode of the algorithm can be found in Algorithm 1. The algorithm first guesses the negative arcs in the solution and then recursively branches on the positive arcs of a negative cycle (as long as such a cycle exists). The algorithm keeps track of the already deleted arcs in the set S_0.

▶ **Lemma 16.** *Algorithm 1 is correct and runs in time* $(k+1)w_-^{k+1}\mathcal{O}(nm)$.

Proof. The algorithm works in three steps: The first **for** loop guesses how many of the deleted arcs have weight -1 with the variable k_-. The second **for** loop then iterates over every (k_-)-element subset of these arcs. The last procedure then tries to fix the negative cycle by only deleting arcs of weight $+1$.

As we enumerate all choices of k_- and the subsets of negative arcs we only need to argue correctness for the last procedure. The procedure only returns a value other than "false" if this value is a solution. So we only need to argue that if there is a solution we will find it. So let S be a solution and S_0 contain all k_- arcs of weight -1 in S. We get S_0 by correct guessing of the **for** loops. If $S_0 = S$, then $|S_0| \leq k$ and the graph $G - S_0$ will contain no negative cycle, so we correctly return S_0. Otherwise, there is a negative cycle C in $G - S_0$. By choice of S_0 there must be an arc $a \in A(C) \cap S$ with weight $+1$. Branching on all possible choices in line 5, one of the branches must have found the right arc and added it to S_0. Thus, in each recursive call we find an additional element of S until $S_0 = S$.

For the runtime, the first main observation to be made is that any negative cycle C has length at most $2w_- - 1$. Furthermore, at most $w_- - 1$ arcs of it can have weight $+1$. Therefore, we can check by Lemma 15 for the existence of a negative cycle in time $\mathcal{O}(w_- nm)$ and iterate over all arcs with weight $+1$ in a cycle in time $\mathcal{O}(w_-)$. As S_0's size increases by one with each branching and we stop (correctly) if $|S_0| > k$ at most $k - k_-$ recursive calls are made by the branching. Thus, the runtime of the inner branching procedure for fixed S_- is at most $(w_-)^{k-k_-+1}\mathcal{O}(nm)$. The inner **for** loop enumerates, for a fixed value of k_-, at most $w_-^{k_-}$ sets S_-. This **for** loop is executed $k+1$ many times by the outer **for** loop. So the algorithm runs in time $(k+1)w_-^{k_-} \cdot (w_-)^{k-k_-+1}\mathcal{O}(nm) = (k+1)(w_-)^{k+1}\mathcal{O}(nm)$. ◀

Proof of Theorem 3. By Theorem 12 we can construct, in polynomial time, an instance of Negative DFAS that has the same parameters $k + w_-$, and such that the original instances is a "yes"-instance if and only if the constructed instance is. Lemma 16 then allows us to solve this instance in the claimed time. ◀

4.2 Few positive right-hand sides

In the second part of this section we will study Negative DFAS with weight functions $w : A(G) \to \{\pm 1\}$ when parameterized by k and w_+. The main observation for our algorithm is made in the following lemma:

▶ **Lemma 17** (⋆). *Let G be a digraph with arc weights $w : A(G) \to \{\pm 1\}$. Then either G has a negative cycle of length at most $2(w_+)^2 + 2w_+$, or every negative cycle C has some arc $a \in A(C)$ that lies only on negative cycles of G.*

This lemma forms the basis of our algorithm, Algorithm 2. First, the algorithm checks for negative cycles with up to $2(w_+)^2 + 2w_+$ arcs. It then guesses the arc contained in a solution like in our first algorithm, Algorithm 1. Afterwards, we are left with a digraph without small cycles. We now identify the set U of arcs which are not part of a non-negative cycle. Then, for any solution S to Negative DFAS, $G - S$ may not contain a cycle on which an arc of U lies, as such a cycle would be negative by definition of U. Likewise, any negative cycle in G has some arc in U by the previous lemma. Thus, it remains to solve an instance of Subset DFAS for input (G, U, k).

Algorithm 2 NegativeCycleDeletion.

Input : A digraph G with arc weights $w : A(G) \to \mathbb{Q}$ and $k \in \mathbb{N}$.
Output : A set $S \subseteq A(G)$ of at most k arcs such that $G - S$ has no negative cycle, or `false` if no such set exists.

```
if k < 0 then
    return false.
if there is some negative cycle C of length at most 2(w_+)^2 + 2w_+ in G then
    Branch on deleting an arc of C and try to solve with k − 1.
else
    Identify the set U of all arcs which do not lie on a non-negative cycle.
    return SubsetDirectedFeedbackArcSet (G, U, k).
```

Before we can prove correctness and runtime we have to show how we can detect the set U of all arcs which lie only on negative cycles. We first argue that this problem is NP-hard even for weights $w : A(G) \to \{\pm 1\}$. To this end, we provide a reduction from the Hamiltonian s-t-Path problem, which for a digraph H and vertices $s, t \in V(H)$ asks for an s-t-path in H visiting each vertex of H exactly once. Its NP-hardness was shown by Karp [26]. The reduction works as follows: Take the original digraph H and two vertices $s, t \in V(H)$ which we want to test for the existence of an Hamiltonian path starting in s and ending in t. Add a path P of length $n - 1$ from t to s to the graph. Assign weight $+1$ to each arc of H, and weight -1 to each arc of P. Then an arc of P lies on a cycle of non-negative length if and only if there is an Hamiltonian s-t-path in H.

However, for this construction of weights w we have $w_+ \in \Omega(n)$. We will now show that the task is indeed fixed-parameter tractable when parameterized by w_+. For that, the main observation is that every non-negative cycle has length at most $2w_+$. We now consider the

Weighted ℓ-Path problem: given a digraph G with arc weights $w : A(G) \to \mathbb{Q}$ and numbers $W \in \mathbb{Q}, \ell \in \mathbb{N}$, the task is to find a path of length exactly ℓ and weight at least W in G. Zehavi [36] gave a fast algorithm for Weighted ℓ-Path, based on color-coding techniques.

▶ **Proposition 18** ([36]). Weighted ℓ-Path *can be solved in time* $2^{\mathcal{O}(\ell)} \cdot \mathcal{O}(m \log n)$ *on digraphs with* n *vertices and* m *arcs.*

So given an arc $a = (s, t)$ one can enumerate all path sizes ℓ from 1 to $2w_+ - 1$ and ask whether there is a t-s-path of length ℓ of weight at least $-w(a)$. This way one can detect a non-negative cycle containing a.

▶ **Corollary 19.** *Let* G *be a digraph, let* $w : A(G) \to \{\pm 1\}$ *and* $(s, t) \in A(G)$*. Then one can detect in time* $2^{\mathcal{O}(w_+)} \cdot m \log n$ *if* $a = (s, t)$ *is part of some non-negative cycle* C*.*

Finally, to argue the correctness and runtime of Algorithm 2, we prove the following:

▶ **Lemma 20** ($\star$). *Algorithm 2 is correct and runs in time* $2^{\mathcal{O}(k^3 + w_+ + k \log w_+)} \cdot n^{\mathcal{O}(1)}$.

Proof of Theorem 2. By Theorem 12 we can construct, in polynomial time, an instance of Negative DFAS that has the same parameters $k + w_-$, and such that the original instances is a "yes"-instance if and only if the constructed instance is. Lemma 20 then allows us to solve this instance in the claimed time. Regarding the run time, note that $m \leq (k+1)n^2 \in 2^{\mathcal{O}(\log k)} \cdot n^{\mathcal{O}(1)}$. ◀

5 NP-Hardness for Incidence Matrices of Constant Pathwidth

In this section we show that MinFB is NP-hard even for constraint matrices A whose pathwidth is bounded by 6. By this we mean that the non-parameterized variant of MinFB (where k is part of the input) is NP-hard. To this end, we reduce Partition to Negative DFAS in digraphs whose underlying undirected graph has pathwidth at most 6. Recall that Partition is the problem of finding, in a set $\mathcal{A} = \{a_1, \ldots, a_n\}$ of positive integers, a subset $\mathcal{A}'$ so that $\sum_{a_i \in \mathcal{A}'} a_i = \sum_{a_i \in \mathcal{A} \setminus \mathcal{A}'} a_i$ or decide that no such set exists. Equivalently, let $A = \sum_{i=1}^{n} a_i$ and reformulate the Partition problem as that of finding a subset $\mathcal{A}'$ such that $\mathcal{A}'$ and $\mathcal{A} \setminus \mathcal{A}'$ each sum up to $\frac{A}{2}$. Karp [26] showed that Partition is NP-complete.

Starting from an instance $\mathcal{A} = \{a_1, \ldots, a_n\} \in \mathbb{N}^n$ of Partition, we now construct a Negative DFAS instance consisting of a digraph G with arc weights $w : A(G) \to \mathbb{Z}$ and some $k \in \mathbb{N}$. Afterwards, we argue why (G, w, k) has a solution if and only if $\mathcal{A}$ has one.

For every number $a_i \in \mathcal{A}$ construct a gadget G_i as follows (see Fig. 1): Let $V^{(i)} = \{s_i^{(j)}, x_i^{(j)}, y_i^{(j)}, z_i^{(j)}, t_i^{(j)} \mid j = 0, 1\}$ be the vertex set of G_i. We have three different kinds of arcs forming the arc set: the first arc set $A_1^{(i)} = \{(x_i^{(j)}, y_i^{(j)}), (y_i^{(j)}, z_i^{(j)}) \mid j = 0, 1\}$ contains the arcs we will consider for deletion later. The arc weight of all arcs $a \in A_1^{(i)}$ is 0.

The second arc set $A_2^{(i)} = \{(z_i^{(j)}, x_i^{(j)}), (y_i^{(j)}, x_i^{(1-j)}), (z_i^{(j)}, y_i^{(1-j)})) \mid j = 0, 1\}$ enforces the deletion of arcs form the first arc set by inducing negative cycles. Also, these arcs are the only arcs between vertices with different superscripts. The weight of each arc $a \in A_2^{(i)}$ is -1.

Finally, the arcs $A_3^{(i)} = \{(s_i^{(j)}, x_i^{(j)}), (s_i^{(j)}, y_i^{(j)}), (y_i^{(j)}, t_i^{(j)}), (z_i^{(j)}, t_i^{(j)}) \mid j = 0, 1\}$ connect the vertices $s_i^{(j)}$ and $t_i^{(j)}$ to the rest of the graph. The arc weights are as follows:

$$w\left((s_i^{(j)}, x_i^{(j)})\right) = 0, \qquad w\left((s_i^{(j)}, y_i^{(j)})\right) = A + 1 + a_i,$$
$$w\left((z_i^{(j)}, t_i^{(j)})\right) = 0, \qquad w\left((y_i^{(j)}, t_i^{(j)})\right) = A + 1 \ .$$

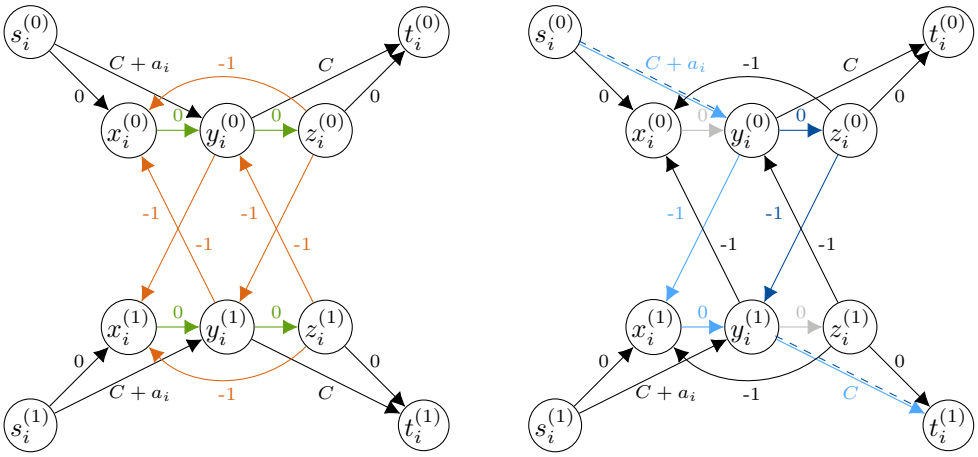


Figure 1 The gadget graph G_i (left) and paths through G_i after removal of a solution $S_i^{(p)}$ (right).

The whole gadget G_i is then defined as $(V^{(i)}, A_1^{(i)} \cup A_2^{(i)} \cup A_3^{(i)})$. The proof of soundness of the reduction and the path decomposition of G of width 6 is omitted from this version of the paper due to space constraints.

Overall, we get that Negative DFAS is NP-hard in graphs of pathwidth 6. Applying Theorem 12 to this result completes the proof of Theorem 5.

6 Discussion

We considered the fundamental MinFB problem from the perspective of parameterized complexity. Our results include a general parameterized intractability result (W[1]-hardness) even for totally unimodular matrices and parameter solution size, as well as fixed-parameter algorithms for totally unimodular matrices when the additional parameter of number of positive/negative right-hand sides is taken into account. It would be interesting to know whether the run times of our algorithms can be improved to $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.

We also ruled out the existence of a polynomial compression for combined parameter $k + w_+$, assuming that coNP $\not\subseteq$ NP/*poly*. It remains a challenging open problem whether DFAS admits a polynomial compression for parameter k, and whether our perspective from the more general MinFB problem can help with that.

References

1 Edoardo Amaldi and Viggo Kann. On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theoret. Comput. Sci.*, 209(1-2):237–260, 1998.

2 Sanjeev Arora, László Babai, Jacques Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. *J. Comput. System Sci.*, 54(2, part 2):317–331, 1997.

3 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

4 Piotr Berman and Marek Karpinski. Approximating Minimum Unsatisfiability of Linear Equations. In *Proc. SODA 2002*, pages 514–516, 2002.

5 Hans L. Bodlaender, Fedor V. Fomin, and Saket Saurabh. Open problems from 2010 Workshop on Kernels, 2010. URL: `http://fpt.wdfiles.com/local--files/open-problems/open-problems.pdf`.

6 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization Lower Bounds by Cross-Composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014.

7 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theoret. Comput. Sci.*, 412(35):4570–4578, 2011.

8 Marthe Bonamy, Łukasz Kowalik, Jesper Nederlof, Michał Pilipczuk, Arkadiusz Socała, and Marcin Wrochna. On Directed Feedback Vertex Set Parameterized by Treewidth. In *Proc. WG 2018*, pages 65–78, 2018.

9 Nilotpal Chakravarti. Some results concerning post-infeasibility analysis. *European J. Oper. Res.*, 73(1):139–143, 1994.

10 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):Art. 21, 19, 2008.

11 John W. Chinneck. An effective polynomial-time heuristic for the minimum-cardinality IIS set-covering problem. *Ann. Math. Artif. Intell.*, 17(1):127–144, 1996.

12 John W. Chinneck. Finding a Useful Subset of Constraints for Analysis in an Infeasible Linear Program. *INFORMS J. Comput.*, 9(2):164–174, 1997.

13 John W. Chinneck. *Feasibility and infeasibility in optimization: algorithms and computational methods*, volume 118 of *International Series in Operations Research & Management Science*. Springer, New York, 2008.

14 Rajesh Chitnis, Marek Cygan, Mohammataghi Hajiaghayi, and Dániel Marx. Directed subset feedback vertex set is fixed-parameter tractable. *ACM Trans. Algorithms*, 11(4):Art. 28, 28, 2015.

15 Gianni Codato and Matteo Fischetti. Combinatorial Benders' cuts for mixed-integer linear programming. *Oper. Res.*, 54(4):756–766, 2006.

16 Robert Crowston, Gregory Gutin, Mark Jones, and Anders Yeo. Parameterized complexity of satisfying almost all linear equations over $\mathbb{F}_2$. *Theory Comput. Syst.*, 52(4):719–728, 2013.

17 Marek Cygan, Fedor Fomin, Bart M.P. Jansen, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Open Problems collected for the 2014 School on Parameterized Complexity in Będlewo, Poland, 2014. URL: `http://fptschool.mimuw.edu.pl/opl.pdf`.

18 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.

19 Marek Cygan, Łukasz Kowalik, and Marcin Pilipczuk. Open problems from 2013 Workshop on Kernels, 2013. URL: `http://worker2013.mimuw.edu.pl/slides/worker-opl.pdf`.

20 Till Fluschnik, Danny Hermelin, André Nichterlein, and Rolf Niedermeier. Fractals for Kernelization Lower Bounds, With an Application to Length-Bounded Cut Problems. In *Proc. ICALP 2016*, volume 55 of *Leibniz Int. Proc. Informatics*, pages 25:1–25:14, 2016.

21 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, MichałPilipczuk, and Marcin Wrochna. Fully Polynomial-Time Parameterized Computations for Graphs and Matrices of Low Treewidth. *ACM Trans. Algorithms*, 14(3):34:1–34:45, 2018.

22 Panos Giannopoulos, Christian Knauer, and Günter Rote. The parameterized complexity of some geometric problems in unbounded dimension. In *Proc. IWPEC 2009*, volume 5917 of *Lecture Notes Comput. Sci.*, pages 198–209. 2009.

23 Petr A. Golovach and Dimitrios M. Thilikos. Paths of bounded length and their cuts: Parameterized complexity and algorithms. *Discrete Optim.*, 8(1):72–86, 2011.

24 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, second edition, 1993.

25 Sylvain Guillemot. FPT algorithms for path-transversal and cycle-transversal problems. *Discrete Optim.*, 8(1):61–71, 2011.

26 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM, Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 85–103. Plenum, New York, 1972.

27 Stefan Kratsch and Magnus Wahlström. Two edge modification problems without polynomial kernels. *Discrete Optim.*, 10(3):193–199, 2013.

28 Neele Leithäuser, Sven O. Krumke, and Maximilian Merkert. Approximating infeasible 2VPI-systems. In *Proc. WG 2012*, volume 7551 of *Lecture Notes Comput. Sci.*, pages 225–236. 2012.

29 Daniel Lokshtanov, M. S. Ramanuajn, and Saket Saurabh. When Recursion is Better Than Iteration: A Linear-time Algorithm for Acyclicity with Few Error Vertices. In *Proc. SODA 2018*, pages 1916–1933, 2018.

30 Matthias Mnich and Erik Jan van Leeuwen. Polynomial kernels for deletion to classes of acyclic digraphs. *Discrete Optim.*, 25:48–76, 2017.

31 Marc E. Pfetsch. Branch-and-cut for the maximum feasible subsystem problem. *SIAM J. Optim.*, 19(1):21–38, 2008.

32 G. Ramalingam, J. Song, L. Joskowicz, and R. E. Miller. Solving Systems of Difference Constraints Incrementally. *Algorithmica*, 23(3):261–275, 1999.

33 Jayaram K. Sankaran. A note on resolving infeasibility in linear programs by constraint relaxation. *Oper. Res. Lett.*, 13(1):19–20, 1993.

34 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

35 K. Subramani and Piotr Wojciechowski. A Combinatorial Certifying Algorithm for Linear Feasibility in UTVPI Constraints. *Algorithmica*, 78(1):166–208, 2017.

36 Meirav Zehavi. Mixing color coding-related techniques. In *Proc. ESA 2015*, volume 9294 of *Lecture Notes Comput. Sci.*, pages 1037–1049. 2015.

Clustering to Given Connectivities

Petr A. Golovach
Department of Informatics, University of Bergen, Norway
petr.golovach@uib.no

Dimitrios M. Thilikos
AlGCo project-team, LIRMM, Université de Montpellier, CNRS, Montpellier, France
sedthilk@thilikos.info

Abstract

We define a general variant of the graph clustering problem where the criterion of density for the clusters is (high) connectivity. In Clustering to Given Connectivities, we are given an n-vertex graph G, an integer k, and a sequence $\Lambda = \langle \lambda_1, \ldots, \lambda_t \rangle$ of positive integers and we ask whether it is possible to remove at most k edges from G such that the resulting connected components are *exactly* t and their corresponding edge connectivities are lower-bounded by the numbers in Λ. We prove that this problem, parameterized by k, is fixed parameter tractable, i.e., can be solved by an $f(k) \cdot n^{O(1)}$-step algorithm, for some function f that depends only on the parameter k. Our algorithm uses the recursive understanding technique that is especially adapted so to deal with the fact that we do not impose any restriction to the connectivity demands in Λ.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases graph clustering, edge connectivity, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.18

Related Version A full version of the paper is available at `https://arxiv.org/abs/1803.09483`.

Funding *Petr A. Golovach*: Supported by the Research Council of Norway via the projects CLASSIS and MULTIVAL. Supported by the Research Council of Norway and the French Ministry of Europe and Foreign Affairs, via the Franco-Norwegian project PHC AURORA 2019.
Dimitrios M. Thilikos: Supported by projects DEMOGRAPH (ANR-16-CE40-0028) and ESIGMA (ANR-17-CE23-0010). Supported by the Research Council of Norway and the French Ministry of Europe and Foreign Affairs, via the Franco-Norwegian project PHC AURORA 2019.

1 Introduction

Clustering deals with grouping the elements of a data set based on some similarity measure between them. As a general computational procedure, clustering is fundamental in several scientific fields including machine learning, information retrieval, bioinformatics, data compression, and pattern recognition (see [7, 68, 70]). In many such applications, data sets are organized and/or represented by graphs that naturally express relations between entities. A *graph clustering problem* asks for a partition of the vertices of a graph into vertex sets, called *clusters*, so that each cluster enjoys some desirable characteristics of "density" or "good interconnectivity", while having few edges between the clusters (see [12, 63] for related surveys).

Parameterizations of graph clustering problems. As a general problem on graphs, graph clustering has many variants. Most of them depend on the *density* criterion that is imposed on the clusters and, in most of the cases, they are NP-complete. However, in many real-world instances, one may expect that the number of edges between clusters is much smaller than the size of the graph. This initiated the research for parameterized algorithms for graph clustering problems. Here the aim is to investigate when the problem is FPT (Fixed Parameter

Tractable), when parameterized by the number k of edges between clusters, i.e. it admits a $f(k) \cdot n^{O(1)}$ step algorithm, also called FPT-algorithm (see [21, 27, 30, 59] for textbooks on parameterized algorithms and the corresponding parameterized complexity class hierarchy). More general parameterizations may also involve k edit operations to the desired cluster property.

In the most strict sense, one may demand that all vertices in a cluster are pair-wise connected, i.e., they form a clique. This corresponds to the Cluster Deletion problem and its more general version Cluster Editing where we ask for the minimum edge additions or deletions that can transform a graph to a collection of cliques. Cluster Editing was introduced by Ben-Dor, Shamir, and Yakhini in [6] in the context of computational biology and, independently, by Bansal, Blum, and Chawla [5] motivated by machine learning problems related to document clustering (see also [65]). Algorithmic research on these problems and their variants is extensive, see [1–3, 28, 35, 65]. Moreover their standard parameterizations are FPT and there is a long list of improvements on the running times of the corresponding FPT-algorithms [10, 11, 13, 14, 17, 18, 29, 39, 41, 62].

In most practical cases, in a good clustering, it is not necessary that clusters induce cliques. This gives rise to several difference measures of density or connectivity. In this direction, Heggernes et al., in [46], introduced the (p, q)-Cluster Graph Recognition problem where clusters are cliques that may miss at most p edges (also called γ-quasi cliques) [60, 61]). This problem was generalized in [56], where, given a function μ and a parameter p, each cluster C should satisfy $\mu(C) \leq p$, and was proved to be FTP for several instantiations of μ. In [47], Hüffner et al. introduced the Highly Connected Deletion problem, where each cluster C should induce a highly connected graph (i.e., have edge connectivity bigger than $|C|/2$ – see also [45,48]) and proved that this problem is FPT. Algorithmic improvements and variants of this problem where recently studied by Bliznets and Karpov in [9]. In [42], Guo et al. studied the problems s-Defective Clique Editing, Average-s-Plex Delection, and μ-Clique Deletion where each cluster S is demanded to be a clique missing s edges, a graph of average degree at least $|C| - s$, or a graph with average density s, respectively (the two first variants are FPT, while this is not expected for the last one). In [64] clusters are of diameter at most s (s-clubs), in [4, 43, 58, 67] every vertex of a cluster should have an edge to all but at most $s - 1$ other vertices of it (s-plexes). In [32], Fomin et al. considered the case where the number of clusters to be obtained is *exactly* p and proved that this version is also in FPT. Other Parameterizations of Cluster Editing where introduced and shown to be FPT in [8, 15, 25, 53, 69]).

Our results. Here we adopt connectivity as a general density criterion for the clusters (following the line of [9, 48]). We study a general variant of graph clustering where we prespecify both the number of clusters (as done in [32]) but also the connectivities of the graphs induced by them. Actually we consider the edge weighted version of the problem where the weighted edge connectivity $\lambda^{\mathsf{w}}(G)$ of an edge weighted graph G is defined as the minimum weight of an edge cut (see Section 2 for formal definitions).

Clustering to Given Weighted Connectivities (CGWC)

Input: A weighted graph G with an edge weight function $w\colon E(G) \to \mathbb{N}$, a t-tuple $\Lambda = \langle \lambda_1, \ldots, \lambda_t \rangle$, where $\lambda_i \in \mathbb{N} \cup \{+\infty\}$ for $i \in \{1, \ldots, t\}$ and $\lambda_1 \leq \ldots \leq \lambda_t$, and a nonnegative integer k.

Task: Decide whether there is a set $F \subseteq E(G)$ with $\mathsf{w}(F) \leq k$ such that $G - F$ has t connected components $G_1, \ldots, G_t$ where each $\lambda^{\mathsf{w}}(G_i) \geq \lambda_i$ for $i \in \{1, \ldots, t\}$.

(It is convenient to allow $\lambda_i = +\infty$, because we assume that the (weighted) connectivity of the single-vertex graph is $+\infty$.)

The above problem can be seen as a generalization of the well-known t-Cut problem, asking for a partition of a graph into exactly t nonempty components such that the total number of edges between the components is at most k. Indeed, this problem is CGWC for unit weights and $\lambda_1 = \ldots = \lambda_t = 1$. As it was observed by Goldschmidt and Hochbaum in [36], t-Cut is NP-hard if t is a part of the input. This immediately implies the NP-hardness of CGWC. Therefore, we are interested in the parameterized complexity of the problem. The main results of Goldschmidt and Hochbaum [36] is that t-Cut can be solved in time $\mathcal{O}(n^{t^2})$, that is, the problem is polynomial for any *fixed* t. In other words, t-Cut belongs in the parameterized complexity class XP when parameterized by t. This results was shown to be tight in the sense that we cannot expect an FPT algorithm for this problem unless the basic conjectures of the Parameterized Complexity theory fail by Downey et al. who proved in [26] that the problem is W[1]-hard when parameterized by t. The situation changes if we parameterize the problem by k. By the celebrated result of Kawarabayashi and Thorup [49], t-Cut is FPT when parameterized by k.

In this paper, we prove that CGWC is FPT when parameterized by k. For our proofs we follow the *recursive understanding* technique introduced by Chitnis et al. [19] (see also [40]) combined with the *random separation* technique introduced by Cai, Chan and Chan in [16]. Already in [19], Chitnis et al. demonstrated that this technique is a powerful tool for the design of FPT-algorithms for various cut problems. This technique was further developed by Cygan et al. in [23] for proving that the Minimum Bisection problem is FPT (see also [33, 37, 52, 57] for other recent applications of this technique on the design of FPT-algorithms). Nevertheless, we stress that for CGWC the application for the recursive understanding technique becomes quite nonstandard and demands additional work due to the fact that neither t nor the connectivity constraints $\lambda_1, \ldots, \lambda_t$ are restricted by any constant or any function of the parameter k (we stress that the general meta-algorithmic framework of [57] is not directly applicable to our problem and in the conclusion section (Section 5) we provide some discussion on how to tackle this). Towards dealing with the diverse connectivities, we deal with special annotated/weighted versions of the problem and introduce adequate connectivity mimicking encodings in order to make recursive understanding possible.

Paper organization. Due to space constraints, we only give high level descriptions of our results. In Section 2, we give definitions that are used throughout the paper. In Section 3, we sketch our algorithm for the basic case where the input graph is connected. For this, we introduce all concepts and results that support the applicability of the recursive understanding technique. We stress that, at this point, the connectivity assumption is important as this makes it easier to control the diverse connectivities of the clusters. In Section 4, we briefly explain how we deal with the general non-connected case. The algorithm in the general case is based on a series of observations on the way connectivities are distributed in the connected components of G. In Section 5, we briefly discuss alternative approaches for CGWC and further directions of research. The details and complete proofs could be found in the full version of the paper [38].

2 Preliminaries

We consider finite undirected simple graphs. We use n to denote the number of vertices and m the number of edges of the considered graphs unless it creates confusion. For disjoint subsets $A, B \subseteq V(G)$, $E(A, B)$ denotes the set of edges with one end-vertex in A and the second in B. A set of edges $S \subseteq E(G)$ of a connected graph G is an *(edge) separator* if $G - S$

is disconnected. For two disjoint subsets $A, B \subseteq V(G)$, $S \subseteq E(G)$ is an (A,B)-separator if $G - S$ has no (u,v)-path with $u \in A$ and $v \in B$. Recall (see, e.g., [24]) that if S is an inclusion minimal (A,B)-separator, then $S = E(A', B')$ for some partition (A', B') of $V(G)$ with $A \subseteq A'$ and $B \subseteq B'$.

Let k be a positive integer. A graph G is *(edge) k-connected* if for every $S \subseteq E(G)$ with $|S| \leq k-1$, $G - S$ is connected, that is, G has no separator of size at most $k-1$. Since we consider only edge connectivity, whenever we say that a graph G is k-connected, we mean that G is edge k-connected. Similarly, whenever we mention a separator, we mean an edge separator.

For technical reasons, it is convenient for us to work with edge weighted graphs. Let G be a graph and let $w\colon E(G) \to \mathbb{N}$ be an (edge) weight function. Whenever we say that G is a weighted graph, it is assumed that an edge weight function is given and we use w to denote weights throughout the paper. For a set of edges S, $\mathsf{w}(S) = \sum_{e \in S} \mathsf{w}(e)$.

For disjoint subsets $A, B \subseteq V(G)$, $w_G(A,B) = \mathsf{w}(E(A,B))$. We say that G is weight k-connected if for every $S \subseteq E(G)$ with $\mathsf{w}(S) \leq k-1$, $G - S$ is connected. We denote by $\lambda^{\mathsf{w}}(G)$ the *weighted connectivity* of G, that is, the maximum value of k such that G is weight k-connected; we assume that every graph is weight 0-connected and for the single-vertex graph G, $\lambda^{\mathsf{w}}(G) = +\infty$. For disjoint subsets $A, B \subseteq V(G)$, $\lambda^{\mathsf{w}}_G(A,B) = \min\{\mathsf{w}(S) \mid S \text{ is an } (A,B)\text{-separator}\}$. We say that an (A,B)-separator S is *minimum* if $\mathsf{w}(S) = \lambda^{\mathsf{w}}_G(A,B)$. For two vertices $u, v \in V(G)$, $\lambda^{\mathsf{w}}(u,v) = \lambda^{\mathsf{w}}(\{u\},\{v\})$ and we assume that $\lambda^{\mathsf{w}}(u,u) = +\infty$. Similarly, for a set A and a vertex v, we write $\lambda^{\mathsf{w}}_G(A,v)$ instead of $\lambda^{\mathsf{w}}_G(A,\{v\})$. Clearly, $\lambda^{\mathsf{w}}(G) = \min\{\lambda^{\mathsf{w}}_G(u,v) \mid u,v \in V(G)\}$. We can omit the subscript if it does not create confusion.

Let $U \subseteq V(G)$. We say that the weighted graph G' is obtained from G by the *weighted contraction* of U if it is constructed as follows: we delete the vertices of U and replace the set by a single vertex u that is made adjacent to every $v \in V(G) \setminus U$ adjacent to a vertex of U and the weight of uv is defined as $\sum_{xv \in E(G),\ x \in U} \mathsf{w}(xv)$. Note that we do not require $G[U]$ be connected. For an edge uv, the weighted contraction of uv is the weighted contraction of the set $\{u,v\}$.

3 Clustering to Given Weighted Connectivities for connected graphs

In this section we show that CGWC is FPT when parameterized by k if the input graph is connected. We prove the following theorem that is used as the main building block for the general case.

▶ **Theorem 1.** *There exist some computable function $f : \mathbb{N} \to \mathbb{N}$, such that* CGWC *can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$ if the input graph is connected.*

The remaining part of the section contains the sketch of the proof of this theorem. In Subsection 3.1 we give some additional definitions and state auxiliary results. Then in Subsection 3.2 we sketch the proof itself.

3.1 Auxiliary results

To solve CGWC for connected graphs, we use the recursive understanding technique introduced by Chitnis et al. in [19]. Therefore, we need notions that are specific to this technique and some results established by Chitnis et al. [19]. Note that we adapt the definitions and the statements of the results for the case of edge weighted graphs.

Weighted good edge separations. Let G be a connected weighted graph with an edge weight function $w\colon E(G) \to \mathbb{N}$. Let also p and q be positive integers. A partition (A, B) of $V(G)$ is called a (q,p)-*good edge separation* if

- $|A| > q$ and $|B| > q$,
- $\mathsf{w}(A, B) \leq p$,
- $G[A]$ and $G[B]$ are connected.

It is said that G is (q,p)-*unbreakable* if for any partition (A, B) of $V(G)$ such that $\mathsf{w}(A, B) \leq p$, it holds that $|A| \leq q$ or $|B| \leq q$.

We use the following variant of Lemma 7 of [19] that is more convenient for our purposes. For the unweighted case, this variant was stated in [31].

▶ **Lemma 2.** *There exists a deterministic algorithm that, given a weighted connected graph G along with positive integers p and q, in time $2^{\mathcal{O}(\min\{p,q\}\log(p+q))} \cdot n^3 \log n$ either finds a (q,p)-good edge separation or correctly concludes that G is (pq,p)-unbreakable.*

Mimicking connectivities by cut reductions. Let r be a nonnegative integer. A pair $(G, \mathbf{x})$, where G is a graph and $\mathbf{x} = \langle x_1, \ldots, x_r\rangle$ is an r-tuple of distinct vertices of G is called an *r-boundaried graph* or simply a *boundaried graph*. Respectively, $\mathbf{x} = \langle x_1, \ldots, x_r\rangle$ is a *boundary*. Note that a boundary is an ordered set. Hence, two r-boundaried graphs that differ only by the order of the vertices in theirs boundaries are distinct. Still, we can treat $\mathbf{x}$ as a set when the ordering is irrelevant. Observe also that a boundary could be empty. Slightly abusing notation, we may say that G is a (r-) boundaried graph assuming that a boundary is given. We say that $(G, \mathbf{x})$ is a *properly* boundaried graph if the vertices of $\mathbf{x}$ are pairwise nonadjacent and each component of G contains at least one vertex of $\mathbf{x}$.

Two r-boundaried weighted graphs $(G_1, \mathbf{x}^{(1)})$ and $(G_2, \mathbf{x}^{(2)})$, where $\mathbf{x}^{(h)} = \langle x_1^{(h)}, \ldots, x_r^{(h)}\rangle$ for $h = 1, 2$, are *isomorphic* if there is an isomorphism of G_1 to G_2 that maps each $x_i^{(1)}$ to $x_i^{(2)}$ for $i \in \{1, \ldots, r\}$ and each edge is mapped to an edge of the same weight.

Let $(G_1, \mathbf{x}^{(1)})$ and $(G_2, \mathbf{x}^{(2)})$ be r-boundaried graphs with $\mathbf{x}^{(h)} = \langle x_1^{(h)}, \ldots, x_r^{(h)}\rangle$ for $h = 1, 2$, and assume that $(G_2, \mathbf{x}^{(2)})$ is a properly boundaried graph. We define the *boundary sum* $(G_1, \mathbf{x}^{(1)}) \oplus_b (G_2, \mathbf{x}^{(2)})$ (or simply $G_1 \oplus_b G_2$) as the (non-boundaried) graph obtained by taking disjoint copies of G_1 and G_2 and identifying $x_i^{(1)}$ and $x_i^{(2)}$ for each $i \in \{1, \ldots, r\}$. Note that the definition is not symmetric as we require that $(G_2, \mathbf{x}^{(2)})$ is a properly boundaried graph and we have no such a restriction for $x_1^{(1)}, \ldots, x_r^{(1)}$.

Let $\mathbf{X} = (X_1, \ldots, X_p)$ and $\mathbf{Y} = (Y_1, \ldots, Y_q)$ be two partitions of a set Z. We define the *product* $\mathbf{X} \times \mathbf{Y}$ of $\mathbf{X}$ and $\mathbf{Y}$ as the partition of Z obtained from $\{X_i \cap Y_j \mid 1 \leq i \leq p,\ 1 \leq j \leq q\}$ by the deletion of empty sets. For partitions $\mathbf{X}^1, \ldots, \mathbf{X}^r$ of Z, we denote their *consecutive product* as $\prod_{i=1}^{r} \mathbf{X}^i$.

Let $(H, \mathbf{x})$ be a connected properly r-boundaried weighted graph. Let p be a positive integer or $+\infty$. Slightly abusing notation we consider here $\mathbf{x}$ as a set. We construct the partition $\mathbf{Z}$ of $V(H)$ as follows. For $X \subseteq \mathbf{x}$, denote $\overline{X} = \mathbf{x} \setminus X$.

- For all distinct pairs $\{X, \overline{X}\}$ for nonempty $X \subset \mathbf{x}$, find a minimum weight $(X, \overline{X})$-separator $S_X = E(Y_X^1, Y_X^2)$ where (Y_X^1, Y_X^2) is a partition of $V(H)$, $X \subseteq Y_X^1$ and $\overline{X} \subseteq Y_X^2$.
- For every $v \in V(H) \setminus \mathbf{x}$, find a minimum weight $(\mathbf{x}, \{v\})$-separator $S(v)$. Find $v^* \in V(H) \setminus \mathbf{x}$ such that $w(S(v^*)) = \min\{w(S(v)) \mid v \in V(H) \setminus \mathbf{x}\}$ and let $S(v^*) = E(Y_{\mathbf{x}}^1, Y_{\mathbf{x}}^2)$ where $(Y_{\mathbf{x}}^1, Y_{\mathbf{x}}^2)$ is a partition of $V(H)$ and $\mathbf{x} \subseteq Y_{\mathbf{x}}^1$.

- Construct the following partition of $V(H)$:

$$\mathbf{Z} = (Z_1, \ldots, Z_h) = \Big(\prod_{\substack{\text{distinct } \{X, \overline{X}\} \\ \varnothing \neq X \subset \mathbf{x}}} (Y_X^1, Y_X^2) \Big) \times (Y_{\mathbf{x}}^1, Y_{\mathbf{x}}^2) \times (\{x_1\}, \ldots, \{x_r\}, V(H) \setminus \mathbf{x}). \quad (1)$$

We construct H' by performing the weighted contraction of the sets of $\mathbf{Z}$. Then for each edge uv of H' with $\mathsf{w}(uv) > p$, we set $\mathsf{w}(uv) = p$, that is, we truncate the weights by p. Notice that because the partition $(\{x_1\}, \ldots, \{x_r\}, V(H) \setminus \mathbf{x})$ is participating the product defining $\mathbf{Z}$, we have that $\{x\} \in \mathbf{Z}$ for each $x \in \mathbf{x}$, that is, the elements of the boundary are not contracted, and this is the only purpose of this partition in (1). We say that H' is obtained from $(H, \mathbf{x})$ by the *cut reduction with respect to* p. Note that H' is not unique as the construction depends on the choice of separators. It could be observed that we construct a *mimicking network* representing cuts of H [44, 50] (see also [51]).

We extend this definition for disconnected graphs. Let $(H, \mathbf{x})$ be a properly r-boundaried weighted graph and let p be a positive integer or $+\infty$. Denote by $H_1, \ldots, H_s$ the components of H and let $\mathbf{x}^i = \mathbf{x} \cap V(H_i)$ for $i \in \{1, \ldots, s\}$. Consider the boundaried graphs $(H_i', \mathbf{x}^i)$ obtained from $(H_i, \mathbf{x})$ by cut reduction with respect to p for $i \in \{1, \ldots, s\}$. We say that $(H', \mathbf{x})$, that is obtained by taking the union of $(H_i', \mathbf{x}^i)$ for $i \in \{1, \ldots, s\}$, is obtained by the *cut reduction with respect to* p.

The crucial property of H' is that it keeps the separators of H that are essential for the connectivity.

▶ **Lemma 3.** *Let $(H, \mathbf{x})$ be a properly r-boundaried weighted graph, and let $p \in \mathbb{N} \cup \{+\infty\}$ and $t \in \mathbb{N}$. Let also $(F, \mathbf{y})$ be an r-boundaried weighted graph, and let $G = (F, \mathbf{y}) \oplus_b (H, \mathbf{x})$. Then for an r-boundaried weighted graph H' obtained from H by the cut reduction with respect to p and t positive integers $\lambda_1, \ldots, \lambda_t \leq p$ it holds that G has exactly t components and they have the connectivities $\lambda_1, \ldots, \lambda_t$ respectively if and only if the same holds for $G' = (F, \mathbf{y}) \oplus_b (H', \mathbf{x})$, that is, G' has t components and they have the connectivities $\lambda_1, \ldots, \lambda_t$ respectively.*

It is also important to observe that it holds that $|V(H')| \leq 2^{2^{r-1}} + r$, that is, the size of an r-boundaried weighted graph obtained by cut reduction is bounded by a function of r.

For positive integers r and s, we define $\mathcal{H}_{r,s}$ as the family of all pairwise nonisomorphic properly r-boundaried weighted graphs $(G, \mathbf{x})$ with at most $2^{2^{r-1}} + r$ vertices where the weights of edges are in $\{1, \ldots, s\}$ and for every component C of G with $V(C) \setminus \mathbf{x} \neq \varnothing$, there is a vertex $v \in V(C) \setminus \mathbf{x}$ such that $\lambda^{\mathsf{w}}(\mathbf{x}, v) \leq s$. We also formally define $\mathcal{H}_{0,s}$ as the set containing the empty graph. Note that $|\mathcal{H}_{r,s}| \leq (s+1)^{\binom{2^{2^{r-1}}}{2} + r}$ and $\mathcal{H}_{r,s}$ can be constructed in time $2^{2^{2^{\mathcal{O}(r)}}} \log s$.

Variants of CGWC. To apply the recursive understanding technique, we also have to solve a special variant of CGWC tailored for recursion. To define it, we first introduce the following variant of the problem. The difference is that a solution should be chosen from a given subset of edges.

Annotated CGWC

Input: A weighted graph G with an edge weight function $w\colon E(G) \to \mathbb{N}$, $L \subseteq E(G)$, a t-tuple $\Lambda = \langle \lambda_1, \ldots, \lambda_t \rangle$, where $\lambda_i \in \mathbb{N} \cup \{+\infty\}$ for $i \in \{1, \ldots, t\}$ and $\lambda_1 \leq \ldots \leq \lambda_t$, and a nonnegative integer k.

Task: Decide whether there is a set $F \subseteq L$ with $\mathsf{w}(F) \leq k$ such that $G - F$ has t connected components $G_1, \ldots, G_t$ where each $\lambda^{\mathsf{w}}(G_i) \geq \lambda_i$ for $i \in \{1, \ldots, t\}$.

Clearly, if $L = E(G)$, then Annotated CGWC is CGWC. Let (G, w, L, Λ, k) be an instance of Annotated CGWC. We say that $F \subseteq L$ with $\mathsf{w}(F) \leq k$ such that $G - F$ has t connected components $G_1, \ldots, G_t$ where each $\lambda^{\mathsf{w}}(G_i) \geq \lambda_i$ for $i \in \{1, \ldots, t\}$ is a *solution* for the instance.

Now we define Border A-CGWC.

Border A-CGWC

Input: A weighted r-boundaried connected graph $(G, \mathbf{x})$ with an edge weight function $w\colon E(G) \to \mathbb{N}$, $L \subseteq E(G)$, a t-tuple $\Lambda = \langle \lambda_1, \ldots, \lambda_t \rangle$, where $\lambda_i \in \mathbb{N} \cup \{+\infty\}$ for $i \in \{1, \ldots, t\}$ and $\lambda_1 \leq \ldots \leq \lambda_t$, and a nonnegative integer k such that $r \leq 4k$ and $k \geq t - 1$.

Task: For each weighted properly r-boundaried graph $(H, \mathbf{y}) \in \mathcal{H}_{r,2k}$ and each $\hat{\Lambda} = \langle \hat{\lambda}_1, \ldots, \hat{\lambda}_s \rangle \subseteq \Lambda$, find the minimum $0 \leq \hat{k} \leq k$ such that $((G, \mathbf{x}) \oplus_b (H, \mathbf{y}), w, L, \hat{\Lambda}, \hat{k})$ is a yes-instance of Annotated CGWC and output a solution F for this instance or output $\varnothing$ if $\hat{k}$ does not exist.

Slightly abusing notation, we use w to denote the weights of edges of G and H. Notice that Border A-CGWC is neither decision nor optimization problem, and its solution is a list of subsets of L. Observe also that a solution of Border A-CGWC is not necessarily unique. Still, for any two solutions, that is, lists $\mathcal{L}_1$ and $\mathcal{L}_2$ of subsets of L, the following holds: for each weighted properly r-boundaried graph $(H, \mathbf{y}) \in \mathcal{H}_{r,2k}$ and each $\hat{\Lambda} = \langle \hat{\lambda}_1, \ldots, \hat{\lambda}_s \rangle \subseteq \Lambda$, the lists $\mathcal{L}_1$ and $\mathcal{L}_2$ contain the sets of the same weight. To solve Annotated CGWC, it is sufficient to solve Border A-CGWC for $r = 0$. If the output contains nonempty set for $\hat{\Lambda} = \Lambda$, we have a yes-instance of Annotated CGWC. If the output contains empty set for this $\hat{\Lambda}$, we should verify additionally whether $\varnothing$ is a solution, that is, whether $\Lambda = \{\lambda_1\}$ and $\lambda^{\mathsf{w}}(G) \geq \lambda_1$. To apply the recursive understanding technique, we first solve Border A-CGWC for $(q, 2k)$-unbreakable graphs for some appropriate value of q and then use this result for the general case of Border A-CGWC.

Restricted BFS subgraphs. Let G be a graph, $u \in V(G)$, and let r be a positive integer. We construct the subgraph $B_r(u)$ using a modified breadth-first search algorithm. Recall that in the standard breadth-first search algorithm (see, e.g., [20]) starting from u, we first label u by $\ell(u) = 0$ and put u into a queue Q. Then we iterate as follows: if Q is nonempty, then take the first vertex x in the queue and for every nonlabeled neighbor y, assign $\ell(y) = \ell(x) + 1$ and put y into Q. We start in the same way by assigning u the label $\ell(u) = 0$ and putting u into Q. Then while Q is nonempty and the first element x has the label $\ell(x) \leq r - 1$, we consider arbitrary chosen $\min\{r, d_G(x)\}$ vertices $y \in N_G(x)$, assign to unlabeled vertices y the label $\ell(y) = \ell(x) + 1$ and put them into Q. The graph $B_r(u)$ is the subgraph of G induced by the labeled vertices v with $\ell(v) \leq r$. We say that $B_r(x)$ is an *r-restricted BFS subgraph* of G. Note that such a subgraph is not unique.

3.2 Sketch of the proof of Theorem 1

First, we construct an algorithm for Border A-CGWC for connected $(q, 2k)$-unbreakable graphs. The crucial step is to solve Annotated CGWC.

▶ **Lemma 4.** Annotated CGWC *can be solved and a solution can be found (if exists) in time* $2^{\mathcal{O}(q(q+k)\log(q+k))} \cdot n^{\mathcal{O}(1)}$ *for connected* $(q, 2k)$*-unbreakable graphs.*

Sketch of the proof. Let (G, w, L, Λ, k) be an instance of Annotated CGWC where G is a connected $(q, 2k)$-unbreakable graph. Let also $\Lambda = \langle \lambda_1, \ldots, \lambda_t \rangle$, $\lambda_1 \leq \ldots \leq \lambda_t$. Clearly, the problem is easy if $t = 1$ and the problem is trivial if $t > k + 1$, because a connected graph G can be separated into at most $k + 1$ components by at most k edge deletions. Let $2 \leq t \leq k + 1$. If $|V(G)| \leq 3q$, we solve Annotated CGWC using brute force. From now we assume that $|V(G)| > 3q$.

Suppose that (G, w, L, Λ, k) is a yes-instance of Annotated CGWC and let $F \subseteq L$ be a solution. Let $G_1, \ldots, G_t$ be the components of $G - F$ and $\lambda^{\mathsf{w}}(G_i) \geq \lambda_i$ for $i \in \{1, \ldots, t\}$. Using that G is a $(q, 2k)$-unbreakable graph, we show that there is a component G_i with at least $q + 1$ vertices and the total number of vertices in the other components is at most q. We say that G_i is a *big* components and call the other components *small*. For each $i \in \{1, \ldots, t\}$, we verify whether there is a solution F where λ_i is the connectivity constraint for the big component of $G - F$.

Assume that $\lambda_i > k$. We show that in this case $V(G_i)$ is an inclusion maximal set of vertices X of G with the property that for every two vertices $u, v \in X$, $\lambda^{\mathsf{w}}_G(u, v) \geq k + 1$. We use this to find the big component and then find other clusters by brute force.

From now we assume that $\lambda_i \leq k$. To deal with this case we apply the *random separation* technique introduces by Cai, Chan and Chan in [16]. To avoid dealing with randomized algorithms, we use the Lemma 1 of [19]. Assume again that (G, w, L, Λ, k) is a yes-instance of Annotated CGWC, $F \subseteq L$ is a solution, and $G_1, \ldots, G_t$ are the components of $G - F$ where G_i is the big component. Let $A = \bigcup_{j \in \{1,\ldots,t\} \setminus \{i\}} V(G_j)$. Recall that $|A| \leq q$. Let also $X \subseteq V(G_i)$ be the set of vertices of G_i that have neighbors in A. Note that $|X| \leq k$. For each $u \in X$, we consider a $(q + \lambda_i)$-restricted BFS subgraph $B(u) = B_{q+\lambda_i}(u)$ of G_i. Let $B = \bigcup_{u \in X} V(B(u))$. We have that $|V(B(u))| = 2^{\mathcal{O}((q+k)\log(q+k))}$ since $\lambda_i \leq k$. Hence, $|B| = 2^{\mathcal{O}((q+k)\log(q+k))}$. Note also that $|B| \geq q + 1$. We say that a set $S \subseteq V(G)$ is (A, B)-*good* or, simply, *good* if $B \subseteq S$ and $A \cap S = \emptyset$. By Lemma 1 of [19], we can construct in time $2^{\mathcal{O}(q(q+k)\log(q+k))} \cdot n \log n$ a family $\mathcal{S}$ of at most $2^{\mathcal{O}(q(q+k)\log(q+k))} \cdot \log n$ subsets of $V(G)$ such that if (G, w, L, Λ, k) is a yes-instance and A and B exist for some solution, then $\mathcal{S}$ contains an (A, B)-good set.

We construct such a family $\mathcal{S}$, and for each $S \in \mathcal{S}$, we look for $F \subseteq L$ such that the following holds:

- **(i)** $\mathsf{w}(F) \leq k$,
- **(ii)** $G - F$ has t components $G_1, \ldots, G_t$ such that each G_j is weight λ_j-connected and $|V(G_i)| > q$, and
- **(iii)** $S \subseteq V(G_i)$.

We describe the algorithm that produces the YES answer if S is good and, moreover, if the algorithm outputs YES, then (G, w, L, Λ, k) is a yes-instance of Annotated CGWC. Note that the algorithm can output the false NO answer if S is not a good set. Nevertheless, because for an yes-instance of Annotated CGWC, we always have a good set $S \in \mathcal{S}$, we have that (G, w, L, Λ, k) is a yes-instance if and only if the algorithm outputs YES for at least one $S \in \mathcal{S}$.

The algorithm uses the following property of (A, B)-good sets.

▷ Claim (A). *If S is an (A, B)-good set, then for each component H of $G - S$, either $V(H) \subseteq V(G_i)$ or $V(H) \cap V(G_i) = \emptyset$.*

We apply a number of reduction rules that either increase the set S or conclude that S is not good and stop. Each rule increasing S is applied exhaustively. For each rule, we show

that it is *safe* in the sense that if we increase S, then if the original S was good, then the obtained set is good as well, and if we conclude that the original S is not good, then this is a correct conclusion and, therefore, we can return NO and stop. We underline that whenever we return NO in the rules, this means that we discard the current choice of S.

Due to the size of B, we get the following rule.

▶ **Reduction Rule 3.1.** If $|S| \leq q$, then return NO and stop.

Denote by $H_1, \ldots, H_s$ the components of $G - S$. Applying Claim A we obtain the next rules.

▶ **Reduction Rule 3.2.** For every $j \in \{1, \ldots, s\}$, if $E(V(H_j), S) \setminus L \neq \emptyset$ or $\mathsf{w}(V(H_j), S) \geq k+1$ or $|V(H_j)| > q$, then set $S = S \cup V(H_j)$.

▶ **Reduction Rule 3.3.** If there is $u \in S$ adjacent to a vertex of H_j for some $j \in \{1, \ldots, s\}$ and there is a connected set $Z \subseteq S$ such that a) $u \in Z$, b) $|Z| \leq q$, c) $\mathsf{w}(Z, S \setminus Z) \leq \lambda_i - 1$, then set $S = S \cup V(H_j)$.

To apply the last rule, we use the result of Fomin and Villanger [34] that allows to list all the sets Z satisfying a)–c) in time $2^{\mathcal{O}(k \log(q+k))} \cdot n$ because $\lambda_i \leq k$. We apply Reduction Rule 3.3 recomputing the components of $G - S$ after each modification of S. Then we again use the result of Fomin and Villanger [34] to apply the next rule.

▶ **Reduction Rule 3.4.** If there is a connected set $Z \subseteq S$ such that $|Z| \leq q$ and $\mathsf{w}(Z, V(G) \setminus Z) \leq \lambda_i - 1$, then set return NO and stop.

Assume that we do not stop while executing Reduction Rule 3.4. We use the flowing claim to identify the components of $G - S$ whose vertices, definitely, are not in the big cluster.

▷ Claim (B). *If S is an (A, B)-good set, then if for a component H of $G - S$, there is $v \in V(H)$ such that $\lambda^{\mathsf{w}}(v, S) < \lambda_i$, then $V(H) \cap V(G_i) = \emptyset$.*

Let $H_1, \ldots, H_s$ be the components of $G - S$. We set

$$I = \{j \in \{1, \ldots, s\} \mid \text{there is } v \in V(H_j) \text{ such that } \mathsf{w}(v, S) < \lambda_i\}.$$

▶ **Reduction Rule 3.5.** If $|\bigcup_{j \in I} V(H_j)| > q$ or $\mathsf{w}(\bigcup_{j \in I} V(H_j), S) > k$, then return NO and stop.

▷ Claim (C). *For any $J \subseteq \{1, \ldots, s\}$ such that $I \subseteq J$ and $\mathsf{w}(\bigcup_{j \in J} V(H_j), S) \leq k$, the graph $G' = G - \bigcup_{j \in J} V(H_j)$ is weight λ_i-connected.*

By applying Reduction Rules 3.1–3.5, we either increase S or stop. Now we have to find an $F \subseteq L$ such that (i)–(iii) are fulfilled and, by applying Claims A and B, we impose two additional constraints:

- **(iv)** for every $j \in \{1, \ldots, s\} \setminus I$, either $V(H_j) \subseteq V(G_i)$ or $V(H_j) \cap V(G_i) = \emptyset$,
- **(v)** for every $j \in I$, $V(H_j) \cap V(G_i) = \emptyset$.

Note that by Claim C, we automatically obtain that $\lambda^{\mathsf{w}}(G_i) \geq \lambda_i$ if (i), (iii)-(v) are fulfilled. Also because of Reduction Rule 3.1, we have that $|V(G_i)| > q$ if (iii) holds. Hence, we can replace (ii) by the relaxed condition:

- **(ii)** $G - F$ has t components $G_1, \ldots, G_t$ such that G_j is weight λ_j-connected for $j \in \{1, \ldots, t\}$, $j \neq i$.

We find F, if such a set exists, by a dynamic programming algorithm that sorts the vertices of $G - S$ staring with the components with indices in I. ◀

Using Lemma 4, we construct the algorithm for Border A-CGWC for connected $(q, 2k)$-unbreakable graphs.

▶ **Lemma 5.** Border A-CGWC *can be solved in time* $2^{q^3 2^{2^{\mathcal{O}(k)}}} \cdot n^{\mathcal{O}(1)}$ *for connected* $(q, 2k)$*-unbreakable graphs.*

Now we construct an algorithm for Border A-CGWC and this result implies Theorem 1.

▶ **Lemma 6.** Border A-CGWC *can be solved in time* $f(k) \cdot n^{\mathcal{O}(1)}$.

Sketch of the proof. We construct a recursive algorithm for Border A-CGWC. Let $(G, \mathbf{x}, w, L, \Lambda, k)$ be an instance of Border A-CGWC. Recall that $(G, \mathbf{x})$ is an r-boundaried connected graph and $r \leq k$. Recall also that Λ contains at most $k+1$ elements.

We define the constants p and q that are used throughout the proof as follows:

$$p = 2k2^{k+1}(2k+1)^{\left(2^{2^{4k-1}}+4k\right)} + 4k \text{ and } q = 2^{2^{p-1}} + p. \tag{2}$$

We are going to use q as a part of the unbreakability threshold.

We apply Lemma 2 and in time $2^{\mathcal{O}(k\log(q+k))} \cdot n^3 \log n$ either find a $(q, 2k)$-good edge separation (A, B) of G or conclude that G is $(2kq, 2k)$-unbreakable.

If G is $(2kq, 2k)$-unbreakable, we apply Lemma 5 and solve the problem in time $2^{q^3 2^{2^{\mathcal{O}(k)}}} \cdot n^{\mathcal{O}(1)}$. Assume from now that we are given a $(q, 2k)$-good edge separation (A, B) of G.

Since $|\mathbf{x}| \leq 4k$ and the vertices of $\mathbf{x}$ are separated between A and B, either A or B contains at most $2k$ vertices of $\mathbf{x}$. Assume without loss of generality that $|A \cap \mathbf{x}| \leq 2k$. Let T be the set of end-vertices of the edges of $E(A, B)$ in A. Clearly, $|T| \leq 2k$. We form a new $\hat{r}$-tuple $\hat{\mathbf{x}} = \langle \hat{x}_1, \ldots, \hat{x}_{\hat{r}} \rangle$ of vertices A from the vertices of $(A \cap \mathbf{x}) \cup T$; note that $\hat{r} \leq 4k$. We consider $\hat{G} = G[A]$ as the $\hat{\mathbf{x}}$-boundaried graph. We set $\hat{L} = L \cap E(\hat{G})$. This way, we obtain the instance $(\hat{G}, \hat{\mathbf{x}}, w, \hat{L}, \Lambda, k)$ of Border A-CGWC.

Now we solve Border A-CGWC for $(\hat{G}, \hat{\mathbf{x}}, w, \hat{L}, \Lambda, k)$.

If $|V(\hat{G})| \leq 2^q$, we can simply use brute force. If $|V(\hat{G})| > 2^q$, we recursively solve Border A-CGWC for $(\hat{G}, \hat{\mathbf{x}}, w, \hat{L}, \Lambda, k)$ by calling our algorithm for the instance that has lesser size, because $|V(\hat{G})| \leq |V(G)| - q$.

By solving Border A-CGWC for $(\hat{G}, \hat{\mathbf{x}}, \hat{L}, \Lambda, k)$, we obtain a list $\mathcal{L}$ of sets where each element is either $\emptyset$ or $F \subseteq \hat{L}$ that is a solution for the corresponding instance of Annotated CGWC for some $(H, \mathbf{y}) \in \mathcal{H}_{\hat{r},2k}$, $\hat{\Lambda} \subseteq \Lambda$ and $\hat{k} \leq k$. Denote by M the union of all the sets in $\mathcal{L}$. Clearly, $M \subseteq \hat{L}$.

We define $L^* = (L \setminus \hat{L}) \cup M$. Since $M \subseteq \hat{L}$, $L^* \subseteq L$. We show that the instances $(G, \mathbf{x}, w, L, \Lambda, k)$ and $(G, \mathbf{x}, w, L^*, \Lambda, k)$ are essentially equivalent by proving the following claim by making use of Lemma 3.

▷ Claim (A). *For every weighted properly r-boundaried graph $(H, \mathbf{y}) \in \mathcal{H}_{r,2k}$, every $\hat{\Lambda} = \langle \hat{\lambda}_1, \ldots, \hat{\lambda}_s \rangle \subseteq \Lambda$ and every nonnegative integer $\hat{k} \leq k$, $((G, \mathbf{x}) \oplus_b (H, \mathbf{y}), w, L, \hat{\Lambda}, \hat{k})$ is a yes-instance of* Annotated CGWC *if and only if $((G, \mathbf{x}) \oplus_b (H, \mathbf{y}), w, L^*, \hat{\Lambda}, \hat{k})$ is a yes-instance of* Annotated CGWC.

By Claim A we obtain that every solution of $(G, \mathbf{x}, w, L^*, \Lambda, k)$ is a solution of $(G, \mathbf{x}, w, L, \Lambda, k)$, and there is a solution of $(G, \mathbf{x}, w, L, \Lambda, k)$ that is a solution of $(G, \mathbf{x}, w, L^*, \Lambda, k)$. Therefore, it is sufficient for us to solve $(G, \mathbf{x}, w, L^*, \Lambda, k)$.

Let $Z \subseteq A$ be the set of end-vertices of the edges of M and the vertices of $\hat{\mathbf{x}}$. Because $\hat{r} \leq 4k$, $\mathcal{H}_{\hat{r},2k} \leq (2k+1)^{\left(2^{2^{4k-1}}+4k\right)}$. Since $t \leq k+1$, there are at most 2^{k+1} subtuples $\hat{\Lambda}$ of Λ.

For each $(H,y) \in \mathcal{H}_{\hat{r},2k}$ and $\hat{\Lambda} \subseteq \Lambda$, the solution $\mathcal{L}$ of Border A-CGWC for $(\hat{G},\hat{\mathbf{x}},\hat{L},\Lambda,k)$ contains a set F of size at most k. This implies that

$$|M| \leq k2^{k+1}(2k+1)^{\left(2^{2^{4k-1}}+4k\right)}.$$

Because $|\hat{\mathbf{x}}| \leq 4k$, we obtain that

$$|Z| \leq 2|M|+4k \leq 2k2^{k+1}(2k+1)^{\left(2^{2^{4k-1}}+4k\right)}+4k = p \tag{3}$$

for p defined in (2).

Let $U = A \setminus Z$. We define $Q = G - U$. Let also R be the graph with the vertex set A and the edge set $E(G[A]) \setminus E(G[Z])$. We order the vertices of Z arbitrarily and consider Z to be $|Z|$-tuple of the vertices of Q and R. Observe that (R,Z) is a properly $|Z|$-boundaried graph as $G[A]$ is connected. Since $V(F) \cap V(R) = Z$, we have that $G = (Q,Z) \oplus_b (R,Z)$. Let (R^*,Z) be the boundaried graph obtained from (R,Z) by the cut reduction with respect to $+\infty$. We define $G^* = (Q,Z) \oplus_b (R^*,Z)$. Note that $L^* \subseteq E(Q) \subseteq E(G^*)$. We show that we can replace G by G^* in the considered instance $(G,\mathbf{x},w,L^*,\Lambda,k)$ of Border A-CGWC by making use of Lemma 3.

▷ Claim (B). *For every weighted properly r-boundaried graph $(H,\mathbf{y}) \in \mathcal{H}_{r,2k}$, every $\hat{\Lambda} = \langle \hat{\lambda}_1,\ldots,\hat{\lambda}_s\rangle \subseteq \Lambda$ and every nonnegative integer $\hat{k} \leq k$, a set $F \subseteq L^*$ is a solution for the instance $((G,\mathbf{x}) \oplus_b (H,\mathbf{y}),w,L^*,\hat{\Lambda},\hat{k})$ if and only if F is a solution for $((G^*,\mathbf{x}) \oplus_b (H,\mathbf{y}),w,L^*,\hat{\Lambda},\hat{k})$.*

By Claim B, to solve Border A-CGWC for $(G,\mathbf{x},w,L^*,\Lambda,k)$, it is sufficient to solve the problem for $(G^*,\mathbf{x},w,L^*,\Lambda,k)$. Observe that $|V(G^*)| = |B| + |V(R^*)|$. Because (R^*,Z) is obtained by the cut reduction, we can show that $|V(R^*)| \leq 2^{2^{|Z|-1}} + |Z|$. Using (3), we have that

$$|V(R^*)| \leq 2^{2^{p-1}} + p = q$$

for q defined in (2). Recall that $|A| > q$ since (A,B) is a $(q,2k)$-good edge separation of G. Therefore,

$$|V(G^*)| = |B| + |V(R^*)| \leq |B| + q < |A| + |B| = |V(G)|.$$

We use this and solve Border A-CGWC for $(G^*,\mathbf{x},w,L^*,\Lambda,k)$ recursively by applying our recursive algorithm for the instance with the input graph of smaller size. ◀

4 The algorithm for Clustering to Given Weighted Connectivities

In this section we extend the result obtained in Section 3 and show that CGWC is FPT when parameterized by k even if the input graph is disconnected. Let $\alpha = \langle \alpha_1,\ldots,\alpha_t\rangle$ where $\alpha_i \in \mathbb{N} \cup \{+\infty\}$ for $i \in \{1,\ldots,t\}$ and $\alpha_1 \leq \ldots \leq \alpha_t$. We call the *variate* of α the set of distinct elements of α and denote it by $\mathbf{var}(\alpha)$. Let also $\beta = \langle \beta_1,\ldots,\beta_t\rangle$ where $\beta_i \in \mathbb{N} \cup \{+\infty\}$ for $i \in \{1,\ldots,s\}$ and $\beta_1 \leq \ldots \leq \beta_t$. We write $\alpha \preceq \beta$ if $\alpha_i \leq \beta_i$ for $i \in \{1,\ldots,t\}$.

▶ **Theorem 7.** CGWC *can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$.*

Sketch of the proof. Let (G, w, Λ, k) be an instance CGWC, $\Lambda = \langle \lambda_1, \ldots, \lambda_t \rangle$.

We find the components of G and compute their weighted connectivities using the algorithm of Stoer and Wagner [66] for finding minimum cuts. Assume that $G_1, \ldots, G_s$ are the components of G and $\lambda^{\mathsf{w}}(G_1) \leq \ldots \leq \lambda^{\mathsf{w}}(G_s)$. If $s > t$, then (G, w, Λ, k) is a trivial no-instance. If $s < t - k$, then (G, w, Λ, k) is no-instance, because by deleting at most k edges it is possible to obtain at most k additional components. In all these cases we return the corresponding answer and stop. From now we assume that $t - k \leq s < t$. We exhaustively apply the following reduction rule based on the observation that a component of high connectivity cannot be split.

▶ **Reduction Rule 4.1.** If there is $i \in \{1, \ldots, s\}$ such that $\lambda^{\mathsf{w}}(G_i) > k$, then find the maximum $j \in \{1, \ldots, t\}$ such that $\lambda^{\mathsf{w}}(G_i) \geq \lambda_j$ and set $G = G - V(G_i)$ and $\Lambda = \Lambda \setminus \{\lambda_j\}$.

To simplify notations, assume that G with its components $G_1, \ldots, G_s$ and $\Lambda = \langle \lambda_1, \ldots, \lambda_t \rangle$ is obtained from the original input by the exhaustive application of Reduction Rule 4.1. Note that we still have that $t - k \leq s < t$. It may happen that $s = 0$, that is, G became empty after the application of the rule. In this case (G, w, Λ, k) is a trivial no-instance, and we return NO and stop. Assume that $s \geq 1$. Observe that we obtain that $\lambda^{\mathsf{w}}(G_i) \leq k$ for $i \in \{1, \ldots, s\}$, because Reduction Rule 4.1 cannot be applied any more. If $|\mathbf{var}(\Lambda)| > 3k$, then it could be shown that we have a no-instance of the problem. Respectively, we return NO and stop. Assume that $|\mathbf{var}(\Lambda)| \leq 3k$, that is, the variety of Λ is bounded. We use this to construct the FPT Turing reduction of the problem to the special case when $\lambda^{\mathsf{w}}(C) = \lambda \leq k$ for every component C of the input graph G. For this special case, we solve CGWC by constructing the FPT Turing reduction of CGWC based on Theorem 1 to the Minimum Cost Matching problem that then can be solved in polynomial time by, e.g, the Hungarian method [54, 55]. ◀

5 Conclusion

We proved that Clustering to Given Connectivities is FPT when parameterized by k. We obtained this result by making use of the recursive understanding technique [19]. The drawback of this approach is that the dependence of the running time on the parameter is huge and it seems unlikely that using the same approach one could avoid towers of the exponents similar to the function in Theorem 7. In particular, we do not see how to avoid using mimicking networks (see [44, 50] for the definitions and lower and upper bounds for the size of such networks) whose sizes are double-exponential in the number of terminals.

It can be observed that if we wish to prove just the *existence* of a (non-constructive) FPT-algorithm for CGWC, we can use a slightly different approach based on the meta-algorithmic result of Lokshtanov et al. [57] which applies to problems that can be expressed in Counting Monadic Second Order Logic (CMSOL).

▶ **Proposition 8** ([57]). *Let ψ be a CMSOL sentence. For all $c \in \mathbb{N}$, there exists $s \in \mathbb{N}$ such that if there exists an algorithm that decides whether $G \models \psi$ on (s, c)-unbreakable graphs in time $f(|\psi|) \cdot n^{O(1)}$ then there exists an algorithm that decides whether $G \models \psi$ on general graphs in time $f(|\psi|) \cdot n^{O(1)}$.*

We can use Proposition 8 to obtain a weaker version of Theorem 1 where the function f is not any more computable. Notice that we cannot express in CMSOL the connectivity lower bounds imposed by the t-tuple $\Lambda = \langle \lambda_1, \ldots, \lambda_t \rangle$ directly, because the values of λ_i are not bounded by any function of the parameter k. Hence, we have to go around of this issue. Let

$I = (G, \mathsf{w}, \Lambda, k)$ be an instance of CGWC. We consider a partition $\mathcal{S} = \{S_1, \ldots, S_q\}$ of $V(G)$ such that two vertices x, y belong in the same set if and only $\lambda^{\mathsf{w}}(x, y) \geq k + 1$. Clearly, $\mathcal{S}$ can be computed in polynomial time. A key observation is that if $\lambda^{\mathsf{w}}(G[S_i]) \geq k + 1$, then either $G[S_i]$ is one of the clusters of a solution or $G[S_i]$ is a part of a cluster of a solution whose weighted edge connectivity is at most k. For each $i \in \{1, \ldots, q\}$ where $\lambda^{\mathsf{w}}(G[S_i]) \geq k + 1$, we contract all vertices in S_i to a single vertex and, in the contracted graph G', we assign to this new vertex a weight equal to $\lambda^{\mathsf{w}}(G[S_i])$. We also assign the weight $+\infty$ to the rest of the vertices of G'. Recall now that when G is connected, $\Lambda = \langle \lambda_1, \ldots, \lambda_t \rangle$ has at most $k + 1$ different values for a yes-instance and for each $i \in \{1, \ldots, t\}$ we set up a set R_i that contains all vertices of G' that have weight at least λ_i. We now consider the structure $\alpha = (G', R_1, \ldots, R_t)$ and a generalization of the connected version of CGWC where the input has a structure α instead of a connected graph and where, in the question of the new problem, we additionally demand that $V(G_i) \subseteq R_i, i \in \{1, \ldots, t\}$ and also we ask $\lambda^{\mathsf{w}}(G_i) \geq \lambda_i$ only when $\lambda_i \leq k$, while we demand that $|V(G_i)| = 1$ when $\lambda_i \geq k + 1$. We call this new problem Generalized-CGWC and we observe that $I = (G, \mathsf{w}, \Lambda, k)$ is a yes-instance of CGWC if and only if $I' = ((G', R_1, \ldots, R_t), \mathsf{w}, \Lambda, k)$ is a yes-instance of Generalized-CGWC . It is now possible to verify that Generalized-CGWC can be expressed using CMSOL for every fixed value of k and given Λ as there are no unbounded connectivities to encode. To apply Proposition 8, we have to solve Generalized-CGWC on unbreakable graphs and this can be done similarly to the proof of Lemma 4. Therefore, we can derive the existence of an FPT-algorithm for CGWC on connected graphs and further extend this to general graphs using the reduction of Section 4.

We would like to underline that due to the plug-in of Proposition 8, the alternative approach provided by the above discussion does not provide *any computable function* bounding the parametric dependence of the running time. Under the light of such an alternative, the algorithm described in Section 3 appears to be "less huge" that it might appear by first sight. This is the main reason why we chose to use a more direct approach in our results. In fact Lemma 6 may be seen as a "constructive detour" to Proposition 8.

The natural question would be to ask whether we can get a better running time using different techniques. This question is interesting even for some special cases of CGC when the connectivity constraints are bounded by a constant or are the same for all components. From the other side, it is natural to ask about lower bounds on the running time. For an FPT parameterized problem, it is natural to ask whether it admits a polynomial kernel. We observe that it is unlikely that CGWC has a polynomial kernel even if there are no weights and the maximum connectivity constraint is one, because it was shown by Cygan et al. in [22] that already t-Cut parameterized by the solution size k has no polynomial kernel unless NP $\subseteq$ coNP /poly. Another direction of research is to consider vertex connectivities instead of edge connectivities.

References

1 Nir Ailon, Moses Charikar, and Alantha Newman. Proofs of Conjectures in "Aggregating Inconsistent Information: Ranking and Clustering". Technical Report TR-719-05, Department of Computer Science, Princeton University, USA, 2005.

2 Noga Alon, Konstantin Makarychev, Yury Makarychev, and Assaf Naor. Quadratic forms on graphs. *Inventiones mathematicae*, 163(3):499–522, March 2006.

3 Sanjeev Arora, Eli Berger, Elad Hazan, Guy Kindler, and Muli Safra. On Non-Approximability for Quadratic Programs. In *46th Annual IEEE Symposium on Foundations of Computer*

Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings, pages 206–215, 2005.

4 Balabhaskar Balasundaram, Sergiy Butenko, and Illya V. Hicks. Clique Relaxations in Social Network Analysis: The Maximum k-Plex Problem. *Operations Research*, 59(1):133–142, 2011. doi:10.1287/opre.1100.0851.

5 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation Clustering. *Machine Learning*, 56(1):89–113, July 2004.

6 Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering Gene Expression Patterns. *Journal of Computational Biology*, 6(3/4):281–297, 1999.

7 Pavel Berkhin. A Survey of Clustering Data Mining Techniques. In *Grouping Multidimensional Data - Recent Advances in Clustering*, pages 25–71. Springer, 2006.

8 Nadja Betzler, Jiong Guo, Christian Komusiewicz, and Rolf Niedermeier. Average parameterization and partial kernelization for computing medians. *Journal of Computer and System Sciences*, 77(4):774–789, 2011.

9 Ivan Bliznets and Nikolay Karpov. Parameterized Algorithms for Partitioning Graphs into Highly Connected Clusters. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, pages 6:1–6:14, 2017.

10 S. Böcker, S. Briesemeister, Q.B.A. Bui, and A. Truss. Going weighted: Parameterized algorithms for cluster editing. *Theoretical Computer Science*, 410(52):5467–5480, 2009.

11 Sebastian Böcker. A golden ratio parameterized algorithm for Cluster Editing. *Journal of Discrete Algorithms*, 16:79–89, 2012. Selected papers from the 22nd International Workshop on Combinatorial Algorithms (IWOCA 2011). doi:10.1016/j.jda.2012.04.005.

12 Sebastian Böcker and Jan Baumbach. Cluster Editing. In Paola Bonizzoni, Vasco Brattka, and Benedikt Löwe, editors, *The Nature of Computation. Logic, Algorithms, Applications*, pages 33–44, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

13 Sebastian Böcker, Sebastian Briesemeister, Quang Bao Anh Bui, and Anke Truß. A Fixed-Parameter Approach for Weighted Cluster Editing. In *Proceedings of the 6th Asia-Pacific Bioinformatics Conference, APBC 2008, 14-17 January 2008, Kyoto, Japan*, pages 211–220, 2008.

14 Sebastian Böcker and Peter Damaschke. Even faster parameterized cluster deletion and cluster editing. *Information Processing Letters*, 111(14):717–721, 2011.

15 Hans L. Bodlaender, Michael R. Fellows, Pinar Heggernes, Federico Mancini, Charis Papadopoulos, and Frances Rosamond. Clustering with partial information. *Theoretical Computer Science*, 411(7):1202–1211, 2010.

16 Leizhen Cai, Siu Man Chan, and Siu On Chan. Random Separation: A New Method for Solving Fixed-Cardinality Optimization Problems. In *IWPEC*, volume 4169 of *Lecture Notes in Computer Science*, pages 239–250. Springer, 2006. doi:10.1007/11847250_22.

17 Yixin Cao and Jianer Chen. Cluster Editing: Kernelization Based on Edge Cuts. *Algorithmica*, 64(1):152–169, September 2012.

18 Jianer Chen and Jie Meng. A $2k$ kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012. JCSS Knowledge Representation and Reasoning. doi:10.1016/j.jcss.2011.04.001.

19 Rajesh Chitnis, Marek Cygan, Mohammad Taghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT Algorithms for Cut Problems Using Randomized Contractions. *SIAM J. Comput.*, 45(4):1171–1229, 2016. doi:10.1137/15M1032077.

20 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.

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

22 Marek Cygan, Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Magnus Wahlström. Clique Cover and Graph Separation: New Incompressibility Results. *TOCT*, 6(2):6:1–6:19, 2014.

23 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Minimum bisection is fixed parameter tractable. In *STOC 2014*, pages 323–332. ACM, 2014. doi:10.1145/2591796.2591852.

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

25 Martin Dörnfelder, Jiong Guo, Christian Komusiewicz, and Mathias Weller. On the parameterized complexity of consensus clustering. *Theor. Comput. Sci.*, 542:71–82, 2014.

26 Rodney G. Downey, Vladimir Estivill-Castro, Michael R. Fellows, Elena Prieto, and Frances A. Rosamond. Cutting Up is Hard to Do: the Parameterized Complexity of k-Cut and Related Problems. *Electr. Notes Theor. Comput. Sci.*, 78:209–222, 2003. doi:10.1016/S1571-0661(04)81014-4.

27 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.

28 Jubin Edachery, Arunabha Sen, and Franz J. Brandenburg. Graph Clustering Using Distance-k Cliques. In Jan Kratochvíyl, editor, *Graph Drawing*, pages 98–106, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

29 Michael Fellows, Michael Langston, Frances Rosamond, and Peter Shaw. Efficient Parameterized Preprocessing for Cluster Editing. In Erzsébet Csuhaj-Varjú and Zoltán Ésik, editors, *Fundamentals of Computation Theory*, 2007.

30 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

31 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Spanning Circuits in Regular Matroids. *CoRR*, abs/1607.05516, 2016. arXiv:1607.05516.

32 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of Cluster Editing with a small number of clusters. *J. Comput. Syst. Sci.*, 80(7):1430–1447, 2014.

33 Fedor V. Fomin, Daniel Lokshtanov, Michal Pilipczuk, Saket Saurabh, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1419–1432, 2017.

34 Fedor V. Fomin and Yngve Villanger. Treewidth computation and extremal combinatorics. *Combinatorica*, 32(3):289–308, 2012. doi:10.1007/s00493-012-2536-z.

35 Ioannis Giotis and Venkatesan Guruswami. Correlation Clustering with a Fixed Number of Clusters. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 1167–1176, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1109557.1109686.

36 Olivier Goldschmidt and Dorit S. Hochbaum. A Polynomial Algorithm for the k-cut Problem for Fixed k. *Math. Oper. Res.*, 19(1):24–37, 1994. doi:10.1287/moor.19.1.24.

37 Petr A. Golovach, Pinar Heggernes, Paloma T. Lima, and Pedro Montealegre. Finding Connected Secluded Subgraphs. *CoRR*, abs/1710.10979, 2017. To appear in IPEC 2017. arXiv:1710.10979.

38 Petr A. Golovach and Dimitrios M. Thilikos. Clustering to Given Connectivities. *CoRR*, abs/1803.09483, 2018.

39 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-Modeled Data Clustering: Exact Algorithms for Clique Generation. *Theory of Computing Systems*, 38(4):373–392, July 2005.

40 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, (STOC 2011)*, pages 479–488, 2011. doi:10.1145/1993636.1993700.

41 Jiong Guo. A more effective linear kernelization for cluster editing. *Theoretical Computer Science*, 410(8):718–726, 2009. doi:10.1016/j.tcs.2008.10.021.

42 Jiong Guo, Iyad A. Kanj, Christian Komusiewicz, and Johannes Uhlmann. Editing Graphs into Disjoint Unions of Dense Clusters. *Algorithmica*, 61(4):949–970, 2011.

43 Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. A More Relaxed Model for Graph-Based Data Clustering: s-Plex Cluster Editing. *SIAM J. Discrete Math.*, 24(4):1662–1683, 2010. doi:10.1137/090767285.

44 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing Multiterminal Flow Networks and Computing Flows in Networks of Small Treewidth. *J. Comput. Syst. Sci.*, 57(3):366–375, 1998. doi:10.1006/jcss.1998.1592.

45 Erez Hartuv and Ron Shamir. A clustering algorithm based on graph connectivity. *Inf. Process. Lett.*, 76(4-6):175–181, 2000.

46 Pinar Heggernes, Daniel Lokshtanov, Jesper Nederlof, Christophe Paul, and Jan Arne Telle. Generalized Graph Clustering: Recognizing (p, q)-Cluster Graphs. In *Graph Theoretic Concepts in Computer Science - 36th International Workshop, WG 2010, Zarós, Crete, Greece, June 28-30, 2010 Revised Papers*, pages 171–183, 2010.

47 Falk Hüffner, Christian Komusiewicz, Adrian Liebtrau, and Rolf Niedermeier. Partitioning Biological Networks into Highly Connected Clusters with Maximum Edge Coverage. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 11(3):455–467, 2014.

48 Falk Hüffner, Christian Komusiewicz, and Manuel Sorge. Finding Highly Connected Subgraphs. In *SOFSEM 2015: Theory and Practice of Computer Science - 41st International Conference on Current Trends in Theory and Practice of Computer Science, Pec pod Sněžkou, Czech Republic, January 24-29, 2015. Proceedings*, pages 254–265, 2015.

49 Ken-ichi Kawarabayashi and Mikkel Thorup. The Minimum k-way Cut of Bounded Size is Fixed-Parameter Tractable. In *FOCS 2011*, pages 160–169. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.53.

50 Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Inf. Process. Lett.*, 114(7):365–371, 2014. doi:10.1016/j.ipl.2014.02.011.

51 Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Information Processing Letters*, 114(7):365–371, 2014.

52 Eun Jung Kim, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. Parameterized algorithms for min-max multiway cut and list digraph homomorphism. *J. Comput. Syst. Sci.*, 86:191–206, 2017.

53 Christian Komusiewicz and Johannes Uhlmann. Alternative Parameterizations for Cluster Editing. In *SOFSEM 2011: Theory and Practice of Computer Science - 37th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 22-28, 2011. Proceedings*, pages 344–355, 2011.

54 H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Res. Logist. Quart.*, 2:83–97, 1955. doi:10.1002/nav.3800020109.

55 H. W. Kuhn. Variants of the Hungarian method for assignment problems. *Naval Res. Logist. Quart.*, 3:253–258 (1957), 1956. doi:10.1002/nav.3800030404.

56 Daniel Lokshtanov and Dániel Marx. Clustering with Local Restrictions. *Inf. Comput.*, 222:278–292, January 2013.

57 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO Model Checking to Highly Connected Graphs. In *ICALP 2018*, volume 107 of *LIPIcs*, pages 135:1–135:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

58 Hannes Moser, Rolf Niedermeier, and Manuel Sorge. Exact combinatorial algorithms and experiments for finding maximum k-plexes. *J. Comb. Optim.*, 24(3):347–373, 2012. doi:10.1007/s10878-011-9391-5.

59 Rolf Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

60 Jeffrey Pattillo, Alexander Veremyev, Sergiy Butenko, and Vladimir Boginski. On the maximum quasi-clique problem. *Discrete Applied Mathematics*, 161(1):244–257, 2013. doi:10.1016/j.dam.2012.07.019.

61 Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. On clique relaxation models in network analysis. *European Journal of Operational Research*, 226(1):9–18, 2013. doi:10.1016/j.ejor.2012.10.021.

62 Fábio Protti, Maise Dantas da Silva, and Jayme Luiz Szwarcfiter. Applying Modular Decomposition to Parameterized Cluster Editing Problems. *Theory of Computing Systems*, 44(1):91–104, January 2009.

63 Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. doi:10.1016/j.cosrev.2007.05.001.

64 Shahram Shahinpour and Sergiy Butenko. Distance-Based Clique Relaxations in Networks: s-Clique and s-Club. In Boris I. Goldengorin, Valery A. Kalyagin, and Panos M. Pardalos, editors, *Models, Algorithms, and Technologies for Network Analysis*, pages 149–174, New York, NY, 2013. Springer New York.

65 Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Appl. Math.*, 144(1–2):173–182, 2004. doi:10.1016/j.dam.2004.01.007.

66 Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *J. ACM*, 44(4):585–591, 1997. doi:10.1145/263867.263872.

67 René van Bevern, Hannes Moser, and Rolf Niedermeier. Approximation and Tidying - A Problem Kernel for s-Plex Cluster Vertex Deletion. *Algorithmica*, 62(3-4):930–950, 2012.

68 Ka-Chun Wong. A Short Survey on Data Clustering Algorithms. *CoRR*, abs/1511.09123, 2015. arXiv:1511.09123.

69 Bang Ye Wu and Jia-Fen Chen. Balancing a Complete Signed Graph by Editing Edges and Deleting Nodes. In Ruay-Shiung Chang, Lakhmi C. Jain, and Sheng-Lung Peng, editors, *Advances in Intelligent Systems and Applications - Volume 1*, pages 79–88, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

70 Dongkuan Xu and Yingjie Tian. A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science*, 2(2):165–193, June 2015.

Finding Cuts of Bounded Degree: Complexity, FPT and Exact Algorithms, and Kernelization

Guilherme C. M. Gomes
Universidade Federal de Minas Gerais, Departamento de Ciência da Computação, Belo Horizonte, Brazil
LIRMM, Université de Montpellier, Montpellier, France
gcm.gomes@dcc.ufmg.br

Ignasi Sau
CNRS, LIRMM, Université de Montpellier, Montpellier, France
ignasi.sau@lirmm.fr

Abstract

A *matching cut* is a partition of the vertex set of a graph into two sets A and B such that each vertex has at most one neighbor in the other side of the cut. The Matching Cut problem asks whether a graph has a matching cut, and has been intensively studied in the literature. Motivated by a question posed by Komusiewicz et al. [IPEC 2018], we introduce a natural generalization of this problem, which we call d-Cut: for a positive integer d, a *d-cut* is a bipartition of the vertex set of a graph into two sets A and B such that each vertex has at most d neighbors across the cut. We generalize (and in some cases, improve) a number of results for the Matching Cut problem. Namely, we begin with an NP-hardness reduction for d-Cut on $(2d+2)$-regular graphs and a polynomial algorithm for graphs of maximum degree at most $d+2$. The degree bound in the hardness result is unlikely to be improved, as it would disprove a long-standing conjecture in the context of internal partitions. We then give FPT algorithms for several parameters: the maximum number of edges crossing the cut, treewidth, distance to cluster, and distance to co-cluster. In particular, the treewidth algorithm improves upon the running time of the best known algorithm for Matching Cut. Our main technical contribution, building on the techniques of Komusiewicz et al. [IPEC 2018], is a polynomial kernel for d-Cut for every positive integer d, parameterized by the distance to a cluster graph. We also rule out the existence of polynomial kernels when parameterizing simultaneously by the number of edges crossing the cut, the treewidth, and the maximum degree. Finally, we provide an exact exponential algorithm slightly faster than the naive brute force approach running in time $\mathcal{O}^*(2^n)$.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Matchings and factors

Keywords and phrases matching cut, bounded degree cut, parameterized complexity, FPT algorithm, polynomial kernel, distance to cluster

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.19

Related Version The full version of this article is permanently available at `https://arxiv.org/abs/1905.03134`.

Funding *Guilherme C. M. Gomes*: Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.
Ignasi Sau: Projects DEMOGRAPH (ANR-16-CE40-0028) and ESIGMA (ANR-17-CE23-0010).

1 Introduction

A *cut* of a graph $G = (V, E)$ is a bipartition of its vertex set $V(G)$ into two non-empty sets, denoted by (A, B). The set of all edges with one endpoint in A and the other in B is the *edge cut*, or the set of *crossing edges*, of (A, B). A *matching cut* is a (possibly empty) edge cut that is a matching, that is, such that its edges are pairwise vertex-disjoint. Equivalently,

(A, B) is a matching cut of G if and only if every vertex is incident to at most one crossing edge of (A, B) [7, 15], that is, it has at most one neighbor across the cut.

Motivated by an open question posed by Komusiewicz et al. [18] during the presentation of their article, we investigate a natural generalization that arises from this alternative definition, which we call *d-cut*. Namely, for a positive integer $d \geq 1$, a d-cut is a a cut (A, B) such that each vertex has at most d neighbors across the partition, that is, every vertex in A has at most d neighbors in B, and vice-versa. Note that a 1-cut is a matching cut. As expected, not every graph admits a d-cut, and the d-Cut problem is the problem of deciding, for a fixed integer $d \geq 1$, whether or not an input graph G has a d-cut.

When $d = 1$, we refer to the problem as Matching Cut. Graphs with no matching cut first appeared in Graham's manuscript [15] under the name of *indecomposable graphs*, presenting some examples and properties of decomposable and indecomposable graphs, leaving their recognition as an open problem. In answer to Graham's question, Chvátal [7] proved that the problem is NP-hard for graphs of maximum degree at least four and polynomially solvable for graphs of maximum degree at most three; in fact, as shown by Moshi [24], every graph of maximum degree three and at least eight vertices has a matching cut.

Chvátal's results spurred a lot of research on the complexity of the problem [1,5,18–21,25]. In particular, Bonsma [5] showed that Matching Cut remains NP-hard for planar graphs of maximum degree four and for planar graphs of girth five; Le and Randerath [21] gave an NP-hardness reduction for bipartite graphs of maximum degree four; Le and Le [20] proved that Matching Cut is NP-hard for graphs of diameter at least three, and presented a polynomial-time algorithm for graphs of diameter at most two. Beyond planar graphs, Bonsma's work [5] also proves that the matching cut property is expressible in monadic second order logic and, by Courcelle's Theorem [8], it follows that Matching Cut is FPT when parameterized by the treewidth of the input graph; he concludes with a proof that the problem admits a polynomial-time algorithm for graphs of bounded cliquewidth.

Kratsch and Le [19] noted that Chvátal's original reduction also shows that, unless the Exponential Time Hypothesis [16] (ETH) fails[1], there is no algorithm solving Matching Cut in time $2^{o(n)}$ on n-vertex input graphs. Also in [19], the authors provide a first branching algorithm, running[2] in time $\mathcal{O}^*\left(2^{n/2}\right)$, a single-exponential FPT algorithm when parameterized by the vertex cover number $\tau(G)$, and an algorithm generalizing the polynomial cases of line graphs [24] and claw-free graphs [5]. Kratsch and Le [19] also asked for the existence a single-exponential algorithm parameterized by treewidth. In response, Aravind et al. [1] provided a $\mathcal{O}^*\left(12^{\mathsf{tw}(G)}\right)$ algorithm for Matching Cut using nice tree decompositions, along with FPT algorithms for other structural parameters, namely neighborhood diversity, twin-cover, and distance to split graph.

The natural parameter – the number of edges crossing the cut – has also been considered. Indeed, Marx et al. [23] tackled the Stable Cutset problem, to which Matching Cut can be easily reduced via the line graph, and through a breakthrough technique showed that this problem is FPT when parameterized by the maximum size of the stable cutset. Recently, Komusiewicz et al. [18] improved on the results of Kratsch and Le [19], providing an exact exponential algorithm for Matching Cut running in time $\mathcal{O}^*(1.3803^n)$, as well as FPT algorithms parameterized by the distance to a cluster graph and the distance to a co-cluster graph, which improve the algorithm parameterized by the vertex cover number, since both parameters are easily seen to be smaller than the vertex cover number. For the distance

[1] The ETH states that 3-SAT on n variables cannot be solved in time $2^{o(n)}$; see [16] for more details.
[2] The $\mathcal{O}^*(\cdot)$ notation suppresses factors that are bounded by a polynomial in the input size.

to cluster parameter, they also presented a quadratic kernel; while for a combination of treewidth, maximum degree, and number of crossing edges, they showed that no polynomial kernel exists unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

A problem closely related to d-Cut is that of Internal Partition, first studied by Thomassen [28]. In this problem, we seek a bipartition of the vertices of an input graph such that every vertex has at least as many neighbors in its own part as in the other part. Such a partition is called an *internal partition*. Usually, the problem is posed in a more general form: given functions $a, b : V(G) \to \mathbb{Z}_+$, we seek a bipartition (A, B) of $V(G)$ such that every $v \in A$ satisfies $\mathsf{deg}_A(v) \geq a(v)$ and every $u \in B$ satisfies $\mathsf{deg}_B(u) \geq b(u)$, where $\mathsf{deg}_A(v)$ denotes the number of neighbors of v in the set A. Such a partition is called an (a, b)-*internal partition*. Originally, Thomassen asked in [28] whether for any pair of positive integers s, t, a graph G with $\delta(G) \geq s + t + 1$ has a vertex bipartition (A, B) with $\delta(G[A]) \geq s$ and $\delta(G[B]) \geq t$, where $\delta(H)$ is the minimum degree of H. Stiebitz [27] answered that, in fact, for any graph G and any pair of functions $a, b : V(G) \to \mathbb{Z}_+$ satisfying $\mathsf{deg}(v) \geq a(v) + b(v) + 1$ for every $v \in V(G)$, G has an (a, b)-internal partition; see [17,22] for follow-up results. It is conjectured that, for every positive integer r, there exists some constant n_r for which every r-regular graph with more than n_r vertices has an internal partition [2,10] (the conjecture for r even appeared first in [26]). The cases $r \in \{3, 4\}$ have been settled by Shafique and Dutton [26]; the case $r = 6$ has been verified by Ban and Linial [2]. This latter result implies that every 6-regular graph of sufficiently large size has a 3-cut.

Our results. We aim at generalizing several of the previously reported results for Matching Cut. First, we show in Section 2, by using a reduction inspired by Chvátal's [7], that for every $d \geq 1$, d-Cut is NP-hard even when restricted to $(2d + 2)$-regular graphs and that, if $\Delta(G) \leq d + 2$ (the maximum degree of G) finding a d-cut can be done in polynomial time. The degree bound in the NP-hardness result is unlikely to be improved: if we had an NP-hardness result for d-Cut restricted to $(2d + 1)$-regular graphs, this would disprove the conjecture about the existence of internal partitions on r-regular graphs [2,10,26] for r odd, unless $P = \mathsf{NP}$. We conclude the section by giving a simple exact exponential algorithm that, for every $d \geq 1$, runs in time $\mathcal{O}^*(c_d^n)$ for some constant $c_d < 2$, hence improving over the trivial brute-force algorithm running in time $\mathcal{O}^*(2^n)$.

We then proceed to analyze the problem in terms of its parameterized complexity. Section 3 begins with a proof, using the treewidth reduction technique of Marx et al. [23], that d-Cut is FPT parameterized by the maximum number of edges crossing the cut. Afterwards, we present a dynamic programming algorithm for d-Cut parameterized by treewidth running in time $\mathcal{O}^*\big(2^{\mathsf{tw}(G)}(d+1)^{2\mathsf{tw}(G)}\big)$; in particular, for $d = 1$ this algorithm runs in time $\mathcal{O}^*\big(8^{\mathsf{tw}(G)}\big)$ and improves the one given by Aravind et al. [1] for Matching Cut, which runs in $\mathcal{O}^*\big(12^{\mathsf{tw}(G)}\big)$ time. By employing the cross-composition framework of Bodlaender et al. [4] and using a reduction similar to the one in [18], we show that, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$, there is no polynomial kernel for d-Cut parameterized simultaneously by the number of crossing edges, the maximum degree, and the treewidth of the input graph. We then present a polynomial kernel and an FPT algorithm when parameterizing by the distance to cluster, denoted by $\mathsf{dc}(G)$. This polynomial kernel is our main technical contribution, and it is strongly inspired by the technique presented by Komusiewicz et al. [18] for Matching Cut. Finally, we give an FPT algorithm parameterized by the distance to co-cluster, denoted by $\mathsf{d\overline{c}}(G)$. These results imply the existence of a polynomial kernel for d-Cut parameterized by the vertex cover number $\tau(G)$. We present in Section 4 our concluding remarks and some open questions.

We use standard notation from graph theory and parameterized complexity; see [9,11–13] for any undefined terminology. Due to space limitations, the proofs of the results marked with '(⋆)' can be found in the full version of this article, permanently available at `https://arxiv.org/abs/1905.03134`. Some basic preliminaries can also be found there.

1.1 Preliminaries

We use standard graph-theoretic notation, and we consider simple undirected graphs without loops or multiple edges; see [11] for any undefined terminology. When the graph is clear from the context, the degree (that is, the number of neighbors) of a vertex v is denoted by $\mathsf{deg}(v)$, and the number of neighbors of a vertex v in a set $A \subseteq V(G)$ is denoted by $\mathsf{deg}_A(v)$. The minimum degree, the maximum degree, the line graph, and the vertex cover number of a graph G are denoted by $\delta(G)$, $\Delta(G)$, $L(G)$, and $\tau(G)$, respectively. For a positive integer $k \geq 1$, we denote by $[k]$ the set containing every integer i such that $1 \leq i \leq k$.

We refer the reader to [9,12] for basic background on parameterized complexity, and we recall here only some basic definitions. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$. For an instance $I = (x, k) \in \Sigma^* \times \mathbb{N}$, k is called the *parameter*. A parameterized problem is *fixed-parameter tractable* (FPT) if there exists an algorithm $\mathcal{A}$, a computable function f, and a constant c such that given an instance $I = (x, k)$, $\mathcal{A}$ (called an FPT *algorithm*) correctly decides whether $I \in L$ in time bounded by $f(k) \cdot |I|^c$.

A fundamental concept in parameterized complexity is that of *kernelization*; see [13] for a recent book on the topic. A kernelization algorithm, or just *kernel*, for a parameterized problem Π takes an instance (x, k) of the problem and, in time polynomial in $|x| + k$, outputs an instance (x', k') such that $|x'|, k' \leqslant g(k)$ for some function g, and $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$. The function g is called the *size* of the kernel and may be viewed as a measure of the "compressibility" of a problem using polynomial-time preprocessing rules. A kernel is called *polynomial* (resp. *quadratic, linear*) if the function $g(k)$ is a polynomial (resp. quadratic, linear) function in k. A breakthrough result of Bodlaender et al. [3] gave the first framework for proving that certain parameterized problems do not admit polynomial kernels, by establishing so-called *composition algorithms*. Together with a result of Fortnow and Santhanam [14] this allows to exclude polynomial kernels under the assumption that NP $\not\subseteq$ coNP/poly, otherwise implying a collapse of the polynomial hierarchy to its third level [29].

2 NP-hardness, polynomial cases, and exact exponential algorithm

In this section we focus on the classical complexity of the d-Cut problem, and on exact exponential algorithms.

Chvátal [7] proved that Matching Cut is NP-hard for graphs of maximum degree at least four. In the next theorem, whose proof is inspired by the reduction of Chvátal [7] from 3-Uniform Hypergraph Bicoloring, we prove the NP-hardness of d-cut for $(2d+2)$-regular graphs. In particular, for $d = 1$ it implies the NP-hardness of Matching Cut for 4-regular graphs, which is a strengthening of Chvátal's [7] hardness proof.

▶ **Theorem 1** (⋆). *For every integer $d \geq 1$, d-cut is NP-hard even when restricted to $(2d+2)$-regular graphs.*

The graphs constructed by Theorem 1 are neither planar nor bipartite, but they are regular, a result that we were unable to find in the literature for Matching Cut. Note that every planar graph has a d-cut for every $d \geq 5$, so only the cases $d \in \{2, 3, 4\}$ remain open,

as the case $d = 1$ is known to be NP-hard [5]. Concerning graphs of bounded diameter, Le and Le [20] prove the NP-hardness of Matching Cut for graphs of diameter at least three by reducing Matching Cut to itself. It can be easily seen that the same construction given by Le and Le [20], but reducing d-Cut to itself, also proves the NP-hardness of d-Cut for every $d \geq 1$.

▶ **Corollary 2.** *For every integer $d \geq 1$, d-Cut is NP-hard for graphs of diameter at least three.*

We leave as an open problem to determine whether there exists a polynomial-time algorithm for d-Cut for graphs of diameter at most two for every $d \geq 2$, as it is the case for $d = 1$ [20].

We now turn to cases that can be solved in polynomial time. Our next result is a natural generalization of Chvátal's algorithm [7] for Matching Cut on graphs of maximum degree three.

▶ **Theorem 3** (⋆). *For any graph G and integer $d \geq 1$ such that $\Delta(G) \leq d + 2$, it can be decided in polynomial time if G has a d-cut. Moreover, for $d = 1$ any graph G with $\Delta(G) \leq 3$ and $|V(G)| \geq 8$ has a matching cut, for $d = 2$ any graph G with $\Delta(G) \leq 4$ and $|V(G)| \geq 6$ has a 2-cut, and for $d \geq 3$ any graph G with $\Delta(G) \leq d + 2$ has a d-cut.*

Theorems 1 and 3 present a "quasi-dichotomy" for d-cut on graphs of bounded maximum degree. Specifically, for $\Delta(G) \in \{d + 3, \ldots, 2d + 1\}$, the complexity of the problem remains unknown. However, we believe that most, if not all, of these open cases can be solved in polynomial time; see the discussion in Section 4.

To conclude this section, we present a simple exact exponential algorithm which, for every $d \geq 1$, runs in time $\mathcal{O}^*(c_d^n)$ for some constant $c_d < 2$. For the case $d = 1$, the currently known algorithms [18, 19] exploit structures that appear to get out of control when d increases, and so has a better running time than the one described below.

▶ **Theorem 4** (⋆). *For every fixed integer $d \geq 1$ and n-vertex graph G, there is an algorithm that solves d-Cut in time $\mathcal{O}^*((c_d)^n)$, for some constant $1 < c_d < 2$.*

3 Parameterized algorithms and kernelization

In this section we focus on the parameterized complexity of d-Cut. More precisely, in Section 3.1 we consider as the parameter the number of edges crossing the cut and in Section 3.2 the distance to cluster (in particular, we provide a quadratic kernel). The FPT algorithms parameterized by treewidth and the distance to co-cluster can be found in the full version of the paper.

Before proceeding, we introduce the notion of *monochromatic sets*.

▶ **Definition 5.** *A set of vertices $X \subseteq V(G)$ is said to be* monochromatic *if, for any d-cut (A, B) of G, $X \subseteq A$ or $X \subseteq B$. A subgraph H of G is monochromatic if $V(H)$ is monochromatic.*

3.1 Crossing edges

In this section we consider as the parameter the maximum number of edges crossing the cut. In a nutshell, our approach is to use as a black box one of the algorithms presented by Marx et al. [23] for a class of separation problems. Their fundamental problem is $\mathcal{G}$-MinCut, for a fixed class of graphs $\mathcal{G}$, which we state formally, along with their main result, below.

$\mathcal{G}$-MinCut
Instance: A graph G, vertices s, t, and an integer k.
Parameter: The integer k.
Question: Is there an induced subgraph H of G with at most k vertices such that $H \in \mathcal{G}$ and H is an $s-t$ separator?

▶ **Theorem 6** (Theorem 3.1 in [23]). *If $\mathcal{G}$ is a decidable and hereditary graph class, $\mathcal{G}$-MinCut is* FPT.

To be able to apply Theorem 6, we first need to specify a graph class to which, on the line graph, our separators correspond. We must also be careful to guarantee that the removal of a separator in the line graph leaves non-empty components in the input graph. To accomplish the latter, for each $v \in V(G)$, we add a private clique of size $2d$ adjacent only to it, choose one arbitrary vertex v' in each of them. The algorithm asks, for each pair v', u', whether or not a "special" separator of the appropriate size between v' and u' exists. We assume henceforth that these private cliques have been added to the input graph G. For each integer $d \geq 1$, we define the graph class $\mathcal{G}_d$ as follows.

▶ **Definition 7.** *A graph H belongs to $\mathcal{G}_d$ if and only if its maximum clique size is at most d.*

Note that $\mathcal{G}_d$ is clearly decidable and hereditary for every integer $d \geq 1$.

▶ **Lemma 8** (⋆). *G has a d-cut separating v' and u' if and only if the line graph of G has a vertex separator belonging to $\mathcal{G}_d$ that separates e_v and e_u, where e_v corresponds to the edge $vv' \in E(G)$ and e_u to the edge $uu' \in E(G)$.*

▶ **Theorem 9.** *For every $d \geq 1$, there is an* FPT *algorithm for d-Cut parameterized by k, the maximum number of edges crossing the cut.*

Proof. For each pair of vertices $s, t \in V(G)$ that do not belong to the private cliques, our goal is to find a subset of vertices $S \subseteq V(L(G))$ of size at most k that separates s and t such that $L(G)[S] \in \mathcal{G}_d$. This is precisely what is provided by Theorem 6, and the correctness of this approach is guaranteed by Lemma 8. Since we perform a quadratic number of calls to the algorithm given by Theorem 6, our algorithm still runs in FPT time. ◀

As to the running time of the FPT algorithm given by Theorem 9, the treewidth reduction technique of [23] relies on the construction of a monadic second order logic (MSOL) expression and Courcelle's Theorem [8] to guarantee fixed-parameter tractability, and therefore it is hard to provide an explicit running time in terms of k.

3.2 Kernelization and distance to cluster

The proof of the following theorem consists of a simple generalization to every $d \geq 1$ of the construction given by Komusiewicz et al. [18] for $d = 1$.

▶ **Theorem 10.** *For any fixed $d \geq 1$, d-Cut does not admit a polynomial kernel when simultaneously parameterized by k, Δ, and $\mathsf{tw}(G)$, unless* NP $\subseteq$ coNP/poly.

Proof. We show that the problem cross-composes into itself. Start with t instances $G_1, \ldots, G_t$ of d-Cut. First, pick an arbitrary vertex $v_i \in V(G_i)$, for each $i \in [t]$. Second, for $i \in [t-1]$, add a copy of K_{2d}, call it $K(i)$, every edge between v_i and $K(i)$, and every edge between $K(i)$ and v_{i+1}. This concludes the construction of G, which for $d = 1$ coincides with that presented by Komusiewicz et al. [18].

Suppose that (A, B) is a d-cut of some G_i and that $v_i \in A$. Note that $(V(G) \setminus B, B)$ is a d-cut of G since the only edges in the cut are those between A and B. For the converse, take some d-cut (A, B) of G and note that every vertex in the set $\{v_t\} \bigcup_{i \in [t-1]} \{v_i\} \cup K(i)$ is contained in the same side of the partition, say A. Since $B \neq \emptyset$, there is some i such that $B \cap V(G_i) \neq \emptyset$, which implies that there is some i (possibly more than one) such that $(A \cap V(G_i), B \cap V(G_i))$ must be a d-cut of G_i.

That the treewidth, maximum degree, and number of edges crossing the partition are bounded by n, the maximum number of vertices of the graphs G_i, is a trivial observation. ◀

We now proceed to show that d-Cut admits a polynomial kernel when parameterizing by the *distance to cluster* parameter, denoted by dc. A *cluster graph* is a graph such that every connected component is a clique; the *distance to cluster* of a graph G is the minimum number of vertices we must remove from G to obtain a cluster graph. Our results are heavily inspired by the work of Komusiewicz et al. [18]. Indeed, most of our reduction rules are natural generalizations of theirs. However, we need some extra observations and rules that only apply for $d \geq 2$, such as Rule 8.

We denote by $U = \{U_1, \ldots, U_t\}$ a set of vertices such that $G - U$ is a cluster graph, and each U_i is called a *monochromatic part* or *monochromatic set* of U, and we will maintain the invariant that these sets are indeed monochromatic. Initially, we set each U_i as a singleton. In order to simplify the analysis of our instance, for each U_i of size at least two, we will have a private clique of size $2d$ adjacent to every vertex of U_i, which we call X_i. The *merge* operation between U_i and U_j is the following modification: delete $X_i \cup X_j$, set U_i as $U_i \cup U_j$, U_j as empty, and add a new clique of size $2d$, $X_{i,j}$, which is adjacent to every element of the new U_i. We say that an operation is *safe* if the resulting instance is a YES instance if and only if the original instance was.

▶ **Observation 1.** *If $U_i \cup U_j$ is monochromatic, merging U_i and U_j is safe.*

It is worth mentioning that the second case of the following rule is not needed in the corresponding rule in [18]; we need it here to prove the safeness of Rules 7 and 8.

▶ **Reduction Rule 1.** *Suppose that $G - U$ has some cluster C such that*
1. *$(C, V(G) \setminus C)$ is a d-cut, or*
2. *$|C| \leq 2d$ and there is $C' \subseteq C$ such that $(C', V(G) \setminus C')$ is a d-cut.*

Then output YES.

After applying Rule 1, for every cluster C, C has some vertex with at least $d+1$ neighbors in U, or there is some vertex of U with at least $d+1$ neighbors in C. Moreover, note that no cluster C with at least $2d+1$ vertices can be partitioned in such a way that one side of the cut is composed only by a proper subset of vertices of C, i.e., C is monochromatic

The following definition is a natural generalization of the definition of the set N^2 given by Komusiewicz et al. [18]. Essentially, it enumerates some of the cases where a vertex, or set of vertices, is monochromatic, based on its relationship with U. However, there is a crucial difference that keeps us from achieving equivalent bounds both in terms of running time and size of the kernel, and which makes the analysis and some of the rules more complicated than in [18]. Namely, for a vertex to be forced into a particular side of the cut, it must have at least $d+1$ neighbors in that side; moreover, a vertex of U being adjacent to $2d$ vertices of a cluster C implies that C is monochromatic. Only if $d = 1$, i.e., when we are dealing with matching cuts, the equality $d + 1 = 2d$ holds. This gap between $d+1$ and $2d$ is the main difference between our kernelization algorithm for general d and the one shown in [18] for Matching Cut, and the main source of the differing complexities we obtain. In particular,

for $d = 1$ the fourth case of the following definition is a particular case of the third one, but this is not true anymore for $d \geq 2$. Figure 1 illustrates the set of vertices introduced in Definition 11.

▶ **Definition 11.** *For a monochromatic part $U_i \subseteq U$, let $N^{2d}(U_i)$ be the set of vertices $v \in V(G) \setminus U$ for which at least one of the following holds:*

1. *v has at least $d+1$ neighbors in U_i.*
2. *v is in a cluster C of size at least $2d+1$ in $G-U$ such that there is some vertex of C with at least $d+1$ neighbors in U_i.*
3. *v is in a cluster C of $G-U$ and some vertex in U_i has $2d$ neighbors in C.*
4. *v is in a cluster C of $G-U$ of size at least $2d+1$ and some vertex in U_i has $d+1$ neighbors in C.*

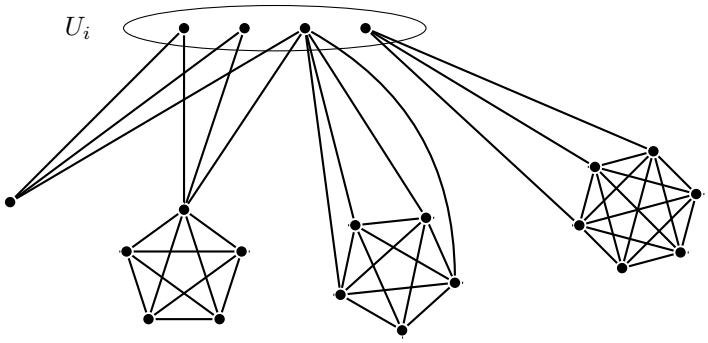


Figure 1 The four cases that define membership in $N^{2d}(U_i)$ for $d = 2$, from left to right.

▶ **Observation 2.** *For every monochromatic part U_i, $U_i \cup N^{2d}(U_i)$ is monochromatic.*

The next rules aim to increase the size of monochromatic sets. In particular, Rule 2 translates the transitivity of the monochromatic property, while Rule 3 identifies a case where merging the monochromatic sets is inevitable.

▶ **Reduction Rule 2.** *If $N^{2d}(U_i) \cap N^{2d}(U_j) \neq \emptyset$, merge U_i and U_j.*

▶ **Reduction Rule 3.** *If there there is a set of $2d+1$ vertices $L \subseteq V(G)$ with two common neighbors u, u' such that $u \in U_i$ and $u' \in U_j$, merge U_i and U_j.*

Proof of safeness of Rule 3. Suppose that in some d-cut (A, B), $u \in A$ and $u' \in B$, this implies that at most d elements of L are in A and at most d are in B, which is impossible since $|L| = 2d+1$. ◀

We say that a cluster is *small* if it has at most $2d$ vertices, and *big* otherwise. Moreover, a vertex in a cluster is *ambiguous* if it has neighbors in more than one U_i. A cluster is *ambiguous* if it has an ambiguous vertex, and *fixed* if it is contained in some $N^{2d}(U_i)$.

▶ **Observation 3.** *If G is reduced by Rule 1, every big cluster is ambiguous or fixed.*

Proof. Since Rule 1 cannot be applied, every cluster C has either one vertex v with at least $d+1$ neighbors in U or there is some vertex of a set U_i with $d+1$ neighbors in C. In the latter case, by applying the fourth case in the definition of $N^{2d}(U_i)$, we conclude that C is fixed. In the former case, either v has $d+1$ neighbors in the same U_i, in which case C is fixed, or its neighborhood is spread across multiple monochromatic sets, and so v and, consequently, C are ambiguous. ◀

Our next goal is to bound the number of vertices outside of U.

▶ **Reduction Rule 4.** *If there are two clusters C_1, C_2 contained in some $N^{2d}(U_i)$, then add every edge between C_1 and C_2.*

Proof of safeness of Rule 4. It follows directly from the fact that adding edges between vertices of a monochromatic set preserves the existence of a d-cut. ◀

The next lemma follows from the pigeonhole principle and exhaustive application of Rule 4.

▶ **Lemma 12.** *If G has been reduced by Rules 1 through 4, then G has $\mathcal{O}(|U|)$ fixed clusters.*

▶ **Reduction Rule 5.** *If there is some cluster C with at least $2d+2$ vertices such that there is some $v \in C$ with no neighbors in U, remove v from G.*

Proof of safeness of Rule 5. That G has a d-cut if and only if $G - v$ has a d-cut follows directly from the hypothesis that C is monochromatic in G and the fact that $|C \setminus \{v\}| \geq 2d+1$ implies that $C \setminus \{v\}$ is monochromatic in $G - v$. ◀

By Rule 5, we now have the additional property that, if C has more than $2d+1$ vertices, all of them have at least one neighbor in U. The next rule provides a uniform structure between a big cluster C and the sets U_i such that $C \subseteq N^{2d}(U_i)$.

▶ **Reduction Rule 6.** *If a cluster C has at least $2d+1$ elements and there is some U_i such that $C \subseteq N^{2d}(U_i)$, remove all edges between C and U_i, choose $u \in U_i$, $\{v_1, \ldots, v_{d+1}\} \subseteq C$ and add the edges $\{uv_i\}_{i \in [d+1]}$ to G.*

Proof of safeness of Rule 6. Let G' be the graph obtained after the operation is applied. If G has some d-cut (A, B), since $U_i \cup N^{2d}(U_i)$ is monochromatic, no edge between U_i and C crosses the cut, so (A, B) is also a d-cut of G'. For the converse, take a d-cut (A', B') of G'. Since C has at least $2d+1$ vertices and there is some $u \in U_i$ such that $|N(u) \cap C| = d+1$, $C \in N^{2d}(U_i)$ in G'. Therefore, no edge between C and U_i crosses the cut and (A', B') is also a d-cut of G. ◀

We have now effectively bounded the number of vertices in big clusters by a polynomial in U, as shown below.

▶ **Lemma 13.** *If G has been reduced by Rules 1 through 6, then G has $\mathcal{O}\big(d|U|^2\big)$ ambiguous vertices and $\mathcal{O}\big(d|U|^2\big)$ big clusters, each with $\mathcal{O}(d|U|)$ vertices.*

Proof. To show the bound on the number of ambiguous vertices, take any two vertices $u \in U_i$, $u' \in U_j$. Since we have $\binom{|U|}{2}$ such pairs, if we had at least $(2d+1)\binom{|U|}{2}$ ambiguous vertices, by the pigeonhole principle, there would certainly be $2d+1$ vertices in $V \setminus U$ that are adjacent to one pair, say u and u'. This, however, contradicts the hypothesis that Rule 3 has been applied, and so we have $\mathcal{O}\big(d|U|^2\big)$ ambiguous vertices.

The above discussion, along with Lemma 12 and Observation 3, imply that the number of big clusters is $\mathcal{O}\big(d|U|^2\big)$. For the bound on their sizes, take some cluster C with at least $2d+2$ vertices. Due to the application of Rule 5, every vertex of C has at least one neighbor in U. Moreover, there is at most one U_i such that $C \subseteq N^{2d}(U_i)$, otherwise we would be able to apply Rule 2.

Suppose first that there is such a set U_i. By Rule 6, there is only one $u \in U_i$ that has neighbors in C; in particular, it has $d+1$ neighbors. Now, every $v \in U_j$, for every $j \neq i$, has at

most d neighbors in C, otherwise $C \subseteq N^{2d}(U_j)$ and Rule 2 would have been applied. Therefore, we conclude that C has at most $(d+1) + \sum_{v \in U \setminus U_i} |N(u) \cap C| \leq (d+1) + d|U| \in \mathcal{O}(d|U|)$ vertices.

Finally, suppose that there is no U_i such that $C \subseteq N^{2d}(U_i)$. A similar analysis from the previous case can be performed: every $u \in U_i$ has at most d neighbors in C, otherwise $C \subseteq N^{2d}(U_i)$ and we conclude that C has at most $\sum_{v \in U} |N(u) \cap C| \leq d|U| \in \mathcal{O}(d|U|)$ vertices. ◀

We are now left only with an unbounded number of small clusters. A cluster C is *simple* if it is not ambiguous, that is, if for each $v \in C$, v has neighbors in a single U_i. Otherwise, C is ambiguous and, because of Lemma 13, there are at most $\mathcal{O}\big(d|U|^2\big)$ such clusters. For a simple cluster C and a vertex $v \in C$, we denote by $U(v)$ the monochromatic part of U to which v is adjacent.

▶ **Reduction Rule 7.** *If C is a simple cluster with at most $d+1$ vertices, remove C from G.*

Proof of safeness of Rule 7. Let $G' = G - C$. Suppose G has a d-cut (A, B) and note that $A \not\subseteq C$ and $B \not\subseteq C$ since Rule 1 does not apply. This implies that $(A \setminus C, B \setminus C)$ is a valid d-cut of G'. For the converse, take a d-cut (A', B') of G', define $C_A = \{v \in C \mid U(v) \subseteq A\}$, and define C_B similarly; we claim that $(A' \cup C_A, B' \cup C_B)$ is a d-cut of G. To see that this is the case, note that each vertex of C_A (resp. C_B) has at most d edges to C_B (resp. C_A) and, since C is simple, C_A (resp. C_B) has no other edges to B' (resp. A'). ◀

After applying the previous rule, every cluster C not yet analyzed has size $d+2 \leq |C| \leq 2d$ which, in the case of the Matching Cut problem, where $d = 1$, is empty. To deal with these clusters, given a d-cut (A, B), we say that a vertex v is in its *natural assignment* if $v \cup U(v)$ is in the same side of the cut; otherwise the vertex is in its *unnatural assignment*. Similarly, a cluster is *unnaturally assigned* if it has an unnaturally assigned vertex, otherwise it is *naturally assigned*.

▶ **Observation 4.** *Let $\mathcal{C}$ be the set of all simple clusters with at least $d+2$ and no more than $2d$ vertices, and (A, B) a partition of $V(G)$. If there are $d|U|+1$ edges uv, $v \in C \in \mathcal{C}$ and $u \in U$, such that uv is crossing the partition, then (A, B) is not a d-cut.*

Proof. Since there are $d|U|+1$ edges crossing the partition between $\mathcal{C}$ and U, there must be at least one $u \in U$ with $d+1$ neighbors in the other set of the partition. ◀

▶ **Corollary 14.** *In any d-cut of G, there are at most $d|U|$ unnaturally assigned vertices.*

Our next lemma limits how many clusters in $\mathcal{C}$ relate in a similar way to U; we say that two simple clusters C_1, C_2 have the same *pattern* if they have the same size s and there is a total ordering of C_1 and another of C_2 such that, for every $i \in [s]$, $v_i^1 \in C_1$ and $v_i^2 \in C_2$ satisfy $U(v_i^1) = U(v_i^2)$. Essentially, clusters that have the same pattern have neighbors in exactly the same monochromatic sets of U and the same multiplicity in terms of how many of their vertices are adjacent to a same monochromatic set U_i. Note that the actual neighborhoods in the sets U_i's do not matter in order for two clusters to have the same pattern. Figure 2 gives an example of a maximal set of unnaturally assigned clusters; that is, any other cluster with the same pattern as the one presented must be naturally assigned, otherwise some vertex of U will violate the d-cut property. As shown by the following Lemma, we may discard clusters that must be naturally assigned, as we can easily extend the kernel's d-cut, if it exists, to include them.

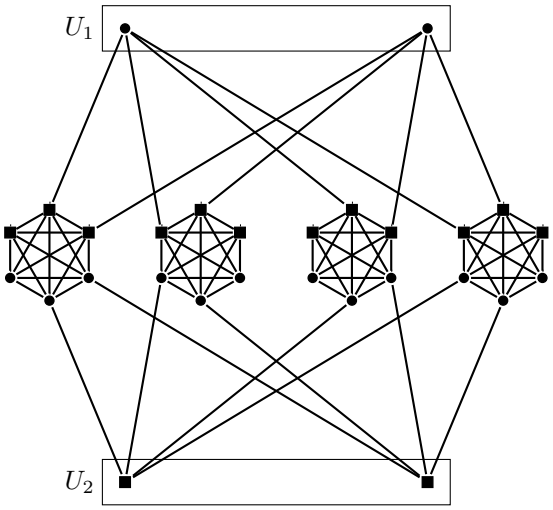


Figure 2 Example for $d = 4$ of a maximal set of unnaturally assigned clusters. Squared (resp. circled) vertices would be assigned to A (resp. B).

▶ **Lemma 15.** *Let $\mathcal{C}^* \subseteq \mathcal{C}$ be a subfamily of simple clusters, all with the same pattern, with $|\mathcal{C}^*| > d|U| + 1$. Let C be some cluster of $\mathcal{C}^*$, and $G' = G - C$. Then G has a d-cut if and only if G' has a d-cut.*

Proof. Since by Rule 1 no subset of a small cluster is alone in a side of a partition and, consequently, U intersects both sides of the partition, if G has a d-cut, so does G'.

For the converse, let (A', B') be a d-cut of G'. First, by Corollary 14, we know that at least one of the clusters of $\mathcal{C}^* \setminus \{C\}$, say C_{n}, is naturally assigned. Since all the clusters in $\mathcal{C}^*$ have the same pattern, this guarantees that *any* of the vertices of a naturally assigned cluster cannot have more than d neighbors in the other side of the partition.

Let (A, B) be the bipartition of $V(G)$ obtained from (A', B') such that $u \in C$ is in A (resp. B) if and only if $U(u) \subseteq A$ (resp. $U(u) \subseteq B$); that is, C is naturally assigned. Define $C_A = C \cap A$ and $C_B = C \cap B$. Because $|C| = |C_{\mathsf{n}}|$ and both belong to $\mathcal{C}^*$, we know that for every $u \in C_A$, it holds that $|N(u) \cap C_B| \leq d$; moreover, note that $N(u) \cap (B \setminus C) = \emptyset$. A symmetric analysis applies to every $u \in C_B$. This implies that no vertex of C has additional neighbors in the other side of the partition outside of its own cluster and, therefore, (A, B) is a d-cut of G. ◀

The safeness of our last rule follows directly from Lemma 15.

▶ **Reduction Rule 8.** *If there is some pattern such that the number of simple clusters with that pattern is at least $d|U| + 2$, delete all but $d|U| + 1$ of them.*

▶ **Lemma 16.** *After exhaustive application of Rules 1 through 8, G has $\mathcal{O}\big(d|U|^{2d}\big)$ small clusters and $\mathcal{O}\big(d^2|U|^{2d+1}\big)$ vertices in these clusters.*

Proof. By Rule 7, no small cluster with less than $d + 2$ vertices remains in G. Now, for the remaining sizes, for each $d + 2 \leq s \leq 2d$, and each pattern of size s, by Rule 8 we know that the number of clusters with s vertices that have the same pattern is at most $d|U| + 1$. Since we have at most $|U|$ possibilities for each of the s vertices of a cluster, we end up with $\mathcal{O}(|U|^s)$ possible patterns for clusters of size s. Summing all of them up, we get that we have $\mathcal{O}\big(|U|^{2d}\big)$ patterns in total, and since each one has at most $d|U| + 1$ clusters of size at most $2d$, we get that we have at most $\mathcal{O}\big(d^2|U|^{2d+1}\big)$ vertices in those clusters. ◀

The exhaustive application of all the above rules and their accompanying lemmas are enough to show that indeed, there is a polynomial kernel for d-Cut when parameterized by distance to cluster.

▶ **Theorem 17.** *When parameterized by distance to cluster* $\mathsf{dc}(G)$*,* d*-*Cut *admits a polynomial kernel with* $\mathcal{O}\big(d^2 \cdot \mathsf{dc}(G)^{2d+1}\big)$ *vertices that can be computed in* $\mathcal{O}\big(d^4 \cdot \mathsf{dc}(G)^{2d+1}(n+m)\big)$ *time.*

Proof. The algorithm begins by finding a set U such that $G - U$ is a cluster graph. Note that $|U| \leq 3\mathsf{dc}(G)$ since a graph is a cluster graph if and only if it has no induced path on three vertices: while there is some P_3 in G, we know that at least one its vertices must be removed, but since we don't know which one, we remove all three; thus, U can be found in $\mathcal{O}(\mathsf{dc}(G)(n+m))$ time. After the exhaustive application of Rules 1 through 8, by Lemma 13, $V(G) \setminus U$ has at most $\mathcal{O}\big(d^2 \cdot \mathsf{dc}(G)^3\big)$ vertices in clusters of size at least $2d+1$. By Rule 7, G has no simple cluster of size at most $d+1$. Ambiguous clusters of size at most $2d$, again by Lemma 13, also comprise only $\mathcal{O}\big(d^2 \cdot \mathsf{dc}(G)^2\big)$ vertices of G. Finally, for simple clusters of size between $d+2$ and $2d$, Lemmas 15 and 16 guarantee that there are $\mathcal{O}\big(d^2 \cdot \mathsf{dc}(G)^{2d+1}\big)$ vertices in small clusters and, consequently, this many vertices in G.

As to the running time, first, computing and maintaining $N^{2d}(U_i)$ takes $\mathcal{O}(d \cdot \mathsf{dc}(G)n)$ time. Rule 1 is applied only at the beginning of the kernelization, and runs in $\mathcal{O}\big(2^{2d}d(n+m)\big)$ time. Rules 2 and 3 can both be verified in $\mathcal{O}\big(d \cdot \mathsf{dc}(G)^2(n+m)\big)$ time, since we are just updating $N^{2d}(U_i)$ and performing merge operations. Both are performed only $\mathcal{O}\big(\mathsf{dc}(G)^2\big)$ times, because we only have this many pairs of monochromatic parts. The straightforward application of Rule 4 would yield a running time of $\mathcal{O}\big(n^2\big)$. However, we can ignore edges that are interior to clusters and only maintain which vertices belong together; this effectively allows us to perform this rule in $\mathcal{O}(n)$ time, which, along with its $\mathcal{O}(n)$ possible applications, yields a total running time of $\mathcal{O}\big(n^2\big)$ for this rule. Note that, when outputting the instance itself, we must write the edges explicitly; this does not change the final complexity of the algorithm, as each of the $\mathcal{O}\big(\mathsf{dc}(G)^{2d+1}\big)$ clusters has $\mathcal{O}(d \cdot \mathsf{dc}(G))$ vertices. Rule 5 is directly applied in $\mathcal{O}(n)$ time; indeed, all of its applications can be performed in a single pass. Rule 6 is also easily applied in $\mathcal{O}(n+m)$ time. Moreover, it is only applied $\mathcal{O}(\mathsf{dc}(G))$ times, since, by Lemma 13, the number of fixed clusters is linear in $\mathsf{dc}(G)$; furthermore, we may be able to reapply Rule 6 directly to the resulting cluster, at no additional complexity cost. The analysis for Rule 7 follows the same argument as for Rule 5. Finally, Rule 8 is the bottleneck of our kernel, since it must check each of the possible $\mathcal{O}\big(\mathsf{dc}(G)^{2d}\big)$ patterns, spending $\mathcal{O}(n)$ time for each of them. Each pattern is only inspected once because the number of clusters in a pattern can no longer achieve the necessary bound for the rule to be applied once the excessive clusters are removed. ◀

In the next theorems, we provide FPT algorithms for d-Cut parameterized by distance to cluster and distance to co-cluster, respectively. Both are based on dynamic programming, with the first being considerably simpler than the one given by Komusiewicz et al. [18] for $d = 1$, which applies four reduction rules and encodes the problem in a 2-SAT formula. However, for $d = 1$ our algorithm is slower, namely $\mathcal{O}^*\big(4^{\mathsf{dc}(G)}\big)$ compared to $\mathcal{O}^*\big(2^{\mathsf{dc}(G)}\big)$. Observe that the minimum distance to cluster and co-cluster sets can be computed in time $1.92^{\mathsf{dc}(G)} \cdot \mathcal{O}\big(n^2\big)$ and $1.92^{\mathsf{d\bar{c}}(G)} \cdot \mathcal{O}\big(n^2\big)$, respectively [6]. Thus, in the proofs of Theorems 18 and 19, we can safely assume that we have these sets at hand.

▶ **Theorem 18** (⋆)**.** *For every integer* $d \geq 1$*, there is an algorithm that solves* d*-*Cut *in time* $\mathcal{O}\big(4^d(d+1)^{\mathsf{dc}(G)}2^{\mathsf{dc}(G)}\mathsf{dc}(G)n^2\big)$.

▶ **Theorem 19** (⋆). *For every integer $d \geq 1$, there is an algorithm solving d-Cut in time $\mathcal{O}\big(32^d 2^{\mathsf{dc}(G)}(d+1)^{\mathsf{dc}(G)+d}(\mathsf{dc}(G)+d)n^2\big)$.*

Using Theorems 18 and 19, and the relation $\tau(G) \geq \max\{\mathsf{dc}(G), \overline{\mathsf{dc}}(G)\}$ [18], we obtain the following corollary.

▶ **Corollary 20.** *For every $d \geq 1$, d-Cut parameterized by vertex cover is in* FPT.

4 Concluding remarks

We presented a series of algorithms and complexity results; many questions, however, remain open. For instance, all of our algorithms have an exponential dependency on d on their running times. While we believe that such a dependency is an intrinsic property of d-cut, we have no proof for this claim. Similarly, the existence of a *uniform* polynomial kernel parameterized by the distance to cluster, i.e., a kernel whose degree does not depend on d, remains an interesting open question.

Also in terms of running time, we expect the constants in the base of the exact exponential algorithm to be improvable. However, exploring small structures that yield non-marginal gains as branching rules, as done by Komusiewicz et al. [18] for $d = 1$ does not seem a viable approach, as the number of such structures appears to rapidly grow along with d.

The distance to cluster kernel is hindered by the existence of clusters of size between $d+2$ and $2d$, an obstacle that is not present in the Matching Cut problem. Aside from the extremal argument presented, we know of no way of dealing with them. We conjecture that it should be possible to reduce the total kernel size from $\mathcal{O}\big(d^2\mathsf{dc}(G)^{2d+1}\big)$ to $\mathcal{O}\big(d^2\mathsf{dc}(G)^{2d}\big)$, matching the size of the smallest known kernel for Matching Cut [18].

We also leave open to close the gap between the polynomial and NP-hard cases in terms of maximum degree. We showed that, if $\Delta(G) \leq d+2$ the problem is easily solvable in polynomial time, while for graphs with $\Delta(G) \geq 2d+2$, it is NP-hard. But what about the gap $d+3 \leq \Delta(G) \leq 2d+1$? After some effort, we were unable to settle any of these cases. In particular, we are interested in 2-Cut, which has a single open case: $\Delta(G) = 5$. After some weeks of computation, we found no graph with more than 18 vertices and maximum degree five that had no 2-cut, in agreement with the computational findings of Ban and Linial [2]. Interestingly, all graphs on 18 vertices without a 2-cut are either 5-regular or have a single pair of vertices of degree 4, which are actually adjacent. In both cases, the graph is maximal in the sense that we cannot add edges to it while maintaining the degree constraints. We recall the initial discussion about Internal Partition; closing the gap between the known cases for d-Cut would yield significant advancements on the former problem.

Finally, the smallest d for which G admits a d-cut may be an interesting additional parameter to be considered when more traditional parameters, such as treewidth, fail to provide FPT algorithms by themselves. Unfortunately, by Theorem 1, computing this parameter is not even in XP, but, as we have shown, it can be computed in FPT time under many different parameterizations.

References

1 N. R. Aravind, Subrahmanyam Kalyanasundaram, and Anjeneya Swami Kare. On Structural Parameterizations of the Matching Cut Problem. In *Proc. of the 11th International Conference on Combinatorial Optimization and Applications (COCOA)*, volume 10628 of *LNCS*, pages 475–482, 2017.

2 Amir Ban and Nati Linial. Internal partitions of regular graphs. *Journal of Graph Theory*, 83(1):5–18, 2016.

3 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.

4 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Cross-Composition: A New Technique for Kernelization Lower Bounds. In *Proc. of the 28th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 9 of *LIPIcs*, pages 165–176, 2011.

5 Paul S. Bonsma. The complexity of the matching-cut problem for planar graphs and other graph classes. *Journal of Graph Theory*, 62(2):109–126, 2009.

6 Anudhyan Boral, Marek Cygan, Tomasz Kociumaka, and Marcin Pilipczuk. A fast branching algorithm for cluster vertex deletion. *Theory of Computing Systems*, 58(2):357–376, 2016.

7 Vasek Chvátal. Recognizing decomposable graphs. *Journal of Graph Theory*, 8(1):51–53, 1984.

8 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.

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

10 Matt DeVos. `http://www.openproblemgarden.org/op/friendly_partitions`, 2009.

11 Reinhard Diestel. *Graph Theory*, volume 173. Springer-Verlag, 4th edition, 2010.

12 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

13 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019.

14 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *Journal of Computer and System Sciences*, 77(1):91–106, 2011.

15 Ron L Graham. On primitive graphs and optimal vertex assignments. *Annals of the New York academy of sciences*, 175(1):170–186, 1970.

16 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

17 Atsushi Kaneko. On decomposition of triangle-free graphs under degree constraints. *Journal of Graph Theory*, 27(1):7–9, 1998.

18 Christian Komusiewicz, Dieter Kratsch, and Van Bang Le. Matching Cut: Kernelization, Single-Exponential Time FPT, and Exact Exponential Algorithms. In *Proc. of the 13th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 115 of *LIPIcs*, pages 19:1–19:13, 2018.

19 Dieter Kratsch and Van Bang Le. Algorithms solving the matching cut problem. *Theoretical Computer Science*, 609:328–335, 2016.

20 Hoàng-Oanh Le and Van Bang Le. On the Complexity of Matching Cut in Graphs of Fixed Diameter. In *Proc. of the 27th International Symposium on Algorithms and Computation (ISAAC)*, volume 64 of *LIPIcs*, pages 50:1–50:12, 2016.

21 Van Bang Le and Bert Randerath. On stable cutsets in line graphs. *Theoretical Computer Science*, 301(1-3):463–475, 2003.

22 Jie Ma and Tianchi Yang. Decomposing C_4-free graphs under degree constraints. *Journal of Graph Theory*, 90(1):13–23, 2019.

23 Dániel Marx, Barry O'Sullivan, and Igor Razgon. Treewidth Reduction for Constrained Separation and Bipartization Problems. In *Proc. of the 27th International Symposium on Theoretical Aspects of Computer Science, (STACS)*, volume 5 of *LIPIcs*, pages 561–572, 2010.

24 Augustine M Moshi. Matching cutsets in graphs. *Journal of Graph Theory*, 13(5):527–536, 1989.

25 Maurizio Patrignani and Maurizio Pizzonia. The complexity of the matching-cut problem. In *Proc. of the 27th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 2204 of *LNCS*, pages 284–295, 2001.

26 Khurram H. Shafique and Ronald D. Dutton. On satisfactory partitioning of graphs. *Congressus Numerantium*, pages 183–194, 2002.

27 Michael Stiebitz. Decomposing graphs under degree constraints. *Journal of Graph Theory*, 23(3):321–324, 1996.

28 Carsten Thomassen. Graph decomposition with constraints on the connectivity and minimum degree. *Journal of Graph Theory*, 7(2):165–167, 1983.

29 Chee-Keng Yap. Some Consequences of Non-Uniform Conditions on Uniform Classes. *Theoretical Computer Science*, 26:287–300, 1983.

Finding Linear Arrangements of Hypergraphs with Bounded Cutwidth in Linear Time

Thekla Hamm
Algorithms and Complexity Group, TU Wien, Vienna, Austria

Abstract

Cutwidth is a fundamental graph layout parameter. It generalises to hypergraphs in a natural way and has been studied in a wide range of contexts. For graphs it is known that for a fixed constant k there is a linear time algorithm that for any given G, decides whether G has cutwidth at most k and, in the case of a positive answer, outputs a corresponding linear arrangement. We show that such an algorithm also exists for hypergraphs.

2012 ACM Subject Classification Mathematics of computing → Permutations and combinations; Mathematics of computing → Hypergraphs; Theory of computation → Dynamic graph algorithms

Keywords and phrases Fixed parameter linear, Path decomposition, Hypergraph

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.20

Funding Supported by the Austrian Science Fund (FWF, Project P31336 and Project W1255-N23).

1 Overview

For a hypergraph H and an enumeration of its vertices $V(H)$, also referred to as *linear arrangement*, the *cutwidth* is the smallest integer k such that for every position i between 1 and $|V(H)| - 1$ there are at most k hyperedges with one endpoint among the first i and the other among the last $|V(H)| - i$ vertices of the arrangement. Finding orderings with small cutwidth for hypergraphs naturally occurs directly in various applications, probably the most classical one being VLSI design [6]. Also, there are problems for which a vertex ordering with small cutwidth can be used to obtain a provably good processing order for heuristic approaches, such as SAT [15]. We want to emphasise that a model using hypergraphs, as opposed to graphs, is necessary in these scenarios among others, as graphs do not capture all dependencies correctly.

If one wants to find an ordering with smallest possible cutwidth, this is the classical Minimum Cut Linear Arrangement Problem which is known to be NP-hard, even on graphs with maximum vertex degree 3 [7]. However, if one considers a bound k on the maximum allowed cutwidth as a parameter, one can decide in linear time if a linear ordering of a hypergraph is possible such that the bound is not exceeded [14]. The known proof is non-constructive in the sense that it does not infer a way to construct such a linear ordering in linear time. The asymptotically fastest constructive algorithm given in literature [8] is not even FPT (fixed parameter tractable) as the runtime complexity lies in $\mathcal{O}(n^{k^2+3k+3})$ where n is the number of vertices of the given hypergraph.

This Work. We give a linear time constructive algorithm.

A key observation is that the fixed bound also bounds the pathwidth of the incidence graph of the hypergraph. This allows to compute a path decomposition of the incidence graph in linear time using the result of Bodlaender [1], and then to dynamically compute a solution along the path decomposition. This approach has been successfully applied to a number of problems. An overview of the general method for the slightly more general notion of tree decompositions, as well as some examples for classical problems for which it was used, can be found e.g. in [3, Chapter 7]. In particular, the method has also been used to give a

linear time algorithm for the problem we consider restricted to graphs [11]. We extend the crucial ideas of this algorithm.

We will construct a linear ordering by successively inserting vertices into it as long as the cutwidth bound is satisfied and successively take into consideration more and more of the hyperedges when checking whether the cutwidth bound is satisfied. It is reasonable that if one takes this approach, after a vertex and all its incident hyperedges have been completely processed the position of this vertex is no longer interesting as it has no further impact on yet to be processed vertices and hyperedges. Similarly if a hyperedge and all its vertices have been processed the hyperedge will not play a role when extending the incomplete linear ordering. The first statement can be strengthened using the fact that only the positions of the outermost vertices of each hyperedge are necessary to compute the cutwidth of a linear ordering: After a vertex has been processed and the positions of the left- and rightmost vertices with respect to the incomplete linear ordering of each incident hyperedge are known, the vertex will not play a role when extending the incomplete linear ordering. A path decomposition of the incidence graph implies some order of processing the vertices and hyperedges such that no more than width-of-the-decomposition-many vertices and hyperedges are relevant in the way described above at the same time.

Structure. We start by giving a formal definition of our problem and an overview over previous research in Section 2. In Section 3, we introduce the pathwidth of the incidence graph of a hypergraph, and prove that a bound on the cutwidth induces a bound on it. We also work to obtain a path decomposition that has convenient structural properties for our dynamic programming approach. Section 4 is dedicated to problem specific arguments: In Section 4.1 we prove that it is sufficient to consider linear arrangements that are constructed in a certain way along the path decomposition. The idea is the same that was used for graphs [11]. It utilises the notion of so called typical sequences. Using the results of the previous sections, we formulate a dynamic program that works on path decompositions of incidence graphs and allows us to present our main result in Section 4.2. In Section 5 we indicate a possibility to lift the algorithm to a more general setting.

2 Formal Introduction and Preliminary Observations

We start with a hypergraph $H = (V, E)$. That is V is some finite set and E contains subsets of vertices, possibly with multiplicities, i.e. we allow parallel hyperedges.

▶ **Definition 1.** *A **linear arrangement** of a hypergraph H is a bijection $\varphi : V(H) \leftrightarrow \{1, \dots, |V(H)|\}$. The **cutwidth** of a linear arrangement φ **at position** $i \in \mathbb{N}$ is defined as* $\mathbf{cw(\varphi, i)} = |\{e \in E(H) \mid \exists v, w \in e \quad \varphi(v) \leq i < \varphi(w)\}|$. *The **cutwidth** of a linear arrangement φ is defined as* $\mathbf{cw(\varphi)} = \max_{i \in \mathbb{N}} \mathrm{cw}(\varphi, i)$. *The **cutwidth** of a hypergraph H is defined as* $\mathbf{cw(H)} = \min_{\substack{\varphi \text{ linear} \\ \text{arrangement of } H}} \mathrm{cw}(\varphi)$.

If one is interested in the cutwidth of hypergraphs, one can neglect all hyperedges in H that contain less than two vertices, as such hyperedges do not contribute to the cutwidth of any linear arrangement of H. From now on we will assume that $\forall e \in E(H)$, $|e| \geq 2$ for our input hypergraph H and refer to this assumption as **hyperedge cardinality assumption**. Note that any hypergraph H' can be transformed to satisfy this condition in time in $\mathcal{O}(|E(H')|)$ by deleting all hyperedges that are too small.

k-Cutwidth Bounded Linear Arrangement (k-CWLA)	
Instance	Hypergraph $H = (V, E)$ such that $\forall e \in E(H)$ $\lvert e\rvert \geq 2$
Task	Find a linear arrangement φ of H with $\mathrm{cw}(\varphi) \leq k$ or decide that $\mathrm{cw}(H) > k$.

We present a linear time algorithm, running in time in $\mathcal{O}(|V(H)|)$ that solves k-CWLA. If one drops the condition on the hyperedge cardinalities, this directly implies an algorithm that runs in time in $\mathcal{O}(|E(H)| + |V(H)|)$.

2.1 Known Results

The best algorithm given in literature for k-CWLA runs in time in $\mathcal{O}(|V(H)|^{k^2+3k+3})$ [8] and relies on complicated dynamic programming on a linear arrangement which is iteratively extended. There is an FPT-algorithm for constructing path decompositions of polymatroids with bounded width. Concretely:

▶ **Theorem 2** ([4]). *Let $\mathbb{F}$ be a fixed finite field. Given n subspaces of $\mathbb{F}^r$ for some r and a parameter k, in time in $\mathcal{O}(rm^2 + n^3)$, we can either find an enumeration $V_1, V_2, \ldots, V_n$ of the subspaces, such that $\dim((V_1 + \ldots + V_i) \cap (V_{i+1} + \ldots + V_n)) \leq k$ for all $i \in \{1, \ldots, n-1\}$, or decide that no such enumeration exists, where each V_i is given by its spanning set of d_i vectors and $m = \sum_{i=1}^{n} d_i$.*

One can write k-CWLA in a way that this theorem is applicable by setting $\mathbb{F} = E(H)$ and the subspaces to be the $|V(H)|$ subspaces spanned by the hyperedges incident to each of the vertices of H. Then one obtains a $\mathcal{O}(|E(H)|^5 + |V(H)|^3)$ (or $\mathcal{O}(|E(H)|^3 + |V(H)|^3)$ using the same observations as we do in Section 2.2) algorithm to solve k-CWLA. Thus this approach yields a runtime which is not linear in the size of the hypergraph. Also, as it is not specifically adapted to the problem at hand but rather general, if one wants to understand the algorithmic details, it is comparatively complicated and probably impractical. We also remark that the algorithm presented in this paper, leaves scope for extending to a more general setting (see Section 5).

The corresponding decision problem to k-CWLA however, i.e. deciding if the cutwidth of a hypergraph is smaller than k without outputting a corresponding linear arrangement, is known to be solvable in time in $\mathcal{O}(|V(H)| + |E(H)|)$. This was proved in [14]. The given algorithm is non-constructive and uses an adaptation of the analogue of the Myhill-Nerode theorem for coloured graphs to work on the incidence graph of hypergraphs. If one restricts the problem to hypergraph instances for which the hyperedge cardinality assumption holds, as we do, one can use the same observations that we will make in Section 2.2 to obtain an algorithm that runs in $\mathcal{O}(|V(H)|)$. In [13] van Bevern restates the open problem of constructing a linear arrangement with k-bounded cutwidth if possible in linear time and points out the importance of such an algorithm for practical purposes. It is also worth noting that the comparatively heavy machinery used to obtain the decision result leads to a high dependency of the complexity on k, and obstructs combinatorial properties of a solution.

For k-CWLA restricted to graphs a linear time algorithm is known [11]. Our algorithm for hypergraphs uses and extends the crucial ideas of this algorithm.

2.2 The Vertex Degree Property

▶ **Lemma 3.** *Let H satisfy the hyperedge cardinality assumption. If for any $v \in V(H)$ we have that $|\{e \in E(H) \mid v \in e\}| > 2k$ then $\mathrm{cw}(H) > k$.*

Lemma 3 motivates considering the **vertex degree property** of a hypergraph H which H satisfies if $\forall v \in V(H)\ |\{e \in E(H) \mid v \in e\}| \leq 2k$. Obviously, if H satisfies the vertex degree property, $|E(H)| \in \mathcal{O}(|V(H)|)$.

When claiming runtime in $\mathcal{O}(|V(H)|)$, we need to make clear how we assume H to be given as input. Note that in general the analogue of an incidence matrix for hypergraphs does not allow to check the vertex degree property in linear time. However a natural analogue of an adjacency list for hypergraphs, in which we list for each vertex the hyperedges it is incident to, can easily be seen to allow the following:

▶ Remark 4.

- Checking if H satisfies the vertex degree property in time in $\mathcal{O}(|V(H)|)$,
- and if H satisfies the vertex degree property, checking the membership of a given vertex in a given hyperedge in constant time.

3 Path Decompositions of Incidence Graphs

The cutwidth of a hypergraph relates to the pathwidth of its incidence graph, in two ways: Firstly this parameter is bounded by the cutwidth and secondly its boundedness allows the decomposition of the hypergraph given as instance in a way that is useful for our algorithm. We will elaborate on the properties that make the decomposition useful in Section 4 and for now focus on the definition of the parameter, proving the fact that it is bounded by cutwidth and showing that there is a decomposition of our hypergraph with certain properties.

▶ **Definition 5.** *The* ***incidence graph*** $G_I(H)$ *of* H *is given by* $V(G_I(H)) = \{v_u \mid u \in V(H)\} \dot\cup \{v_e \mid e \in E(H)\}$ *and* $E(G_I(H)) = \{\{v_u, v_e\} \mid u \in e\}$.

For a thorough description of our algorithm, we need the following technical statement that allows us to transform our input into easily accessible information about the incidence graph in linear time. The proof is not difficult.

▶ **Lemma 6.** *Let* H *be a hypergraph that has the vertex degree property. The following can be computed in time in* $\mathcal{O}(|V(H)|)$ *from our representation of* H*:*

- *The representation of* $G_I(H)$ *by its adjacency list;*
- *for each* $x \in V(G_I(H))$, $\text{type}(x) \in \{$*'vertex'*, *'hyperedge'*$\}$ *with*

$$\text{type}(x) = \begin{cases} \text{'vertex'} & \text{if } x \in \{v_u \in G_I(H) \mid u \in V(H)\} \\ \text{'hyperedge'} & \text{if } x \in \{v_e \in G_I(H) \mid e \in E(H)\} \end{cases}; \text{ and}$$

- *for each* $x \in \{v_u \in V(G_I(H)) \mid u \in V(H)\}$ *a lookup table for the corresponding* $u \in V(H)$.

The following definition was introduced by Robertson and Seymour [10]. It is a specialisation of the notion of tree decomposition and treewidth. Both have been studied in various contexts, one being as foundation for dynamic parameterised algorithms [3, Chapter 7].

▶ **Definition 7.** *A* ***path decomposition*** *of a graph* $G = (V, E)$ *is a sequence* $(X_1, \ldots, X_s)$ *with* $X_i \subseteq V(G)$ *such that*

(i) $\forall e \in E(G)\ \exists 1 \leq i \leq s$ *such that* $e \subseteq X_i$*; and*

(ii) $\forall v \in V(G)\ \exists 1 \leq i \leq j \leq s$ *such that* $v \in X_k \Leftrightarrow i \leq k \leq j$.

The ***width*** *of a path decomposition* $(X_1, \ldots, X_s)$ *is defined as* $\mathbf{width}((X_1, \ldots, X_s)) = \max_{1 \leq i \leq s} |X_i| - 1$. *The* ***pathwidth*** *of a graph* G *is defined as minimum width of a path decomposition of* G*, i.e.* $\mathbf{pw}(G) = \min_{\substack{(X_1,\ldots,X_s) \\ \text{path decomposition of } G}} \text{width}((X_1, \ldots, X_s))$. *The* X_i *are called the* ***bags*** *of the path decomposition.*

▶ **Theorem 8.** *Let H satisfy the hyperedge cardinality assumption. If* $\mathrm{cw}(H) \leq k$ *then* $\mathrm{pw}(G_I(H)) \leq k$.

Proof. Let φ be a linear arrangement of H with $\mathrm{cw}(\varphi) \leq k$ and define for $i = 1, \ldots, |V(H)|$
$X_{2i-1} = \{v_e \mid \exists v, w \in e \quad \varphi(v) < i \leq \varphi(w)\} \cup \{v_{\varphi^{-1}(i)}\}$ and
$X_{2i} = \{v_e \mid \exists v, w \in e \quad \varphi(v) \leq i < \varphi(w)\} \cup \{v_{\varphi^{-1}(i)}\}$.
It is easy to verify that this defines a path decomposition of $G_I(H)$ with width at most k. ◀

The boundedness of pathwidth in connection with the following well-known result will allow us to find a path decomposition of the incidence graph with bounded width in linear time.

▶ **Theorem 9** ([1])**.** *Given some graph G, it can be decided in time in $\mathcal{O}(|V(G)|)$ if* $\mathrm{pw}(G) \leq k$ *and if this is the case a path decomposition $(X_1, \ldots, X_s)$ of G with width at most k and $s \in \mathcal{O}(|V(G)|)$ can be computed in time in $\mathcal{O}(|V(G)|)$.*

The remainder of this section is dedicated to transforming the path decomposition we obtain from Theorem 9 into a more convenient form. The first step is standard.

▶ **Definition 10.** *A path decomposition $(X_1, \ldots, X_s)$ is* ***nice*** *if for all $1 \leq i \leq s$ it holds that $|X_i \triangle X_{i-1}| = 1$. (We artificially set $X_0 = \emptyset$.) Then for every X_i one of the following holds:*

- *$|X_i \setminus X_{i-1}| = 1$, then call X_i* ***introduce bag****. We say x with $\{x\} = X_i \setminus X_{i-1}$ is* ***introduced****.*
- *$|X_{i-1} \setminus X_i| = 1$, then call X_i* ***forget bag****. We say x with $\{x\} = X_{i-1} \setminus X_i$ is* ***forgotten****.*

▶ **Lemma 11** ([2])**.** *A path decomposition $(X_1, \ldots, X_s)$ of a graph G can be transformed into a nice path decomposition $(X'_1, \ldots, X'_{s'})$ of G in time in $\mathcal{O}(s \cdot \mathrm{width}((X_1, \ldots, X_s)))$ without increasing the width and with $s' \in \mathcal{O}(s \cdot \mathrm{width}((X_1, \ldots, X_s)))$.*

The fact that we also need to handle hyperedges leads to some additional technicalities. For this reason we will want to use a nice path decomposition with an additional restriction, which we call *extra niceness*, in our dynamic program.

▶ **Definition 12.** *A path decomposition $(X_1, \ldots, X_s)$ of an incidence graph $G_I(H)$ is* ***extra nice*** *if it is nice and for all $e \in E(H)$ there is some $u \in e$ such that v_u appears in the path decomposition before v_e, i.e.* $\forall e \in E(H) \quad \min_{\substack{v_u \in X_i \\ u \in e}} i < \min_{v_e \in X_i} i$.

Just like path decompositions can be efficiently be transformed into nice path decompositions, it is easy to modify nice path decompositions of an incidence graph to obtain extra nice path decompositions efficiently.

▶ **Lemma 13.** *Let H satisfy the hyperedge cardinality assumption and have the vertex degree property. A nice path decomposition $(X_1, \ldots, X_s)$ of $G_I(H)$ can be transformed into an extra nice path decomposition $(X'_1, \ldots, X'_s)$ of $G_I(H)$ in time in $\mathcal{O}(s \cdot \mathrm{width}((X_1, \ldots, X_s))^2)$ without increasing the width.*

The results of this section applied in series prove the following theorem:

▶ **Theorem 14.** *Given a hypergraph H that satisfies the hyperedge cardinality assumption and has the vertex degree property it can be decided in time in $\mathcal{O}(|V(H)|)$ if* $\mathrm{pw}(G_I(H)) \leq k$ *and, if this is the case, an extra nice path decomposition $(X_1, \ldots, X_s)$ of $G_I(H)$ with width at most k and $s \in \mathcal{O}(|V(H)|)$ can be computed in time in $\mathcal{O}(|V(H)|)$.*

After the observations in this section we may assume to have an extra nice path decomposition $(X_1, \ldots, X_s)$ of $G_I(H)$ with $s \in \mathcal{O}(|V(H)|)$ and $\mathrm{width}((X_1, \ldots, X_s)) \leq k$.

4 A Linear Time Algorithm for k-CWLA

Using our path decomposition we will dynamically construct a linear arrangement with bounded cutwidth, working on hypergraphs H_i that grow along the path decomposition and are contained in H in a certain sense.

▶ **Definition 15.** *For a hypergraph H and some $W \subseteq V(H)$ and $F \subseteq E(H)$ the* ***trimmed subhypergraph*** *of H* ***induced by W and F*** *is defined as $\boldsymbol{H[W,F]}$ with vertex set $V(H[W,F]) = W$ and hyperedge set $E(H[W,F]) = \{e \cap W \mid e \in F\}$. We call a hypergraph H'* ***trimmed subhypergraph*** *of H if there are $W \subseteq V(H)$ and $F \subseteq E(H)$ such that $H' = H[W,F]$.*

▶ **Definition 16.** *For a linear arrangement φ of $H = (V,E)$ and some $H' = (V',E')$ trimmed subhypergraph of H define the* ***restriction*** *of φ to H' $\boldsymbol{\varphi|_{H'}}$ to be the unique linear arrangement of H' such that $\forall v,w \in V(H') \quad \varphi(v) \leq \varphi(w) \Leftrightarrow \varphi|_{H'}(v) \leq \varphi|_{H'}(w)$.*
Correspondingly, if ψ is a linear arrangement of $H = (V,E)$ and φ is the restriction of ψ to some $H' = (V',E')$ where $V' \subseteq V$ and $E' = \{e \cap V' \mid e \in E''\}$ where $E'' \subseteq E$, then ψ is an ***extension*** *of φ. For a vertex $v \in V(H) \setminus V(H')$ and $1 \leq p < |V(H')|$, we say that v is* ***inserted*** *into position p of φ by ψ, if $\psi(v) \in \{\psi(\varphi^{-1}(p)), \ldots, \psi(\varphi^{-1}(p+1))\}$, and that v is inserted into position 0 os φ by ψ if $\psi(v) < \psi(\varphi^{-1}(1))$, and into position $|V(H')|$ of φ by ψ if $\psi(v) > \psi(\varphi^{-1}(|V(H')|))$.*

▶ Remark 17. Note that restrictions are indeed well-defined as a linear arrangement of $V(H)$ infers a unique linear arrangement of any subset of $V(H)$.

We are interested specifically in the following trimmed subhypergraphs of H, along which we will later iteratively extend our linear arrangement.

▶ **Definition 18.** *For $1 \leq i \leq s$ define*
$\boldsymbol{H_i} = H[\{u \in V(H) \mid v_u \in \bigcup_{1 \leq j \leq i} X_j\}, \{e \in E(H) \mid v_e \in \bigcup_{1 \leq j \leq i} X_j\}]$, *and* $\boldsymbol{H_0} = (\emptyset, \emptyset)$.

Assume to be at stage $1 \leq i \leq s$ when traversing the path decomposition, and let φ be a linear arrangement of H_i that we want to extend. The vertices of H that are represented in X_i, as well as the outermost vertices of (possibly trimmed) hyperedges that are represented in X_i, are of particular interest. This is due to the fact that, by the properties of a path decomposition, these are the vertices that may be important in a certain sense, when updating information about φ to extensions that include parts of the hypergraph that are still to be encountered. We formalise this intuition in the following definition and characterising remark.

▶ **Definition 19.** *We say a vertex $v \in V(H_i)$ is* ***unimportant after $\boldsymbol{\varphi}$*** *if*

- *all hyperedges of H, incident to v are considered in $E(H_i)$, i.e. $\forall e \in E(H) \quad v \in e \Rightarrow e \cap V(H_i) \in E(H_i)$; and*
- *no hyperedge of H, containing vertices from $V(H_i)$, as well as $V(H) \setminus V(H_i)$, has any v as left- or rightmost vertex in φ, i.e.*
  $\forall e \in E(H) \quad e \cap V(H_i) \neq \emptyset \wedge e \cap \big(V(H) \setminus V(H_i)\big) \neq \emptyset$
  $$\Rightarrow \operatorname*{argmin}_{w \in e \cap V(H_i)} \varphi(w) \neq v \wedge \operatorname*{argmax}_{w \in e \cap V(H_i)} \varphi(w) \neq v.$$

Let $1 \leq i \leq s$ and φ be a linear arrangement of H_i. The set of ***vertices distinguished by $\boldsymbol{\varphi}$*** *is the set $\{u \in V(H) \mid v_u \in X_i\} \cup \{\operatorname*{argmin}_{u \in e \cap V(H_i)} \varphi(u), \operatorname*{argmax}_{u \in e \cap V(H_i)} \varphi(u) \mid v_e \in X_i\}$.*

▶ Remark 20. Note that by the properties of a path decomposition in particular all vertices in $V(H_i) \setminus \{u \in V(H) \mid v_u \in X_i\} \cup \{\operatorname{argmin}_{u \in e \cap V(H_i)} \varphi(u), \operatorname{argmax}_{u \in e \cap V(H_i)} \varphi(u) \mid v_e \in X_i\}$ are unimportant after φ, i.e. all vertices that are not distinguished by φ, are unimportant after φ.

The properties in the definition of unimportantness are what we will exploit to identify linear arrangements that we have need to consider at a certain stage, in terms of whether or not they allow an extension to a complete linear arrangement of H with cutwidth at most k. In this way we will be able to restrict ourselves to considering at most constantly many (w.r.t. the size of the hypergraph) linear arrangements at each stage. We give the details in Section 4.1 and use these results in Section 4.2 to give a linear time algorithm for k-CWLA.

4.1 Identifying Partial Linear Arrangements

We use the same idea as in [11] to give a condition under which two linear arrangements of H_i for some $i \in \{1, \ldots, s\}$ either can both, or can both not be extended to a linear arrangement of H satisfying the cutwidth bound. More specifically we consider the relative order of distinguished vertices in such linear arrangements and where exactly, vertices that are yet to be encountered may be inserted. In certain cases we will be able to shift vertices that are not distinguished and are strewn between vertices distinguished by φ between two distinguished vertices that are consecutive w.r.t. φ without increasing the cutwidth.

▶ **Lemma 21.** *Let $1 \leq i \leq s$ and φ be a linear arrangement of H_i. Assume there are p and $q \in \mathbb{N}$ such that*

(i) $\operatorname{cw}(\varphi, p) = \min_{0 \leq j \leq q} \operatorname{cw}(\varphi, p+j)$ *and* $\operatorname{cw}(\varphi, p+q) = \max_{0 \leq j \leq q} \operatorname{cw}(\varphi, p+j)$*; and*

(ii) $\varphi^{-1}(p+1), \ldots, \varphi^{-1}(p+q)$ *are unimportant after φ.*

Let ψ be a linear arrangement of H that extends φ. Obtain ψ' from ψ by shifting for each $1 \leq \bar{j} \leq q$ the vertices from in between $\varphi^{-1}(p+\bar{j})$ and $\varphi^{-1}(p+\bar{j}+1)$ to in between $\varphi^{-1}(p)$ and $\varphi^{-1}(p+1)$ and otherwise maintaining the same relative ordering as given by ψ. Formally the described ψ' is defined by:

$$\psi'(v) = \begin{cases} \psi(v) - \bar{j} & \text{if } \psi(\varphi^{-1}(p+\bar{j})) < \psi(v) < \psi(\varphi^{-1}(p+\bar{j}+1)) \\ & \quad \text{for some } 1 \leq \bar{j} \leq q \\ \psi(\varphi^{-1}(p+q+1)) - (q+1-\bar{j}) & \text{if } v = \varphi^{-1}(p+\bar{j}) \text{ for some } 1 \leq \bar{j} \leq q \\ \psi(v) & \text{otherwise} \end{cases}$$

Then $\operatorname{cw}(\psi') \leq \operatorname{cw}(\psi)$.

▶ Remark 22. See Figure 1 for an illustration of how φ, ψ and ψ' relate.

Proof. Note that if $j \notin \{\psi(\varphi^{-1}(p+1)), \ldots, \psi(\varphi^{-1}(p+q+1)) - 1\}$ by construction of ψ' it holds that $\forall v \in V(H) \quad \psi(v) \leq j \Leftrightarrow \psi'(v) \leq j$. So for all such j we have $\operatorname{cw}(\psi, j) = \operatorname{cw}(\psi', j)$. It remains to consider positions between $\psi(\varphi^{-1}(p+1))$ and $\psi(\varphi^{-1}(p+q+1)) - 1$. For $j' \in \{\psi(\varphi^{-1}(p+1)), \ldots, \psi(\varphi^{-1}(p+2)) - 1\}$ we will find $j \in \mathbb{N}$ such that $\operatorname{cw}(\psi', j') \leq \operatorname{cw}(\psi, j)$. We distinguish two cases according to the cases considered in the definition of the value of ψ' at $\psi'^{-1}(j')$:

Case 1: $\psi(\varphi^{-1}(p+\bar{j})) < \psi(\psi'^{-1}(j')) < \psi(\varphi^{-1}(p+\bar{j}+1))$ **for some** $1 \leq \bar{j} \leq q$ **.** One can set $j = \psi(\psi'^{-1}(j'))(= j' + \bar{j})$ (see Figure 1, red), and use the fact that $\varphi^{-1}(p+1), \ldots, \varphi^{-1}(p+q)$ are unimportant after φ, and the fact that $\operatorname{cw}(\varphi, p) \leq \operatorname{cw}(\varphi, p+\bar{j})$ to show $\operatorname{cw}(\psi', j') \leq \operatorname{cw}(\psi, j)$.

Case 2: $\psi'^{-1}(j') = \varphi^{-1}(p+\bar{j})$ **for some** $1 \leq \bar{j} \leq q$**.** One can set $j = \psi(\varphi^{-1}(p+q))$ (see Figure 1, blue), and use the fact that $\varphi^{-1}(p+1), \ldots, \varphi^{-1}(p+q)$ are unimportant after φ, and the fact that $\operatorname{cw}(\varphi, p+\bar{j}) \leq \operatorname{cw}(\varphi, p+q)$ to show $\operatorname{cw}(\psi', j') \leq \operatorname{cw}(\psi, j)$. ◀

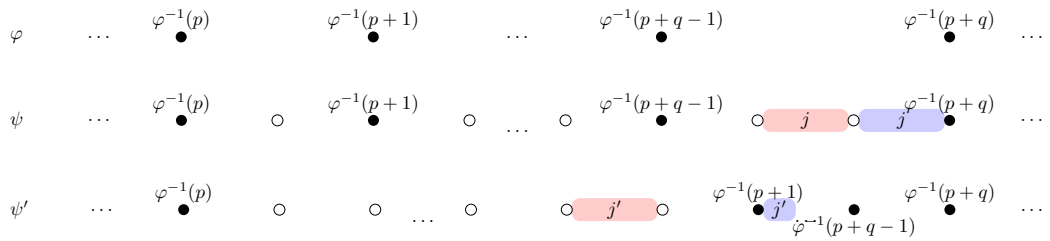


Figure 1 Illustrations of the situation in Lemma 21 and its proof – vertices in $V(H_i)$ are filled, exemplary positions at which cutwidths are compared in Case 1 red and Case 2 blue.

One can show an analogous result for the smallest cutwidth being at the largest position:

▶ **Lemma 23.** *Let $1 \leq i \leq s$ and φ be a linear arrangement of H_i. Assume there are p and $q \in \mathbb{N}$ such that*

(i) $\mathrm{cw}(\varphi, p+q) = \min_{0 \leq j \leq q} \mathrm{cw}(\varphi, p+j)$ *and* $\mathrm{cw}(\varphi, p) = \max_{0 \leq j \leq q} \mathrm{cw}(\varphi, p+j)$*; and*

(ii) $\varphi^{-1}(p+1), \ldots, \varphi^{-1}(p+q)$ *are unimportant after φ.*

Let ψ be a linear arrangement of H that extends φ. Obtain ψ' from ψ by shifting for each $0 \leq \bar{j} \leq q-1$ the vertices from in between $\varphi^{-1}(p+\bar{j})$ and $\varphi^{-1}(p+\bar{j}+1)$ to in between $\varphi^{-1}(p+q)$ and $\varphi^{-1}(p+q+1)$ and otherwise maintaining the same relative ordering as given by ψ. Then $\mathrm{cw}(\psi') \leq \mathrm{cw}(\psi)$.

Lemma 21 and Lemma 23 yield:

▶ **Lemma 24.** *Let $1 \leq i \leq s$ and φ be a linear arrangement of H_i.*

1. *If $p \in \{1, \ldots, |V(H_i)| - 1\}$ is a position, $\varphi^{-1}(p+1)$ is not a vertex distinguished by φ, and $\mathrm{cw}(\varphi, p) = \mathrm{cw}(\varphi, p+1)$, then there is a linear arrangement ψ' of H that extends φ such that $\mathrm{cw}(\psi') = \min_{\substack{\psi \text{ extension of} \\ \varphi \text{ to } H}} \mathrm{cw}(\psi)$, and no vertex is inserted into position $p+1$ of φ by ψ'.*
2. *If $q \geq 2$ and $p \in \{1, \ldots, |V(H_i)| - 1 - q\}$ is a position, for all $1 \leq j \leq q$, $\varphi^{-1}(p+j)$ is not a vertex distinguished by φ, and $\min_{0 \leq j \leq q} \mathrm{cw}(\varphi, p+j) = \mathrm{cw}(\varphi, p)$ and $\max_{0 \leq j \leq q} \mathrm{cw}(\varphi, p+j) = \mathrm{cw}(\varphi, p+q)$ or vice versa, then there is a linear arrangement ψ' of H that extends φ such that $\mathrm{cw}(\psi') = \min_{\substack{\psi \text{ extension of} \\ \varphi \text{ to } H}} \mathrm{cw}(\psi)$, and no vertex is inserted into any of the positions $p+1, \ldots, p+q-1$ of φ by ψ'.*

Proof. Apply Lemma 21 or Lemma 23 to ψ and the relevant positions, where ψ is any linear arrangement of H that extends φ and achieves minimal cutwidth among these. ◀

This immediately relates to so called typical sequences. Such sequences were introduced and studied in [2] (and implicitly in [5]) where it was shown that there are boundedly many as well as that they have bounded length. These bounds will be important for us later on.

▶ **Definition 25.** *For a sequence of natural numbers $n_1 \ldots n_t$ its **typical sequence**, $\boldsymbol{\tau(n_1 \ldots n_t)}$, arises from $n_1 \ldots n_t$ by performing the following operations until neither of them is applicable anymore:*

- *Removing consecutive repetitions of entries; or*
- *removing subsequences $n_{i_2} \ldots n_{i_{u-1}}$, with $u \geq 3$ and $\min_{1 \leq j \leq u} n_{i_j} = n_{i_1}$ and $\max_{1 \leq j \leq u} n_{i_j} = n_{i_u}$ or vice versa.*

*A sequence of natural numbers is **typical** if it is the typical sequence of some sequence.*

Note that for a sequence $n_1 \ \dots \ n_t$, its typical sequence can be computed by going through the sequence repeatedly and performing the two operations until no longer possible. As the operations necessarily shorten the sequence, this can be done in time in $\mathcal{O}(t^2)$.

▶ **Lemma 26** ([2, Lemma 3.5]). *There are no more than $\frac{8}{3}2^{2k}$ different typical sequences whose entries are bounded by k.*

▶ **Lemma 27** ([2, Lemma 3.3(ii)]). *A typical sequence whose entries are bounded by k has length at most $2k-1$.*

By iterating Lemma 21 and Lemma 23 and making the additional observation that we never shift vertices into a position of φ whose cutwidth is removed in the typical sequences of the cutwidths between distinguished vertces, we can refine Lemma 24 to:

▶ **Lemma 28.** *Let $1 \leq i \leq s$ and φ be a linear arrangement of H_i. Then there is a linear arrangement ψ' of H that extends φ such that $\mathrm{cw}(\psi') = \min_{\substack{\psi \text{ extension of} \\ \varphi \text{ to } H}} \mathrm{cw}(\psi)$, and for $q \geq 1$ and a position $p \in \{1, \dots, |V(H_i)| - 1 - q\}$ such that $\varphi^{-1}(p)$ and $\varphi^{-1}(p+q+1)$ are distinguished by φ, all $\varphi^{-1}(p+j)$ are not distinguished by φ, no vertex is inserted into any position p' of φ such that $\mathrm{cw}(\varphi, p')$ is removed in $\tau(\mathrm{cw}(\varphi, p) \ \dots \ \mathrm{cw}(\varphi, p+q))$ by ψ'.*

▶ **Definition 29.** *For a linear arrangement φ of H_i with $1 \leq i \leq s$, $p \in \{1, \dots, |V(H_i)| - 1\}$ is a **typical insertion position** of φ, if it is a position into which an extension of φ to H as described in Lemma 28 inserts vertices. 0 and $|V_i|$ are always typical insertion positions of φ. I.e. p is a typical insertion position of φ if p is not removed in $\tau(\mathrm{cw}(\varphi, \max_{\substack{p' \leq p \wedge \varphi^{-1}(p') \text{ is} \\ \text{distinguished by } \varphi}} p') \ \dots \ \mathrm{cw}(\varphi, \min_{\substack{p' > p \wedge \varphi^{-1}(p') \text{ is} \\ \text{distinguished by } \varphi}} p'))$. We define **insertion-typical** linear arrangements inductively: The empty linear arrangement of H_0 is insertion-typical. for $1 \leq i+1 \leq s$, a linear arrangement is a insertion-typical linear arrangement of H_{i+1} if it arises from an insertion-typical linear arrangement of H_i by insertion of the at most one introduced vertex into a typical insertion position of this linear arrangement.*

▶ Remark 30. Let $1 \leq i \leq s$ and φ be a linear arrangement of H_i. Because, by definition, there is always a maximum value entry of $n_1 \ \dots \ n_t$ in $\tau(n_1 \ \dots \ n_t)$, there is a typical insertion position $p \in \{1, \dots, |V(H_i)| - 1\}$ such that $\mathrm{cw}(\varphi) = \mathrm{cw}(\varphi, p)$.

▶ Remark 31. An insertion-typical linear arrangement with cutsidth at most k has at most $(2k-1)^2 \in \mathcal{O}(k^2)$ typical insertion positions. This follows from the fact that at most $2k$ vertices are distinguished by a linear arrangement between every two consecutive of which, by Lemma 27, the cutwidths at typical insertion positions form a sequence of length $\leq 2k-1$.

▶ Remark 32. Let $1 \leq i \leq s$ and φ be an insertion-typical linear arrangement of H_i with a typical insertion position p of φ. Then $p' = \max\{q \in \{0, \dots, |V(H_{i-1})| \mid \varphi(\varphi|^{-1}_{H_{i-1}}(q)) \leq p\}$ is a typical insertion position of $\varphi|_{H_{i-1}}$. Otherwise, assume $\mathrm{cw}(\varphi|_{H_{i-1}}, p')$ to be removed in the typical sequence of the cutwidths among which the cutwidth at p' is considered. Because vertices in $V(H_i) \setminus V(H_{i-1})$ are only inserted into typical insertion positions of $\varphi|_{H_{i-1}}$ by φ, and hyperedges that are considered in H_i but not in H_{i-1} only contain vertices that are distinguished by $\varphi|_{H_{i-1}}$, the cutwidth increase is uniform for the cutwidths among which the cutwidth at p' is considered when moving from $\varphi|_{H_{i-1}}$ to φ. This contradicts p being a typical insertion position of φ.

▶ **Theorem 33.** *If $\mathrm{cw}(H) \leq k$, then there is an insertion-typical linear arrangement ψ of H such that $\mathrm{cw}(\varphi) \leq k$.*

Proof. Let ψ witness $\mathrm{cw}(H) \leq k$. $\psi|_{H_0}$ is trivially insertion-typical. By applying Lemma 28 to $\psi|_{H_0}$ and ψ we can modify ψ in a way that $\psi|_{H_1}$ is insertion-typical while maintaining the same restriction to H_0 and the cutwidth bound of k. (This first step is somewhat pathological, since no actual modification is needed.)
Similarly we can use Lemma 28 on the restriction of this modified ψ to H_1 and this modified ψ to obtain a ψ with an insertion-typical restriction to H_2 while maintaining the same restriction to H_1, and hence also to H_0, and the cutwidth bound of k. Iterating this procedure up to s proved the statement. ◀

4.2 The Dynamic Programming Procedure

Theorem 33 allows us to consider only insertion-typical linear arrangements along the path decomposition. In the following, we describe a dynamic program to construct linear arrangements of the trimmed subhypergraphs induced by the extra nice path decomposition while enforcing a bound of k on their cutwidth. As is usual for algorithms processing path decompositions, we first define our records (i.e. the information stored about each partial solution), what they look like for the initial (empty) stage and how to infer a complete solution from the record at the last stage. Then we describe how to update these records when encountering introduce- and forget bags respectively.

The records. For $1 \leq i \leq s$ our record consists of the following for each insertion-typical linear arrangement φ of H_i with cutwidth at most k:

- The **working information**, which is the weak order $<_\varphi$ on $\{u \in V(H) \mid v_u \in X_i\} \cup \{(e, \mathrm{left}), (e, \mathrm{right}) \mid e \in E(H) \wedge v_e \in X_i\}$ and the cutwidths of φ at each typical insertion position, that is given in the following way:
  - for $v, w \in \{u \in V(H) \mid v_u \in X_i\}$, $v <_\varphi w$, iff $\varphi(v) < \varphi(w)$;
  - for $v \in \{u \in V(H) \mid v_u \in X_i\}$, $v <_\varphi (e, \mathrm{left})$, iff $\varphi(v) < \min_{w \in e \cap V(H_i)} \varphi(w)$ (analogously for (e, right) and $\max_{w \in e \cap V(H_i)} \varphi(w)$);
  - for $v \in \{u \in V(H) \mid v_u \in X_i\}$ and a typical insertion position $p \in \{1, \ldots, |V(H_i)| - 1\}$, $v <_\varphi \mathrm{cw}(\varphi, p)$, iff $\varphi(v) < p$;
  - for (e, left) and a typical insertion position $p \in \{1, \ldots, |V(H_i)| - 1\}$, $v <_\varphi \mathrm{cw}(\varphi, p)$, iff $\min_{w \in e \cap V(H_i)} \varphi(w) < p$ (analogously for (e, right) and $\max_{w \in e \cap V(H_i)} \varphi(w)$); and
  - for typical insertion positions p and q, $\mathrm{cw}(\varphi, p) <_\varphi \mathrm{cw}(\varphi, q)$, iff $p < q$.
- The **solution information**, which in turn consists of
  a map $pos : (\mathrm{cw}(\varphi, p))_{p \text{ a typical insertion position of } \varphi} \to \{0, \ldots, |V(H_i)|\}$, that associates each cutwidth value which is stored in the working information to the typical insertion position at which it is attained;
  a pointer *prec* to the entry in the record at the preceding stage, that corresponds to $\varphi|_{H_{i-1}}$;
  and, if X_i introduces v_u for some $u \in V(H)$, the solution information also specifies $ins \in \{0, \ldots, |V(H_{i-1})|\}$ to be $\mathrm{cw}(\varphi|_{H_{i-1}}, p)$ where p is the typical insertion position of $\varphi|_{H_{i-1}}$ which u is inserted into by φ.

Then the initial record (for $i = 0$) is given by a single cutwidth value 0 as working information, and the single cutwidth value 0 in the working information is mapped to position 1 by *pos*. If the record at stage s is empty then there is no insertion-typical linear arrangement of $H_s = H$ with cutwidth at most k. In this case we output that no linear arrangement of H satisfying cutwidth bound k exists. By Theorem 33 this means $\mathrm{cw}(H) > k$. Otherwise there is an entry in the record at stage s which corresponds to an insertion-typical linear

arrangement of H with (using Remark 30) cutwidth at most k, hence in particular a solution to k-CWLA. We can retrace this linear arrangement using the solution information: Starting the entry in the record at stage i, iteratively traverse the entries corresponding to the restriction at the preceding stage by using the *prec*-pointers in every step. Concurrently save all encountered *ins* in reverse order. By definition, the j-th element of the resulting sequence $\pi = \pi_1 \ \dots \ \pi_{|V(H)|}$ describes the position of the vertex $u \in V(H)$ such that v_u is the j-th of $\{v_w \mid w \in V(H)\}$ that is introduced in our path decomposition, relative to the $j-1$ first such vertices. This allows us to reconstruct the linear arrangement, e.g. as a linked list of vertices of H, representing the order in which it enumerates $V(H)$: For $j = 1, \dots, |V(H)|$ Set the j-th introduced vertex to be between the π_j-th and $(\pi_j + 1)$-th element of the list constructed so far. As inserting into a linked list, takes constant time, the construction runs in time linear in the number of bags of the path decomposition, which in turn is linear in the size of the incidence graph because it is nice.

In the following we describe how to update records when encountering introduce- and forget bags respectively. Proofs for correctness can be given by technical, but straightforward inductions on the stage, using the properties of a nice path decomposition and Remarks 30 and 32. It is also easy to check, using the bound from Remark 31 that the runtime of each of the given procedures lies in $\mathcal{O}(k^3)$.

Introduce bag. Let $1 \leq i \leq s$, assume to have the record for stage i and that X_{i+1} is a bag that introduces $v \in G_I(H)$. We do a case distinction on the type of v:

$v = v_w$ for some $w \in V(H)$. A insertion-typical linear arrangement of H_{i+1} arises from an insertion-typical linear arrangement of H_i by inserting w into a typical insertion position of that linear arrangement. By induction hypothesis, there is an entry in the record at stage i for such a linear arrangement φ which also contains all cutwidths at typical insertion positions of φ in form of the $\mathrm{dom}(<_\varphi) \setminus V(H_i)$. So, for every entry of the record at stage i which corresponds to some φ, and for each element $c \in \mathrm{dom}(<_\varphi) \setminus V(H_i)$, create an entry of the record at stage $i+1$ that corresponds to the linear arrangement φ' arising from φ by inserting w into position $pos(c)$ of φ. The working- and solution information for this entry can be obtained in the following way: Obviously *prec* points to the entry for φ and $ins = pos(c)$.
Define $<_{\varphi'}$ on $\mathrm{dom}(<_\varphi) \setminus \{c\} \cup \{c_{\text{left}}, c_{\text{right}}, w\}$ by setting $x <_{\varphi'} y$ whenever $x, y \in \mathrm{dom}(<_\varphi)$ and $x <_\varphi y$; $x <_{\varphi'} y$ if $x \in \{c_{\text{left}}, c_{\text{right}}, w\}, y \in \mathrm{dom}(<_\varphi)$ and $c <_\varphi y$; and $c_{\text{left}} <_{\varphi'} w <_{\varphi'} c_{\text{right}}$. – Intuitively, in this way we fix the correct relative position of w in φ. The position corresponding to c is split up by the insertion of w into that position.
For every $e \in \{f \in E(H) \mid v_f \in X_{i+1}\} (= \{f \in E(H) \mid v_f \in X_i\})$, if $w \in e$ and $w <_{\varphi'} (e, \text{left})$, increment every cutwidth value d with $w <_{\varphi'} d <_{\varphi'} (e, \text{left})$ by one, and set $(e, \text{left}) <_{\varphi'} x$ iff $w <_{\varphi'} x$(, i.e. '$(e, \text{left}) =_{\varphi'} w$'). Similarly, if $w \in e$ and $(e, \text{right}) <_{\varphi'} w$, we increment every cutwidth value d with $(e, \text{right}) <_{\varphi'} d <_{\varphi'} w$ by one , and set $(e, \text{right}) <_{\varphi'} x$ iff $w <_{\varphi'} x$(, i.e. '$(e, \text{right}) =_{\varphi'} w$'). – Intuitively, in this way we increment the cutwidth values at positions at which a hyperedge contributes to the cutwidth value only after taking w into account. If, at any point, we surpass k by these incrementations, we abort and do not add the entry to the record, as it corresponds to a linear arrangement violating the cutwidth bound.
For every $d \in \mathrm{dom}(<_{\varphi'}) \setminus V(H_{i+1}) \setminus \{c_{\text{left}}, c_{\text{right}}\}$, pos remains the same, if $d <_{\varphi'} c_{\text{left}}$, and is increased by one otherwise. We also set $pos(c_{\text{left}})$ to be the evaluation of the old pos of c and $pos(c_{\text{right}}) = pos(c_{\text{left}}) + 1$. – Intuitively this means we shift the positions in a way to make space for the newly split position, that was created by the insertion of w.
Finally, note that after these modifications our information might include cutwidths at positions that are no longer typical insertion positions (e.g. because the cutwidth values around a position changed). To amend this, we can apply the typical sequence operator τ to

sequences of cutwidth values that are between the remaining elements of $\mathrm{dom}(<_{\varphi'}) \cap (\{u \in V(H) \mid v_u \in X_i\} \cup \{(e, \mathrm{left}), (e, \mathrm{right}) \mid e \in E(H) \wedge v_e \in X_i\})$ and delete the cutwidths removed by this operation from the domains of $<_{\varphi'}$ and *pos*. ◁

$v = v_e$ for some $e \in E(H)$. An insertion-typical linear arrangement of H_{i+1} is by definition also an insertion-typical linear arrangement of H_i. So, for every entry of the record at stage i which corresponds to some φ we create an entry of the record at stage $i+1$ also corresponding to φ. We need to adapt the working- and solution information to take e into account in the cutwidths of φ: Let *prec* point to the entry corresponding to φ in the record at stage i. Because of the extra niceness of our path decomposition there is at least one vertex of e represented in X_{i+1}. Let v be the $<_\varphi$-minimal, and v' to be the $<_\varphi$-maximal $w \in e \cap \{w \in V(H) \mid v_w \in X_{i+1}\}$ ($v = v'$ is possible). Set $(e, \mathrm{left}) <_{\varphi'} x$ iff $v <_{\varphi'} x$(, i.e. '$(e, \mathrm{left}) =_{\varphi'} v$') and set $(e, \mathrm{right}) <_{\varphi'} x$ iff $v' <_{\varphi'} x$(, i.e. '$(e, \mathrm{right}) =_{\varphi'} v'$'). – Intuitively, we find the outermost vertices of e and set the markers (e, left) and (e, right) correspondingly. Leave $<_\varphi$ and *pos* unchanged.
Increment all cutwidth values $c \in \mathrm{dom}(<_{\varphi'})$ with $(e, \mathrm{left}) <_{\varphi'} c <_{\varphi'} (e, \mathrm{right})$ by one. – Intuitively this corresponds to counting e in all cutwidth values, to which it contributes when it is considered. If we surpass k as a cutwidth value by these incrementations, we abort and do not add the entry to the record, as it corresponds to a linear arrangement violating the cutwidth bound. Because this incrementation actually only changes the cutwidth values uniformly for a sequence of cutwidths between distinguished vertices, the typical insertion positions remain unchanged by the consideration of e. ◁

Forget bag. Let $1 \leq i \leq s$, assume to have the record for stage i and that X_{i+1} is a bag that forgets $v \in G_I(H)$. By definition, in this situation, an insertion-typical linear arrangement of H_{i+1} is also an insertion-typical linear arrangement of H_i. Moreover the cutwidth values of any such linear arrangement do not change when moving from H_i to H_{i+1}. Thus, we only have to remove elements from the domain of $<_\varphi$ after forgetting v and make exactly the same amends described in the last paragraph of the case of a bag introducing v_w with $w \in V(H)$, because the removal of certain distinguished vertices might lead to fewer typical insertion positions. For the first step, we remove w from the domain of $<_\varphi$, if $v = v_w$, and remove (e, left) and (e, right) from the domain of $<_\varphi$, if $v = v_e$.

As presented, the dynamic program solves k-CWLA, given a path decomposition of width at most k, in time in $\mathcal{O}(|V(H)| \cdot \mathfrak{b} \cdot k^4)$, where $\mathfrak{b}$ is a bound on the largest number of insertion-typical linear arrangements with cutwidth at most k at a stage. However, at closer inspection, we realise that during the dynamic program solution information is only used to update solution information, and never working information. What is more, solution information is never used to check the cutwidth bound, i.e. to exclude potential linear arrangements, but solely to reconstruct a solution, after a successful traversal of the path decomposition. This implies that actually two insertion-typical linear arrangements with cutwidth at most k can be either both or both not be extended to such a linear arrangement of H if they imply the same solution information. Thus during the algorithm, we only need to add an entry to a record at stage i, if there is no entry with the same solution information already. With this additional argument the runtime of the algorithm lies in $\mathcal{O}(|V(H)| \cdot \mathfrak{w} \cdot k^3)$, where $\mathfrak{w}$ is the number of possibilities for working information of an insertion-typical linear arrangement with cutwidth at most k. We can bound $\mathfrak{w}$ by $(2k)! \cdot (\frac{8}{3} 2^{2k})^{2k-1}$ using Lemma 26 and the bounded width of the path decomposition.

Thus we obtain a linear time algorithm for k-CWLA, by first checking the vertex degree property, in case of a positive answer computing an extra nice tree decomposition and, in case of success applying the described dynamic program, and outputting $\mathrm{cw}(H) > k$ if any stage fails.

▶ **Theorem 34.** *k-CWLA can be solved in time in $\mathcal{O}(|V(H)|)$ and $\mathcal{O}^*(2^{\mathcal{O}(k^2)})$.*

5 Future Work – Using a Tree Decomposition of the Incidence Graph

One can show that the cutwidth of a hypergraph is equal to the product of its maximum vertex degree and the pathwidth of its incidence graph. Hence our algorithm immediately infers an algorithm for solving the Minimum Cut Linear Arrangement Problem in FPT-time parameterised by the maximum vertex degree and the pathwidth of its incidence graph. Reductions in literature show that dropping the restriction on the incidence pathwidth results in NP-hardness [9, Corollary 2.10], and can easily be modified to show the same for dropping the restriction on the maximum vertex degree [9, slight modification of the proof of Lemma 2.4]. (It is not difficult to see that graphs of bounded path- or treewidth also have bounded incidence path- or treewidth respectively.) In this sense, the algorithm is tight.

However the natural question arises, whether incidence pathwidth can be generalised to incidence treewidth. This has been successfully achieved for the analogous algorithm for k-CWLA restricted to graphs [12], and also seems possible for hypergraphs. The idea is, to extend the given dynamic program to also be able to handle tree decompositions of the incidence graph, by specifying the behaviour at *join bags*. In this situation partial linear arrangements for each trimmed subhypergraph corresponding to a bag immediately below the join bag have to "interleave". The number of possibilities for them to do so, is linear in the size of the hypergraph, however we can argue as in Section 4.1 that only interleaving at typical insertion positions of the respective partial linear arrangements needs to be considered.

References

1. Hans L. Bodlaender. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM Journal on Computing*, 25:1305–1317, December 1996. doi:10.1137/S0097539793251219.
2. Hans L. Bodlaender and Ton Kloks. Efficient and Constructive Algorithms for the Pathwidth and Treewidth of Graphs. *Journal of Algorithms*, 21(2):358–402, 1996. doi:10.1006/jagm.1996.0049.
3. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.
4. Jisu Jeong, Eun Jung Kim, and Sang-il Oum. Constructive algorithm for path-width of matroids. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1695–1704, 2016. doi:10.1137/1.9781611974331.ch116.
5. Jens Lagergren and Stefan Arnborg. Finding minimal forbidden minors using a finite congruence. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez Artalejo, editors, *Automata, Languages and Programming*, pages 532–543, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.
6. Fillia Makedon and Ivan Hal Sudborough. On Minimizing Width in Linear Layouts. *Discrete Appl. Math.*, 23(3):243–265, June 1989. doi:10.1016/0166-218X(89)90016-4.

7 Fillia S. Makedon, Christos H. Papadimitriou, and Ivan H. Sudborough. Topological bandwidth. In Giorgio Ausiello and Marco Protasi, editors, *CAAP'83*, pages 317–331, Berlin, Heidelberg, 1983. Springer Berlin Heidelberg.

8 Zevi Miller and Ivan Hal Sudborough. A Polynomial Algorithm for Recognizing Bounded Cutwidth in Hypergraphs. *Mathematical Systems Theory*, 24(1):11–40, 1991. doi:10.1007/BF02090388.

9 Burkhard Monien and Ivan Hal Sudborough. Min Cut is NP-Complete for Edge Weighted Treees. *Theor. Comput. Sci.*, 58:209–229, 1988.

10 Neil Robertson and P.D. Seymour. Graph minors. I. Excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983. doi:10.1016/0095-8956(83)90079-5.

11 Dimitrios M. Thilikos, Maria Serna, and Hans L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. *Journal of Algorithms*, 56(1):1–24, 2005. doi:10.1016/j.jalgor.2004.12.001.

12 Dimitrios M. Thilikos, Maria Serna, and Hans L. Bodlaender. Cutwidth II: Algorithms for partial w-trees of bounded degree. *Journal of Algorithms*, 56(1):25–49, 2005. doi:10.1016/j.jalgor.2004.12.003.

13 René van Bevern. *Fixed-parameter linear-time algorithms for NP-hard graph and hypergraph problems arising in industrial applications*. PhD thesis, Berlin Institute of Technology, 2014. URL: http://d-nb.info/1058974750.

14 René van Bevern, Rodney G. Downey, Michael R. Fellows, Serge Gaspers, and Frances A. Rosamond. Myhill-Nerode Methods for Hypergraphs. *Algorithmica*, 73(4):696–729, December 2015. doi:10.1007/s00453-015-9977-x.

15 Dong Wang, Edmund M. Clarke, Yunshan Zhu, and James H. Kukula. Using cutwidth to improve symbolic simulation and Boolean satisfiability. In *Proceedings of the Sixth IEEE International High-Level Design Validation and Test Workshop 2001, Monterey, California, USA, November 7-9, 2001*, pages 165–170, 2001. doi:10.1109/HLDVT.2001.972824.

The Independent Set Problem Is FPT for Even-Hole-Free Graphs

Edin Husić
Department of Mathematics, LSE, Houghton Street, London, WC2A 2AE, United Kingdom
e.husic@lse.ac.uk

Stéphan Thomassé
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France
Institut Universitaire de France, Paris, France
stephan.thomasse@ens-lyon.fr

Nicolas Trotignon
Univ Lyon, ENS de Lyon, Université Claude Bernard Lyon 1, CNRS, LIP, F-69342,
Lyon Cedex 07, France
nicolas.trotignon@ens-lyon.fr

Abstract

The class of even-hole-free graphs is very similar to the class of perfect graphs, and was indeed a cornerstone in the tools leading to the proof of the Strong Perfect Graph Theorem. However, the complexity of computing a maximum independent set (MIS) is a long-standing open question in even-hole-free graphs. From the hardness point of view, MIS is W[1]-hard in the class of graphs without induced 4-cycle (when parameterized by the solution size). Halfway of these, we show in this paper that MIS is FPT when parameterized by the solution size in the class of even-hole-free graphs. The main idea is to apply twice the well-known technique of augmenting graphs to extend some initial independent set.

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

Keywords and phrases independent set, FPT algorithm, even-hole-free graph, augmenting graph

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.21

Funding The second and third named authors are partially supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

Acknowledgements The majority of paper was prepared while the first named author was a student at ENS de Lyon.

1 Introduction

Given a (finite, simple, undirected) graph $G = (V, E)$ we say that a subset of vertices $I \subseteq V$ is *independent* if every two vertices in I are non-adjacent. The *maximum independent set problem* is the problem of finding an independent set of maximum cardinality in a given graph G. This problem is NP-hard even for planar graphs of degree at most three [5], unit disk graphs [3], and C_4-free graphs [1]. , To see that the independent set problem is NP-hard in the class of C_4-free graphs, one can use the following observation by Poljak [10]. Namely, $\alpha(G') = \alpha(G) + 1$ where the graph G' is obtained from G by replacing a single edge with a P_4 (i.e., subdividing it twice). By replacing every edge with a P_4 we obtain a graph that has girth at least nine, and thus MIS is NP-hard for C_4-free graphs. Similarly, MIS is NP-hard for the class of graphs with girth at least l, where $l \in \mathbb{N}$ is fixed.

On the contrary, when the input is restricted to some particular class of graphs the problem can be solved efficiently. Examples of such classes are bipartite graphs [8], chordal

graphs [6] and claw-free graphs [9, 11]. The maximum independent set problem is also polynomially solvable when the input is restricted to the class of perfect graphs using the ellipsoid method [7], but it remains an open question to find a combinatorial algorithm[1] in this case. In fact, we do not even have a combinatorial FPT algorithm for the maximum independent set problem on perfect graphs.

Closely related to the class of perfect graphs is the class of even-hole-free graphs. The class of even-hole-free graphs was introduced as a class structurally similar to the class of Berge graphs. We say that a graph is *Berge* if and only if it is odd-hole-free and odd-antihole-free, i.e., $\{C_5, C_7, \overline{C_7}, C_9, \overline{C_9}, \dots\}$-free[2]. The similarity follows from the fact that by forbidding C_4, we also forbid all antiholes on at least 6 vertices. Hence, an even-hole-free graph does not contain an antihole on at least 6 vertices, i.e., it is $\{C_4, C_6, \overline{C_6}, \overline{C_7}, C_8, \overline{C_8} \dots\}$-free. It should be noted that techniques obtained in the study of even-hole-free graphs were successfully used in the proof of the Strong Perfect Graph Theorem. A decomposition theorem, an algorithm for the maximum weighted clique problem and several other polynomial algorithms for classical problems in subclasses of even-hole-free graphs can be found in survey [12].

We denote by $\alpha(G)$ the maximum cardinality of an independent set in a graph G. In this paper we consider a parameterized version of the problem, that is we consider the following decision problem.

Independent Set:	
Input:	A graph G.
Parameter:	k.
Output:	TRUE if $\alpha(G) \geq k$ and FALSE otherwise.

We say that a problem is *fixed parameter tractable* (FPT) parameterized by the solution size k, if there is an algorithm running in time $O(f(k)n^c)$ for some function f and some constant c. More generally, a problem is fixed parameter tractable with respect to the parameter k (e.g. solution size, tree-width, ...) if for any instance of size n, it can be solved in time $O(f(k)n^c)$ for some fixed c. Usually, we consider whether a problem is FPT if the problem is already known to be NP-hard. In that case, the function f is not in any way bounded by a polynomial. In other words, for fixed parameter tractable problems, the difficulty is not in the input size, but rather in the size of the solution (parameter). In general, the Independent Set problem is not fixed-parameter tractable (parameterized by the size of solution) unless W[1]=FPT or informally, we believe that there is no FPT algorithm for the problem [4]. Recently, it has been shown that MIS is W[1]-hard for C_4-free graphs [2]. Even stronger, the same paper proves that MIS is W[1]-hard in any family of graphs defined by finitely many forbidden induced holes.

While the exact complexity of the maximum independent set problem is still open for the class of even-hole-free graphs, we present a step forward by showing that there is an FPT algorithm for the problem.

Main idea

Our algorithm is based on the augmentation technique. More precisely, in order to compute a solution of size $k+1$, we compute disjoint solutions of size k. The main property we use is that the union of two independent sets in an even-hole-free graph induces a forest. The

[1] The term combinatorial algorithm is used for an algorithm that does not rely on the ellipsoid method.
[2] Berge graphs are exactly perfect graphs by the Strong Perfect Graph Theorem.

key-point of our algorithm is that if W, X are disjoint solutions of size k, and Y is some (unknown) solution of size $k+1$, then the two trees induced by $X \cup Y$ and $W \cup Y$ are very constrained. This leads to a reduction to the chordal graph case, where MIS is tractable by dynamic programming.

Preliminaries

We consider finite, simple and undirected graphs. For a graph $G = (V, E)$ we write $uv \in E$ for an edge $\{u, v\} \in E(G)$, in this case u and v are *adjacent*. For a vertex $v \in V(G)$ we denote by $N_G(v) = \{u \in V : uv \in E\}$ *the neighborhood* of v and for $W \subseteq V$, we define $N_G(W) = \cup_{w \in W} N_G(w) \setminus W$. We drop the subscript when it is clear from the context. Let $S \subseteq V$. We say that S is *complete* to W if every vertex in S is adjacent to every vertex in W. The *induced subgraph* $G[W]$ is defined as the graph $H = (W, E \cap \binom{W}{2})$ where $\binom{W}{2}$ is the set of all unordered pairs in W. For a set A we denote by A^2 the set of all ordered pairs with elements in A. The graph $G[V \setminus W]$ is denoted $G \setminus W$ and when $W = \{w\}$ we write $G \setminus w$. A subset of vertices is called a *clique* if all the vertices are pairwise adjacent. A chordless cycle on at least four vertices is called a *hole*. A hole is even (resp. odd) if it contains an even (resp. odd) number of vertices. A *path* is a graph obtained by deleting one vertex of a chordless cycle. A path with endvertices u, v is called a u, v-path. Given a path Z and two of its vertices v, u we denote by vZu the smallest subpath of Z containing v and u. An *in-arborescence* is an orientation of a tree in which every vertex apart one (the *root*) has outdegree one.

2 Reduction steps and augmenting graphs

Our main goal is to show that the following problem is FPT.

Independent Set in Even-Hole-Free Graphs (ISEHF):	
Input:	An even-hole-free graph G.
Parameter:	k.
Output:	An independent set of size k if $\alpha(G) \geq k$ and FALSE otherwise.

We define a simpler version of the ISEHF problem where we know more about the structure of G. Later, we show that it suffices to find an FPT algorithm for the simpler version.

Transversal Independent Set in Even-Hole-Free Graphs (TISEHF):	
Input:	An even-hole-free graph G and a partition of $V(G)$ into cliques $X_1, \ldots, X_k$.
Parameter:	k.
Output:	An independent set of size k if $\alpha(G) \geq k$ and FALSE otherwise.

Note that in TISEHF, an independent set of size k must intersect every clique on exactly one vertex, i.e., it must traverse all cliques.

▶ **Lemma 1.** *The ISEHF problem is FPT if and only if the TISEHF problem is FPT.*

Proof. Note that the only if implication is obvious, so we assume that we already have an FPT algorithm $\mathcal{A}$ for TISEHF, and provide one for ISEHF. We claim that it suffices to exhibit an algorithm $\mathcal{B}$ running in time $g(k)n^c$ which takes as input the pair (G, k) and either outputs an independent set of size k or a cover of $V(G)$ by $2^{k-1} - 1$ cliques. Indeed, one

then just has to apply algorithm $\mathcal{A}$ to every possible choice of k disjoint cliques induced by the $2^{k-1}-1$ cliques which are output by $\mathcal{B}$. We describe $\mathcal{B}$ inductively on k: If $k=2$, then G is either a clique, or contains two non-adjacent vertices x, y. When $k>2$, we compute two non-adjacent vertices x, y (or return the clique G). We now apply $\mathcal{B}$ to the graph induced by the set X of non-neighbors of x: we either get an independent set of size $k-1$ (in which case we are done by adding x) or cover X by $2^{k-2}-1$ cliques. We apply similarly $\mathcal{B}$ to the set Y of non-neighbors of y. Note that $X \cup Y$ covers all vertices of G except the common neighbors N of x and y. Since G is C_4-free, N is a clique, and therefore we have constructed a cover of $V(G)$ by $2(2^{k-2}-1)+1$ cliques. ◀

We turn to our main result. In the rest of this section we further reduce the problem to a graph together with two particular trees. Section 3 defines the notion of bi-trees and shows how two trees interact under certain conditions. Then, in Section 4, we prove that bi-trees arising from even-hole-free graphs satisfy these conditions and conclude the algorithm.

▶ **Theorem 2.** *The TISEHF problem is FPT.*

Proof. We assume that we have already shown that there is an algorithm $\mathcal{A}$ which solves TISEHF(G, j) in time $O(f(j)n^3)$ for every $j \leq k$. Our goal is to extend this by showing that $f(k+1)$ exists. Our input is a partition of G into cliques $X_1, \ldots, X_k, X_{k+1}$ (which we call *parts*) and we aim to either find an independent set intersecting all parts or show that none exists. In what follows, we assume that an independent set $Y = \{y_1, \ldots, y_k, y_{k+1}\}$ intersecting all parts exists, and whenever a future argument will end up with a contradiction, this will always be a contradiction to the existence of Y, and thus our output will implicitly be FALSE.

The first step is to apply $\mathcal{A}$ to $X_1, \ldots, X_k$ to compute an independent set $W = \{w_1, \ldots, w_k\}$. If it happens that $W \cap Y \neq \emptyset$, we guess which w_i belongs to Y and run $\mathcal{A}$ on the k remaining parts in which we have deleted all neighbors of w_i. This costs k calls to TISEHF(G, k) which is in our budget. So we may assume that W is disjoint from Y, and even stronger that no vertex of W belongs to an independent set of size $k+1$, since one of the previous k calls would have detected it. Moreover, since there is no even hole, $W \cup Y$ induces a forest T_1. Note that no vertex of W is isolated in T_1 since the parts are cliques. Note also that T_1 cannot have a leaf w_i in W, since w_i would belong to an independent set of size $k+1$ by exchanging it with y_i. Thus every vertex of W has degree at least two in T_1. Since the number of edges of T_1 is at most $2k$, we have that every vertex of W has degree 2 and T_1 is a tree.

As there is only $h(k)$ possible choices for the structure of T_1, we call $h(k)$ branches of computations for each of these choices of T_1. This means that in each call, we only keep the vertices of the parts X_i which corresponds to the possible neighborhoods of vertices of W. For instance, in the call corresponding to a tree T_1 in which w_1 is adjacent to y_1 and y_2, we delete all neighbors of w_1 in parts $X_3, \ldots, X_{k+1}$ and delete all non-neighbors of w_1 in X_2 (no further cleaning is needed in X_1 since it is a clique). Therefore, we assume that every vertex of W is complete to exactly two parts (including its own) and non-adjacent to others. Moreover, we define a *white tree* on vertex set $\{1, \ldots, k+1\}$ by having an edge between i and j if there exists a vertex w of W which is complete to X_i and X_j. We will refer to this vertex w as $w_{i,j}$. In what follows, we do not consider anymore that the vertices of W belong to the parts X_j and rather see them as external vertices of our problem. Thus, since we are free to rename the parts, we can assume that $k+1$ is a leaf of the white tree.

This is the crucial point of the algorithm, we have obtained a more structured input, but unfortunately we could not directly take advantage of it to conclude the main theorem.

Instead, we apply again algorithm $\mathcal{A}$ to $X_1, \ldots, X_k$ to compute a second independent set $X = \{x_1, \ldots, x_k\}$ (if such an X does not exist, we thus return FALSE as Y cannot exist). As done previously, we may assume that X is disjoint from Y, the tree T_2 spanned by $X \cup Y$ can also be guessed, and the degrees of vertices of X in T_2 is two (see Figure 1, down-left). We now interpret T_2 in a slightly different way: we root T_2 at y_{k+1} and orient all the edges toward the root. By doing so, every edge $\{x_i, y_i\}$ gives the arc $y_i x_i$ while the unique neighbor $y_{r(i)}$ of x_i, which is different from y_i, gives the arc $x_i y_{r(i)}$. We now further clean the parts X_j as follows: for every x_i, we delete all neighbors of x_i in X_j for $j \neq i, r(i)$, and we delete all non-neighbors of x_i in $X_{r(i)}$. We now have two trees which endow our parts: the white tree and the *red in-arborescence* defined on vertex set $\{1, \ldots, k+1\}$ by the arc set $\{ir(i) : i = 1, \ldots, k\}$. Our tool is now ready: the correlation between these two trees will provide an $O(k \cdot n^3)$ time algorithm to compute Y, or show that Y does not exist. We now turn to a special section devoted to bi-trees, i.e., trees defined on the same set of vertices under some structural constraints.

3 Bi-trees

Let V be a set of vertices. A *bi-tree* is a triple $T = (V, A, E)$ where $E \subseteq \binom{V}{2}$ is a set of edges such that (V, E) is a tree and $A \subseteq V^2$ is a set of arcs such that (V, A) is an in-arborescence. For convenience, we view edges of (V, E) as *white* edges, and arcs of (V, A) as *red* arcs.

A *separation* of a bi-tree is a triple (v, X, Y) such that:

- V is partitioned into nonempty sets $\{v\}$, X and Y,
- no white edge has an end in X and an end in Y, and
- no red arc has an end in X and an end in Y.

When the sets X and Y are clear from the context, we will simply say that v is a separation. Note that if (v, X, Y) is a separation of a bi-tree (V, E, A), then $(X \cup \{v\}, A \cap (X \cup \{v\})^2, E \cap \binom{X \cup \{v\}}{2})$ is the bi-tree *induced by* $T \setminus Y$. Observe that if the root is not in X, then $T \setminus Y$ is rooted at v.

Let $T = (V, A, E)$ be a bi-tree and a, b, v be three distinct vertices of V. Let P_{ab} be a white path from a to b, of length one or two. Let P_{av} be a directed red path, from a to v, of length at least one. Let P_{bv} be a directed red path, from b to v, of length at least one. We suppose that the three paths are internally vertex disjoint (meaning that if a vertex is in at least two of the paths, then it must be a, b or v). Three such paths are said to form an *obstruction directed to* v.

Let $T = (V, A, E)$ be a bi-tree and a, b, c, d be four distinct vertices of V. Let P_{ab} be a white path from a to b, P_{bc} be a red path which is directed from b to c or from c to b, P_{cd} be a white path from c to d and P_{da} be a red path which is directed from d to a or from a to d. Suppose that at least one of P_{ab}, P_{cd} has length exactly one and that the four paths are internally vertex disjoint. Four such paths are said to form an *alternating obstruction*.

A *bi-path* is a bi-tree $T = (V, A, E)$ on at least two vertices with an ordering $v_1, \ldots, v_n$ of V and an integer t such that:

- $A = \{v_1 v_2, \ldots, v_{n-1} v_n\}$,
- $v_1 v_n \in E$,
- $1 \leq t \leq n-1$,
- if $t \geq 2$, then $\{v_1 v_2, \ldots, v_1 v_t\} \subseteq E$, and
- if $t \leq n-2$, then $\{v_{t+1} v_n, \ldots, v_{n-1} v_n\} \subseteq E$.

▶ **Lemma 3.** *A bi-tree $T = (V, A, E)$ on at least two vertices, with no separation, no directed obstruction and no alternating obstruction is a bi-path.*

Proof. *Case 1*: (V, A) contains some vertex with in-degree at least 2.

We choose such a vertex v as close as possible to the root r of (V, A). Since (V, A) is an in-arborescence, $(V, A) \setminus v$ has at least $m \geq 2$ in-components $A_1, \ldots, A_m$ and possibly one out-component B. By the choice of v, every vertex of B has in-degree exactly 1. Therefore $(B \cup \{v\}, A \cap (B \cup \{v\})^2)$ is a directed red path from v to r, that we call Z. We now state and prove two claims.

▷ Claim 4. For any $1 \leq i < j \leq m$, there is no white edge with one end in A_i and one end in A_j.

Proof. Indeed, such an edge would yield an obstruction directed to v. ◁

▷ Claim 5. For every $1 \leq i \leq m$, there exists a white edge with one end in A_i and one end in B (so, in particular, B exists).

Proof. For otherwise, Claim 4 implies that $(v, A_i, V \setminus (A_i \cup \{v\})$ is a separation. ◁

Let $P = v, \ldots, z$ be the shortest white path such that $z \in B$ where all internal vertices of P are in $A_1 \cup \cdots \cup A_m$ (P has possibly length 1). By Claim 4, P contains vertices from at most one component, say possibly A_2, among $A_1, \ldots, A_m$. By Claim 5, there exists a vertex $x \in A_1$ with a white neighbor w in B. Let Q be the directed red path from x to v.

If w is an internal vertex of vZz then the edge xw, the directed path wZz, the path P, and the directed path Q form an alternating obstruction. If w is a vertex of zZr different from z, then the edge xw, the directed path zZw, the path P, and the directed path Q form an alternating obstruction. If follows that $w = z$.

If P has length greater than 1, then in particular z has a white neighbor y in A_2. Now, the white path xzy and the in-components A_1 and A_2 yield an obstruction directed to v. So, P has length 1. Consider, by Claim 5, a vertex y' in A_2 with a neighbor in B. The previous argument, with A_1 and A_2 interchanged, shows that y' is adjacent to z (just as we proved that x is adjacent to z). Again, the white path xzy' and the red in-components A_1 and A_2 yield an obstruction directed to v.

Case 2: Every vertex in (V, A) has in-degree at most 1.

Since (V, A) is an in-arborescence, it follows that (V, A) is a directed path. Hence, there exists an ordering $v_1, \ldots, v_n$ of the vertices of T such that $A = \{v_1v_2, \ldots, v_{n-1}v_n\}$.

Suppose that there exists a white edge v_iv_j with $1 < i < j < n$. Then there exists a white edge $v_{i'}v_k$ between $\{v_1, \ldots, v_{i-1}\}$ and $\{v_{i+1}, \ldots, v_n\}$ for otherwise $(v_i, \{v_1, \ldots, v_{i-1}\}, \{v_{i+1}, \ldots, v_n\})$ is a separation. If $k < j$ there is an alternating obstruction, and also if $k > j$. It follows that $k = j$. We proved that there exists a white edge $v_{i'}v_j$, with $i' < i$. By a symmetric argument, we can prove that there exists $j' > j$ and a white edge $v_iv_{j'}$. Now, the white edges $v_{i'}v_j$, $v_iv_{j'}$ and the red paths $v_{i'} \ldots v_i$ and $v_j \ldots v_{j'}$ form an alternating obstruction.

Thus there is no white edge v_iv_j with $1 < i < j < n$. Hence, every white edge is incident to v_1 or to v_n. If there exist two white edges v_1v_j and v_iv_n with $1 < i < j < n$, there is an alternating obstruction, again a contradiction. Hence, if we define t as the greatest integer in $\{2, \ldots, n-1\}$ such that v_1 is adjacent to v_t in (V, E) (with $t = 1$ if v_1 has no white neighbor among $v_2, \ldots, v_{n-1}$), we have that v_n has no white neighbor among $\{v_2, \ldots, v_{t-1}\}$. Since every vertex has a white neighbor, it follows that v_1 is white-complete (complete in (V, E))

to $\{v_2, \ldots, v_t\}$ (when $t \geq 2$). For the same reason, v_n is white-complete to $\{v_{t+1}, \ldots, v_{n-1}\}$ (when $t \leq n-2$).

If $t > 1$ and $v_t v_n$ is a white edge, then $(v_t, \{v_1, \ldots, v_{t-1}\}, \{v_{t+1}, \ldots, v_n\})$ is a separation. So, if $t > 1$ then $v_1 v_n$ is a white edge, and also if $t = 1$. ◀

Given two bi-trees T_1, T_2 and a vertex v of T_1, we denote by (T_1, v, T_2) the bi-tree obtained by *gluing* T_2 at v on T_1, i.e., by identifying the root of T_2 with v. A *bi-spider* is a bi-tree which is obtained by iteratively gluing bi-paths at the root vertex (see Figure 1, right; a bi-spider is induced by the set $\{1, 3, 4, 7, 5\}$). Alternatively, a bi-spider is a bi-tree with no directed obstruction and no alternating obstruction, which is either a bi-path or has only the root as a separation vertex.

Let T be a bi-tree with no directed obstruction and no alternating obstruction. Note that the previous lemma asserts that T can be obtained by iteratively gluing bi-paths. Indeed, a separation v which is chosen as far as possible from the root must isolate a bi-path.

Consider a vertex v of a bi-tree T. Since T can be obtained by iteratively gluing bi-paths, if v is not a separation then it is a vertex in T which is not used in gluing. Thus, the following property holds for T: every vertex v which is not the root is either a separation vertex, a leaf of the white tree, or a leaf of the red in-arborescence. We use it to obtain the following result:

▶ **Corollary 6.** *A bi-tree $T = (V, A, E)$ on at least two vertices, with no directed obstruction and no alternating obstruction is either a bi-spider, or admits a separation (v, X, Y) such that*

(a) *$T \setminus Y$ is a bi-spider,*

(b) *v is either a leaf of the red in-arborescence induced by $T \setminus X$ or a leaf of the white tree induced by $T \setminus X$.*

Proof. If $T = (V, A, E)$ is not a bi-spider, it has a separation (v, X, Y) distinct from the root, and we assume that among all choices, v is chosen as far as possible from the root r of the red in-arborescence. W.l.o.g., we assume that Y contains r. Then $T \setminus Y$ is a bi-tree rooted at v which can only admit v as a separation. Hence, $T \setminus Y$ is a bi-spider. Assume moreover that Y is chosen minimum by inclusion for this property (equivalently, $T \setminus Y$ is a maximum bi-spider rooted at v). We claim that v is not a separation in bi-tree $T \setminus X$. If v is a separation in $T \setminus X$ isolating a bi-path, then we have a contradiction to the minimality of Y. If v is a separation not isolating a bi-path, then we have a contradiction to the choice of v. Hence, $T \setminus X$ is a bi-tree in which v is not a separation. Since v is not the root either, it follows that v is a white leaf or a red leaf in $T \setminus X$. ◀

▶ Note 7. A separation isolating a bi-spider with the properties (a) and (b) can be found efficiently. In particular, we find a separation (v, X, Y) isolating a path and then take the maximal (inclusion-wise) set X such that $T \setminus Y$ is still a bi-spider.

4 The end of the proof

We now resume our proof of Theorem 2 as follows. Lemma 8 shows that the bi-trees arising from even-hole-free graphs do not have the obstructions. Hence, we can use the results from Section 3 where we proved that a bi-tree is either a bi-spider or has a separation isolating a bi-spider. Lemma 9 gives an algorithm for the problem when the underlying bi-tree is a bi-spider. When the bi-tree is obtained by gluing bi-spiders, Lemma 13 proves that combining the partial solutions for each of the bi-spiders produces a valid solution.

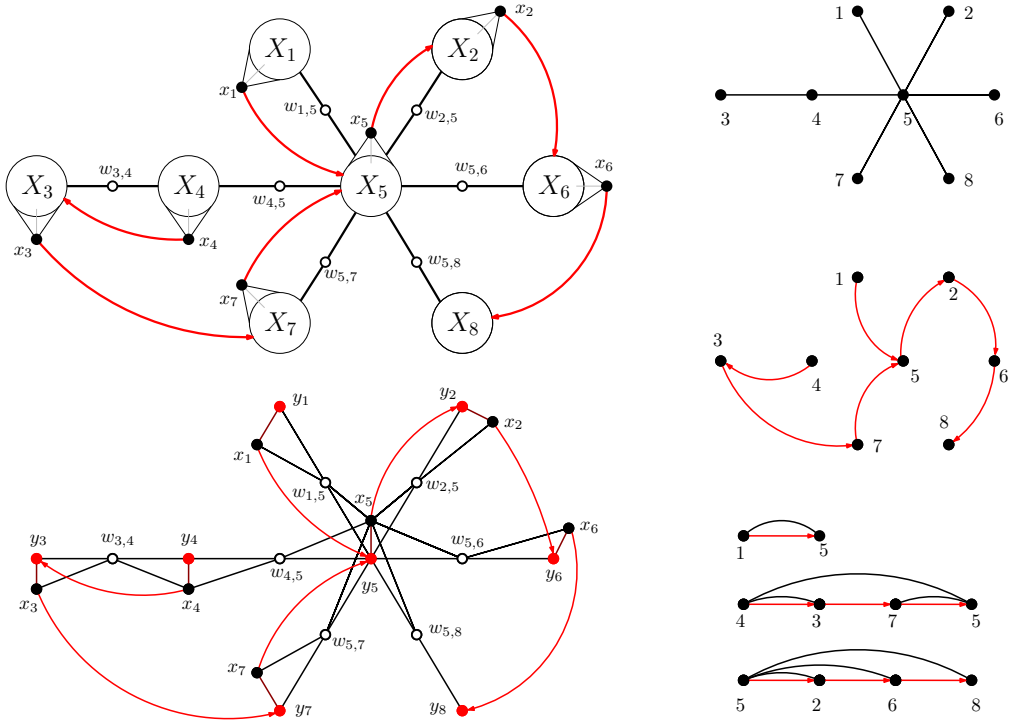


Figure 1 Up-left: Graph G. Down-left: Set of y_i's. Up-right: White tree. Middle-right: Red in-arborescence. Down-right: Decomposition of bi-tree into bi-paths.

Let us recall the hypothesis of Theorem 2 (see Figure 1):

1. The set of vertices of G is partitioned into $k+1$ cliques $X_1, \ldots, X_{k+1}$ and an additional set W consisting of k vertices $w_{a_1b_1}, \ldots, w_{a_kb_k}$.
2. Every $w_{a_ib_i}$ is completely joined to the two parts X_{a_i} and X_{b_i} and has no neighbor in the other parts.
3. The set of pairs $E = \{\{a_i, b_i\} : i = 1, \ldots, k\}$, seen as edges on the vertex set $V = \{1, \ldots, k+1\}$, forms a white tree in which $k+1$ is a leaf.
4. Every X_i, with $1 \leq i \leq k$ contains a particular vertex x_i.
5. The set $\{x_1, \ldots, x_k\}$ is an independent set.
6. For every vertex x_i, there is some $r(i) \neq i$ such that x_i is completely joined to $X_{r(i)} \setminus x_{r(i)}$ (which is just $X_{r(i)}$ when $r(i) = k+1$).
7. The vertex x_i is non-adjacent to every vertex of X_j, when $j \neq i$ or $j \neq r(i)$.
8. The set of ordered pairs $A = \{(i, r(i)) : i = 1, \ldots, k\}$, seen as arcs on the vertex set $V = \{1, \ldots, k+1\}$, forms a red in-arborescence rooted at $k+1$.

We then have a bi-tree $T = (V, E, A)$ on the vertex set $V = \{1, \ldots, k+1\}$. Furthermore, we want to decide if every part X_i, with $1 \leq i \leq k+1$ contains a particular vertex y_i distinct from x_i and such that the set of these y_i's forms an independent set.

▶ **Lemma 8.** *If G has no even holes and a set Y exists, then $T = (V, E, A)$ has no directed obstruction and no alternating obstruction.*

Proof. Let us assume that we have a directed obstruction, i.e., we have three distinct vertices a, b, v of V, a white path P_{ab} from a to b of length one or two, a directed red path P_{av} of the form $a = a_0, a_1, \ldots, a_r = v$, and a directed red path P_{bv} of the form $b = b_0, b_1, \ldots, b_s = v$.

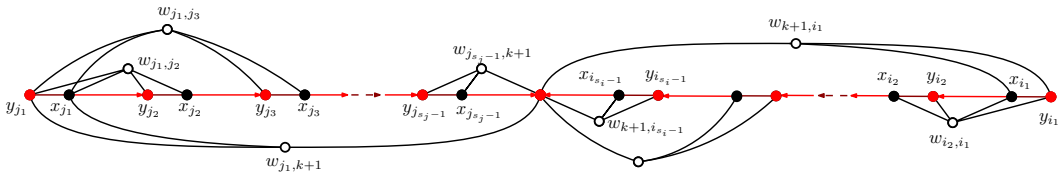


Figure 2 An example for Lemma 9.

Our goal is to exhibit an even hole in G. The path P_{ab} is either ab or acb and corresponds in G to the path P_1 which is either x_a, w_{ab}, x_b or $x_a, w_{ac}, y_c, w_{cb}, x_b$. The path corresponding to P_{av} is $P_2 = x_{a_0}, y_{a_1}, x_{a_1}, \ldots, y_{a_r}$ and the path corresponding to P_{bv} is $P_3 = x_{b_0}, y_{b_1}, x_{b_1}, \ldots, y_{b_s}$. Note that $C = P_1 \cup P_2 \cup P_3$ is an even length cycle. Moreover, since each x_i in C is complete to only one class X_j apart from its own, there is no chord in C, a contradiction.

Let us assume that we have an alternating obstruction on four distinct vertices a, b, c, d of V. Two cases arise depending of the direction of the two red paths. When their directions are the same, we have a white path P_{ab} from a to b, a red path P_{bc} directed from b to c, a white path P_{cd} from c to d, and a red path P_{ad} directed from a to d. By definition of alternating obstruction the four paths are internally vertex disjoint. Assuming that P_{ab} is of the form $a = a_0, a_1, \ldots, a_r = b$, we consider in G the corresponding path $P_1 = x_{a_0}, w_{a_0a_1}, y_{a_1}, w_{a_1a_2}, y_{a_2}, w_{a_2a_3}, \ldots, x_{a_r}$. Assuming that P_{bc} is of the form $b = b_0, b_1, \ldots, b_s = c$, we consider in G the corresponding path $P_2 = x_{b_0}, y_{b_1}, x_{b_1}, \ldots, y_{b_s}$. Assuming that P_{ad} is of the form $a = d_0, d_1, \ldots, d_u = d$, we consider in G the corresponding path $P_3 = x_{d_0}, y_{d_1}, x_{d_1}, \ldots, y_{d_u}$. Finally, if P_{cd} is of the form $c = c_0, c_1, \ldots, c_v = d$, we consider in G the corresponding path $P_4 = y_{c_0}, w_{c_0c_1}, y_{c_1}, w_{c_1c_2}, y_{c_2}, \ldots, y_{c_v}$.

When the red paths are in the opposite direction; we have a white path P_{ab} from a to b, a red path P_{bc} directed from b to c, a white path P_{cd} from c to d and a red path P_{da} directed from d to a. Again, the four paths are internally vertex disjoint. Assuming that P_{ab} is of the form $a = a_0, a_1, \ldots, a_r = b$, we consider in G the corresponding path $P_1 = y_{a_0}, w_{a_0a_1}, y_{a_1}, w_{a_1a_2}, y_{a_2}, w_{a_2a_3}, \ldots, x_{a_r}$. Assuming that P_{bc} is of the form $b = b_0, b_1, \ldots, b_s = c$, we consider in G the corresponding path $P_2 = x_{b_0}, y_{b_1}, x_{b_1}, \ldots, y_{b_s}$. Assuming that P_{da} is of the form $d = d_0, d_1, \ldots, d_u = a$, we consider in G the corresponding path $P_3 = x_{d_0}, y_{d_1}, x_{d_1}, \ldots, y_{d_u}$. Finally, if P_{cd} is of the form $c = c_0, c_1, \ldots, c_v = d$, we consider in G the corresponding path $P_4 = y_{c_0}, w_{c_0c_1}, y_{c_1}, w_{c_1c_2}, y_{c_2}, \ldots, x_{c_v}$.

Note that both P_1, P_4 are even length paths, and P_2, P_3 are odd length. Consequently $C = P_1 \cup P_2 \cup P_3 \cup P_4$ is an even length cycle. Moreover, no chord can arise so C is an even hole, a contradiction. ◀

By Corollary 6, the bi-tree $T = (V, E, A)$ is either a bi-spider, or has a separation i isolating a bi-spider. We first conclude in the case of bi-spiders.

▶ **Lemma 9.** *If T is a bi-spider then there is an $O(n^3)$ time algorithm which computes Y or shows that Y does not exist.*

Proof. Recall that a bi-spider is a graph obtained by iteratively gluing bi-paths at the root vertex. Denote with $T_1, \ldots, T_l$ the bi-paths glued at the root vertex $k+1$ to obtain T. Moreover, assume that the in-arborescence T_j is a directed path $j_1, \ldots, j_{s_j} = k+1$ for $1 \leq j \leq l$. Since each T_j is a bi-path, there is a vertex $w_{j_1,j_{s_j}}$ and for some value $t_j \in \{2, \ldots, s_j\}$ (if any) we have the vertices $\{w_{j_1,j_2}, \ldots, w_{j_1,j_{t_j}}\}$ and $\{w_{j_{t_j+1},j_{s_j}}, \ldots, w_{j_{s_j-1},j_{s_j}}\}$ (see Figure 2).

We decide if Y exists in two phases. First, for every $1 \leq j \leq l$ we find the set Y_{j_1} of all vertices y_{j_1} which are contained in an independent set of size t_j intersecting $X_{j_1}, \ldots, X_{j_{t_j}}$. (Intuitively, Y_{j_1} is the of vertices which can be extended to an independent set traversing $X_{j_1}, \ldots, X_{j_{t_j}}$, i.e., all the parts that have a common white neighbor with y_{j_1} except X_{k+1}.) Clearly, if Y_{j_1} is empty for some j then the set Y does not exist.

Secondly, we find the set Y_{k+1} of vertices y_{k+1} which are contained in an independent set of size $k - \sum_{j=1}^{l} t_j$ intersecting Y_{j_1} and $X_{j_{t_j+1}}, \ldots X_{j_{s_j-1}}$ for all $1 \leq j \leq l$. (Intuitively, Y_{k+1} is the set of vertices which can be extended to an independent set traversing all the parts that have a common white neighbor with y_{k+1}.) Again, if Y_{k+1} is empty then the set Y does not exists.

We first assume that we have the sets Y_{j_1}'s and Y_{k+1} and show how to conclude the lemma in this case. Later, we show that the sets are easy to find. Let $y_{k+1} \in Y_{k+1}$ and let $J = \{y_{k+1}\} \cup_{j=1}^{l} \{y_{j_1}\} \cup_{j=1}^{l} \{y_{j_{t_j+1}}, \ldots, y_{j_{s_j-1}}\}$ be an independent set of size $k - \sum_{j=1}^{l} t_j$ intersecting all Y_{j_1} and $X_{j_{t_j+1}}, \ldots X_{j_{s_j-1}}$. For each $1 \leq j \leq l$, denote with $I_j = \{y_{j_1}, \ldots, y_{j_{t_j}}\}$ an independent set which contains y_{j_1} and intersects $X_{j_1}, \ldots, X_{j_{t_j}}$. Observe that the set $Y = J \cup_{j=1}^{l} I_j$ intersects each part of the graph. It suffices to prove the following claim.

▷ Claim 10. $J \cup_{j=1}^{l} I_j$ is also an independent set.

Proof. For the sake of contradiction suppose otherwise. We consider two cases. Either there is an edge with one end in J and the other end in I_j for some j, or there is an edge with ends in I_j and I_i for some $1 \leq i < j \leq l$. Let us deal with them respectively.

The mentioned edge is of the form $y_{j_p} y_{i_q}$ where $p \leq t_j$ and $t_i < q$ (possibly $j = i$) by definition of I_j and J. Choose smallest such p. Observe that $p \neq 1$ since y_{j_1} is a vertex of both J and I_j. If $i_q \neq k+1$ then

$$y_{j_p}, x_{j_{p-1}}, \ldots, y_{j_2}, x_{j_1}, w_{j_1,j_{s_j}} (= w_{j_1,k+1}), y_{k+1}, w_{k+1,i_q} (= w_{i_{s_i},i_q}), y_{i_q}$$

is a cycle of even length. Moreover, the cycle is induced by the choice of p and since $\{y_{j_2}, \ldots, y_{j_p}\}$ is an independent set, a contradiction. An analogous situation arises if $i_q = k+1$.

Now, we deal with the second case where there is an edge $y_{j_p} y_{i_q}$ where $p \leq t_j$, $q \leq t_i$ and $j \neq i$. Choose largest such q. It might happen that $p = 1$ or $q = 1$, but not both since $y_{j_1}, y_{i_1} \in J$. Without loss of generality, $p \neq 1$. Then

$$y_{j_p}, x_{j_{p-1}}, \ldots, y_{j_2}, x_{j_1}, w_{j_1,j_{s_j}} (= w_{j_1,k+1}), y_{k+1}, x_{i_{s_i-1}}, y_{i_{s_i-1}}, \ldots, x_{i_q}, y_{i_q}$$

is an even cycle. By the previous case there is no edge between y_{k+1} and $I_j \cup I_i$ Moreover, by the choice of q, we deduce that the even cycle is induced, a contradiction. ◁

It remains to show how to find the sets Y_{j_1}'s and Y_{k+1}. For the rest of the proof we only use the white tree. Observe that it suffices to prove the following (by setting $p = j_1$ for all j and then $p = k+1$).

▷ Claim 11. Let $y_p \in X_p$ and let G' be the graph induced by X_i such that pi is an edge in the bi-tree T. Remove neighbors of y_p in G'. Then G' is chordal.

Proof. For a contradiction, assume that H is an odd hole in G'. Each part of G' is a clique and, thus, contains at most two vertices of H. Therefore, there exist an induced path on three vertices y_a, y_b, y_c of H, with y_a, y_b, y_c in different parts X_a, X_b, X_c. By construction there are vertices $w_{p,a}, w_{p,b}$ and $w_{p,c}$. Then $y_p, w_{p,a}, y_a, y_b, y_c, w_{p,c}$ induces an even hole in G, a contradiction. Since G is even-hole-free so is G'. Hence G' is hole-free. ◁

Now, for each j, we can check if y_{j_1} is in Y_{j_1} by finding a maximum independent set in $G' = G[\cup_{i=2}^{t_j} X_i] \setminus N(y_{j_1})$. The latter can be done in $O(n^2)$ since G' is chordal [6]. Then, we can check if y_{k+1} is in Y_{k+1} by finding a maximum independent set in $G' = G[\cup_j \{Y_{j_1} \cup_{i=t_j+1}^{s_j-1} X_i\}] \setminus N(y_{k+1})$. This can be done in $O(n^2)$ since G' is chordal. The overall running time follows since each part is used exactly once in some G'. ◀

In fact, the previous algorithm gives a stronger result:

▶ **Corollary 12.** *When T is a bi-spider, there is an $O(n^3)$ time algorithm which computes all vertices y_{k+1} which belong to an independent set of size $k+1$.*

We now deal with the case when i is a separation isolating a bi-spider. By Corollary 6 bi-tree T admits a separation (i, B, C) isolating a bi-spider $T \setminus C$ such that i is either a red leaf or a white leaf in $T \setminus B$. Recall that the vertex $k+1$ is a leaf of the white tree, hence, as a separation, i is not equal to $k+1$. In particular, the vertex x_i exists. Moreover, since $T \setminus C$ is a bi-spider it follows that $k+1 \in C$. As before, assuming the set Y exists, we obtain the following lemma.

▶ **Lemma 13.** *There is no edge from some y_j with $j \in B \setminus i$ to some vertex $u \in X_s$ with $s \in C$.*

Proof. We denote by r the root of T. As argued above $r \in C$ ($r = k+1$). For the sake of contradiction suppose that there is an an edge $y_j u$.

Let us consider bi-spider $T \setminus C$. There is a red path $j = j_0, \ldots, j_a = i$ in (V, A) which can be turned into an induced path $P_0 = y_{j_0}, x_{j_0}, y_{j_1}, x_{j_1}, \ldots, y_{j_a}, x_{j_a}$ in G from y_j to x_i with odd length. There is also a white path $j = b_0, \ldots, b_d = i$ in (V, E) which can be turned into an induced path $P_1 = y_{b_0}, w_{b_0 b_1}, y_{b_1}, w_{b_1 b_2}, \ldots, x_{b_d}$ in G from y_j to x_i with even length. Now, in order to conclude the lemma it suffices to find a u, x_i path P such that $P.P_0$ and $P.P_1$ induce cycles. Then, since P_0 and P_1 are of different parity a contradiction arises. In the rest of the proof we show how to find P.

First, observe that since $T \setminus C$ contains a white subtree, u is non-adjacent to y_i or to any y_q where $q \in B$ and $q \neq j$ since it would yield an even hole (there is an even path between any two different vertices y_p, y_q). Hence, u is adjacent to y_j and non-adjacent to all other vertices in P_0 and P_1.

By Corollary 6, i is either a red leaf or a white leaf in $T \setminus B$. We consider two cases.

Case 1: i is a red leaf. Then there is a (an undirected) red path $i = i_0, \ldots, i_s = s$ in $T \setminus B$, which can be turned into an induced path $P = x_{i_0}, y_{i_1}, x_{i_1}, \ldots, x_{i_{s-1}}, u$ in G from x_i to u. By construction, this path is induced. Moreover, since i is a red leaf in $T \setminus B$ it follows that $y_{i_1} \neq y_i$. Therefore, both $P.P_0$ and $P.P_1$ induce cycles, i.e., there is no chord with one end in P and the other in P_0 or P_1.

Note that the same argument holds whenever the red path $i = i_0, \ldots, i_s = s$ does not contain y_i. Hence, the only remaining case is when i is on the red directed path from s to r in $T \setminus B$. Denote with Q the directed red sr path in $T \setminus B$.

Case 2: i is a white leaf and $i \in Q$. Let $Q' = iQr$ be subpath of Q starting at i and ending at r. Since i is not a separation of $T \setminus B$, there exists a white path $s = s_0, \ldots, s_t$ connecting s and Q'. Moreover, the path does not contain i. We choose the shortest such s, Q' path. This path can be turned into an induced path $P_2 = u, w_{s_0,s_1}, y_{s_1}, \ldots, w_{s_{t-1},s_t}, y_{s_t}$ in G with endpoints u and y_{s_t}. By the above $y_i \notin P_2$ and also no vertex $w_{i,\cdot}$ is used in P_2.

Consider the directed path iQs_t ($= iQ's_t$). Denote it as $i = i_0, i_1, \ldots, i_l = s_t$. It can be turned into an induced path $P_3 = x_{i_0}, y_{i_1}, \ldots y_{i_l}$ in G with endpoints x_i and y_{s_t}. Then $P_2.P_3$

is a u, x_i path in G. The concatenation $P_2.P_3$ might not be an induced path, but we can shorten it to obtain an induced ux_i path P in G. Now, it can be checked that $P.P_0$ and $P.P_1$ induce cycles since P does not use y_i or any of the vertices $w_{i,.}$. ◀

We are now ready to show that there is an $O(k \cdot n^3)$ time algorithm which computes Y when $T = (V, E, A)$ is a bi-tree. If T is a bi-spider, we are done by Lemma 9. Otherwise, by Corollary 6, there is a separation (i, B, C) which isolates a bi-spider $T \setminus C$. By Lemma 13, one can delete all vertices $y_j \in X_j$ for $j \in B \setminus i$ with a neighbor $u \in X_k$ for $k \notin B$, and this reduction is sound since no candidate y_j can have such an edge. Now, by Corollary 12, one can compute in $O(n^3)$ time the set $X'_i \subseteq X_i$ of vertices, each of which extends, in the bi-spider $T \setminus C$, to an independent set of size $|B|$. From the bi-spider $T \setminus C$, we only keep these vertices X'_i. Observe that the number of parts has now decreased by at least one. We repeat this process until we either construct X'_{k+1} or conclude that this set is empty. If $X'_{k+1} \neq \emptyset$, then we can reconstruct the set Y. The total time is $O(k \cdot n^3)$. ◀

References

1 VE Alekseev. The effect of local constraints on the complexity of determination of the graph independence number. *Combinatorial-algebraic methods in applied mathematics*, pages 3–13, 1982.

2 Édouard Bonnet, Nicolas Bousquet, Pierre Charbit, Stéphan Thomassé, and Rémi Watrigant. Parameterized Complexity of Independent Set in H-Free Graphs. 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 17:1–17:13, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

3 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990.

4 Rodney G. Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

5 Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

6 Fănică Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

7 Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

8 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

9 George J Minty. On maximal independent sets of vertices in claw-free graphs. *Journal of Combinatorial Theory, Series B*, 28(3):284–304, 1980.

10 Svatopluk Poljak. A note on stable sets and colorings of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 15(2):307–309, 1974.

11 Najiba Sbihi. Algorithme de recherche d'un stable de cardinalité maximum dans un graphe sans étoile. *Discrete Mathematics*, 29(1):53–76, 1980.

12 Kristina Vušković. Even-hole-free graphs: a survey. *Applicable Analysis and Discrete Mathematics*, pages 219–240, 2010.

Improved Analysis of Highest-Degree Branching for Feedback Vertex Set

Yoichi Iwata
National Institute of Informatics, Tokyo, Japan
yiwata@nii.ac.jp

Yusuke Kobayashi
Kyoto University, Kyoto, Japan
yusuke@kurims.kyoto-u.ac.jp

Abstract

Recent empirical evaluations of exact algorithms for Feedback Vertex Set have demonstrated the efficiency of a highest-degree branching algorithm with a degree-based pruning heuristic. In this paper, we prove that this empirically fast algorithm runs in $O(3.460^k n)$ time, where k is the solution size. This improves the previous best $O(3.619^k n)$-time deterministic algorithm obtained by Kociumaka and Pilipczuk (Inf. Process. Lett., 2014).

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Feedback Vertex Set, Branch and bound, Measure and conquer

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.22

Funding *Yoichi Iwata*: Supported by JSPS KAKENHI Grant Number JP17K12643.
Yusuke Kobayashi: Supported by JSPS KAKENHI Grants Number JP16K16010, 17K19960, and 18H05291.

Acknowledgements We would like to thank Yixin Cao for valuable discussions and thank organizers of PACE challenge 2016 for motivating us to study FVS.

1 Introduction

Feedback Vertex Set (FVS) is a problem of finding the minimum-size vertex deletion set to make the input graph a forest, which is one of the Karp's 21 NP-complete problems [18]. It is known that this problem is *fixed-parameter tractable (FPT)* parameterized by the solution size k [2, 10]; i.e., we can find a deletion set of size k in $O^*(f(k))$[1] time for some function f. FVS is one of the most comprehensively studied problems in the field of parameterized algorithms, and various FPT algorithms using different approaches have been developed, including short-cycle branching [11], highest-degree branching [4], iterative-compression branching [5, 6, 20], LP-guided branching [16, 17], cut-and-count dynamic programming [7], and random sampling [1].

The current fastest deterministic FPT algorithm for FVS is a branching algorithm combined with the iterative compression technique [20] which runs in $O^*(3.619^k)$ time. When allowing randomization, the current fastest one is a cut-and-count dynamic programming algorithm [7] which runs in $O^*(3^k)$ time. In this paper, we give a faster deterministic algorithm which runs in $O^*(3.460^k)$ time. As explained below, this study is strongly motivated by

[1] The $O^*(\cdot)$ notation hides factors polynomial in n. Note that for FVS, any $O(f(k)n^{O(1)})$-time FPT algorithms can be improved to $O(f(k)k^{O(1)} + k^{O(1)}n)$ time by applying a linear-time polynomial-size kernel [15] as a preprocess. We can therefore focus only on the $f(k)$ factor when comparing the running time.

Parameterized Algorithms and Computational Experiments (PACE) challenge and its follow-up empirical evaluation by Kiljan and Pilipczuk [19]. Instead of designing a new theoretically fast algorithm, we analyze the theoretical worst-case running time of an empirically fast algorithm that has been developed through the PACE challenge and the empirical evaluation, and we show that this algorithm is not only empirically fast but also theoretically fast.

PACE challenge is an annual programming challenge started in 2016. Due to the importance of FVS in the field, FVS was selected as the subject of track B in the first PACE challenge [9]. Although in theoretical studies, the current fastest algorithm is the randomized cut-and-count dynamic programming, the result of the challenge suggests that branching is the best choice in practice. This is not so surprising; because the theoretical analysis of branching algorithms is difficult, the proved upper bound of the running time is not so tight, and moreover, the worst-case instances for branching algorithms are usually very rare in practice.

Among seven submissions to the PACE challenge, top six submissions used branching algorithms; the first used an LP-guided branching; the second and third used branching on highest-degree vertices; the fourth and sixth used branching combined with iterative compression; and the fifth used branching on short cycles. In the branching algorithms, we recursively solve the problem by picking a vertex v and branching into two cases: deleting v (the case when v is contained in the solution) or making v undeletable (the case when v is not contained in the solution). Because a forest has average degree less than one, in order to obtain a solution of size k, the graph must contain k high-degree deletable vertices. Based on this observation, in addition to the pruning by the LP lower bound, the first-place solver by Imanishi and Iwata [14] used the following *degree-based pruning* heuristic:

▶ **Lemma 1** ([14]). *Given a graph $G = (V, E)$ and a set of undeletable vertices $F \subseteq V$, let $v_1, v_2, \ldots$ be the vertices of $V \setminus F$ in the non-decreasing order of the degrees $d(v_i)$ in G. If $k \leq |V \setminus F|$ and $|E| - \sum_{i=1}^{k} d(v_i) \geq |V| - k$ holds, there is no feedback vertex set $S \subseteq V \setminus F$ of size k.*

The follow-up empirical evaluation [19] shows that the use of the degree-based pruning is much more important than the choice of branching rules. By combining with the degree-based pruning, the performances of the LP-guided branching [16], the highest-degree branching [4], and the iterative-compression branching [20], are all significantly improved, and among them, the highest-degree branching slightly outperforms the others. Cao [4] showed that one can stop the highest-degree branching at depth $3k$ by using a degree-based argument, and therefore the running time is $O(8^k)$. On the other hand, the theoretically proved running time of other branching algorithms (without the degree-based pruning) are $O^*(4^k)$ for the LP-guided branching [16] and $O^*(3.619^k)$ for the iterative-compression branching [20]. These affairs motivated us to refine the analysis of the highest-degree branching with the degree-based pruning.

In our analysis, instead of bounding the depth of the search tree as Cao [4] did, we design a new measure to bound the size of the search tree. The measure is initially at most k and we show that the measure drops by some amount for each branching. In contrast to the standard analysis of branching algorithms, our measure has a negative term and thus can have negative values; however, we show that we can immediately apply the degree-based pruning for all such cases. A simple analysis already leads to an $O^*(4^k)$-time upper bound which significantly improves the $O^*(8^k)$-time upper bound obtained by Cao [4]. We then apply the computer-aided measure-and-conquer analysis [12] and improve the upper bound to $O^*(3.460^k)$.

Algorithm 1 Highest-degree branching algorithm with a degree-based pruning.

1: **if** $k < 0$ **then return** No.
2: **if** G is a forest **then return** Yes.
3: Apply reduction rules 1–6.
4: Apply the degree-based pruning.
5: Apply the highest-degree branching.

1.1 Organization

Section 2 describes the highest-degree branching algorithm with the degree-based pruning. In Section 3, we analyze the running time of the algorithm. We first give a simple analysis in Section 3.1 and then give a computer-aided measure-and-conquer analysis in Section 3.2. While the correctness of the simple analysis can be easily checked, we need to verify thousands of inequalities to check the correctness of the measure-and-conquer analysis. The source code of a program to verify the inequalities is available at `https://github.com/wata-orz/FVS_analysis`.

2 Algorithm

Because our algorithm may introduce multiple edges connecting the same pair of vertices, we deal with multi-graphs. Note that a double edge is also considered as a cycle. Let $G = (V, E)$ be a multi-graph. We define the degree $d(u)$ of vertex u as the total multiplicity of edges incident to u. For subset $U \subseteq V$, $G[U] = (U, E[U])$ denotes the induced subgraph, where $E[U]$ is the subset of E whose two endpoints are in U. For $U \subseteq V$, the contraction of U into a new vertex u is the following operation. We first introduce a new vertex u. For each edge $vw \in E$ with $v \in U$ and $w \notin U$, we insert an edge uw of the same multiplicity (if uw has been already inserted, its multiplicity is increased by the multiplicity of vw). Finally, we delete U and its incident edges from the graph. Note that the contraction does not change degrees of vertices in $V \setminus U$.

Algorithm 1 describes our algorithm. An input to our branching algorithm is a tuple (G, F, k) consisting of a multi-graph $G = (V, E)$, a set of undeletable vertices $F \subseteq V$ such that $G[F]$ forms a forest, and an integer k. Our task is to find a subset of vertices $S \subseteq V \setminus F$ such that $|S| \leq k$ and $G[V \setminus S]$ contains no cycles. For convenience, we use D to denote $\max_{v \in V \setminus F} d(v)$ when G and F are clear from the context (when using D, $V \setminus F$ is ensured to be non-empty). If $k < 0$, we return No, and if G is a forest, we return Yes. Otherwise, we apply reduction rules, the degree-based pruning rule, and the highest-degree branching rule. In the following, we give details of each rule with proof of correctness.

Our algorithm uses exactly the same set of reduction rules used in the empirical evaluation by Kiljan and Pilipczuk [19] listed below in the given order (i.e., rule i is applied only when none of the rules j with $j < i$ are applicable). Recall that D denotes $\max_{v \in V \setminus F} d(v)$.

▶ **Reduction Rule 1.** If there exists a vertex u of degree at most one, delete u.

▶ **Reduction Rule 2.** If there exists a vertex $u \notin F$ such that $G[F \cup \{u\}]$ contains a cycle, delete u and decrease k by one.

▶ **Reduction Rule 3.** If there exists a vertex u of degree two, add an edge connecting the two neighbors of u, and then delete u.

▶ **Reduction Rule 4.** If there exists an edge e of multiplicity more than two, reduce its multiplicity to two.

▶ **Reduction Rule 5.** If there exists a vertex $u \notin F$ incident to a double edge uw with $d(w) \leq 3$, delete u and decrease k by one.

▶ **Reduction Rule 6.** If $D \leq 3$, solve the problem in polynomial time by a reduction to the linear (co-graphic) matroid parity [5, 20].

Note that, in reduction rules 1 and 3, the vertex u might be in F, and in such a case, the reductions also delete u from F. The first four reduction rules are standard and have been used in branching FPT algorithms [4–6, 11, 20], the random sampling FPT algorithm [1], and also polynomial kernels [3, 15] for FVS. Reduction rule 5 was used in the Imanishi–Iwata solver [14] and the empirical evaluation [19], and its correctness can be proved as follows. Because there is a double edge uw, any feedback vertex set must contain at least one of u and w. Because w has at most one edge other than the double edge uw, every cycle containing w also contains u. Therefore, there always exists a minimum feedback vertex set containing u. Reduction rule 6 was introduced by Cao, Chen, and Liu[5] and then was simplified by Kociumaka and Pilipczuk [20].

▶ **Lemma 2.** *After applying the reductions, the following conditions hold.*

1. *G has minimum degree at least three.*
2. *No double edges are incident to F.*
3. *$D \geq 4$.*
4. *For any vertex $v \notin F$, $G - v$ has minimum degree at least two.*

Proof. The first three conditions clearly hold. We show the fourth condition. Suppose that in $G - v$, a vertex u has degree at most one. Because u has degree at least three in G, the set of incident edges to u in G must be a double-edge uv and a single-edge uw for another vertex w. If $u \notin F$, we can apply reduction rule 5, which is a contradiction. If $u \in F$, the double edge uv is incident to F in G, which is also a contradiction. Therefore, when no reductions are applicable, $G - v$ contains no such vertex u. ◀

If none of the reductions are applicable, we apply the following pruning rule.

▶ **Pruning Rule.** If $kD < \sum_{v \in F}(d(v) - 2)$, return NO.

The correctness of the pruning rule follows from the following lemma.

▶ **Lemma 3.** *If the minimum degree of G is at least two and $kD < \sum_{v \in F}(d(v) - 2)$ holds, there is no feedback vertex set $S \subseteq V \setminus F$ of size at most k.*

Proof. Suppose that there is a feedback vertex set $S \subseteq V \setminus F$ of size at most k. We have

$$\begin{aligned} kD - \sum_{v \in F}(d(v) - 2) &\geq \sum_{v \in S} d(v) - \sum_{v \in V \setminus S}(d(v) - 2) \\ &= \sum_{v \in S} d(v) - \sum_{v \in V \setminus S} d(v) + 2|V \setminus S| \\ &= 2|E[S]| - 2|E[V \setminus S]| + 2|V \setminus S| \\ &\geq -2|E[V \setminus S]| + 2|V \setminus S|. \end{aligned}$$

Because $G - S$ is a forest, this must be non-negative. ◀

Note that this pruning is different from the degree-based pruning (Lemma 1) used in the Imanishi–Iwata solver [14] and the empirical evaluation [19]; however, as the following lemma shows, this pruning is applicable only if the original degree-based pruning is applicable. Therefore, we can use the same analysis against the original degree-based pruning. We use this weaker version because it is sufficient for our analysis. We leave whether the stronger version helps further improve the analysis as future work.

▶ **Lemma 4.** *For a subset $F \subseteq V$, let $v_1, v_2, \dots$ be the vertices of $V \setminus F$ in the non-increasing order of the degrees $d(v_i)$ in G. If $k \leq |V \setminus F|$ and the minimum degree of G is at least two, $kd(v_1) < \sum_{v \in F}(d(v) - 2)$ implies $|E| - \sum_{i=1}^{k} d(v_i) \geq |V| - k$.*

Proof. Let $X = \{v_1, \dots, v_k\}$ and assume that $kd(v_1) < \sum_{v \in F}(d(v) - 2)$ holds. We have

$$\begin{aligned} 2\left(|V| - k - |E| + \sum_{v \in X} d(v)\right) &= \sum_{v \in X} d(v) - \left(2|E| - \sum_{v \in X} d(v) - 2(|V| - k)\right) \\ &= \sum_{v \in X} d(v) - \sum_{v \in V \setminus X} (d(v) - 2) \\ &\leq kd(v_1) - \sum_{v \in F} (d(v) - 2) < 0. \end{aligned}$$

◀

Finally, when none of the reduction rules nor the pruning rule are applicable, we apply the highest-degree branching rule.

▶ **Branching Rule.** Pick a vertex $u \in V \setminus F$ of the highest degree D. Let U be the set of neighbors of u that are contained in F and let G' be the graph obtained by contracting $U \cup \{u\}$ into a new vertex u'. We branch into two cases: $(G - u, F, k - 1)$ and $(G', F - U + u', k)$.

We pick a vertex $u \in V \setminus F$ of the highest degree D and branch into two cases: whether u is contained in the solution or not. In the former case, we delete u and decrease k by one, and in the latter case, we make u undeletable by inserting it into F. Because no feedback vertex set $S \subseteq V \setminus F$ can delete edges inside F, if u has neighbors U in F, we can safely contract $U \cup \{u\}$. We use the contraction for simplicity of analysis; instead of using the number of connected components of $G[F]$, we can simply use $|F|$. Note that, by using an additional reduction rule, we can ensure that F forms an independent set; however, we do not use it because our analysis do not require such a strong condition.

▶ **Proposition 5.** *Algorithm 1 correctly solves* FVS.

3 Analysis

For parameters $0 \leq \alpha \leq 1$ and $(\beta_d)_{d=0}^{\infty}$ satisfying $0 = \beta_0 = \beta_1 = \beta_2 \leq \beta_3 \leq \beta_4 \leq \cdots$, we define

$$\mu(G, F, k) := k - \frac{\alpha}{D} \sum_{v \in F} (d(v) - 2) + \sum_{v \in F} \beta_{d(v)}.$$

Initially, the algorithm is called with $F = \emptyset$, and we have $\mu(G, \emptyset, k) = k$.

▶ **Lemma 6.** *If none of the reduction rules 1–6 nor the pruning rule are applicable for an input (G, F, k), then $\mu(G, F, k) \geq 0$ holds.*

Proof. If the pruning is not applicable, we have $kD \geq \sum_{v \in F}(d(v) - 2)$. Hence we have

$$\mu(G, F, k) = (1 - \alpha)k + \frac{\alpha}{D}\left(kD - \sum_{v \in F}(d(v) - 2)\right) + \sum_{v \in F} \beta_{d(v)} \geq 0. \quad \blacktriangleleft$$

We now show that applying the reduction rules does not increase μ. We can easily see that D never increases by the reduction but may decrease; however, because such decrease leads to a smaller μ, we can analyze as if D is not changed by the reduction.

From condition 4 in Lemma 2 and the fact that contraction does not change degrees, the graph immediately after branching has minimum degree at least two. Hence, we apply reduction rule 1 only after applying reduction rules 2 or 5 (or against the initial instance, in which case, μ does not increase because F remains empty).

▶ **Lemma 7.** *Reduction rules 2 or 5, together with the subsequent applications of reduction rule 1, do not increase μ.*

Proof. For convenience of analysis, when a vertex of degree one is generated, we immediately put it into F. This does not affect the behavior of the algorithm because all such vertices will be deleted by reduction rule 1.

We first show that applying reduction rule 1 does not increase μ. When $d(u) = 0$, we have $\mu(G - u, F - u, k) = \mu(G, F, k) - \frac{\alpha}{D}$. When $d(u) = 1$, let v be the neighbor of u. If $v \in F$, we have $\mu(G - u, F - u, k) = \mu(G, F, k) + \beta_{d(v)-1} - \beta_{d(v)} \leq \mu(G, F, k)$. If $v \notin F$ and $d(v) \geq 3$, we have $\mu(G - u, F - u, k) = \mu(G, F, k) - \frac{\alpha}{D}$. If $v \notin F$ and $d(v) = 2$, deleting u puts v into F, and hence we have $\mu(G - u, F - u + v, k) = \mu(G, F, k)$.

We next show that applying reduction rules 2 or 5 does not increase μ. Let P be the set of vertices in $V \setminus (F \cup \{u\})$ whose degree in $G - u$ is one. Let q be the number (i.e., the total multiplicity) of edges between u and F. Because P is a subset of neighbors of u, we have $|P| + q \leq d(u) \leq D$. Then, we have $\mu(G - u, F \cup P, k - 1) \leq \mu(G, F, k) - 1 + \frac{\alpha}{D}(|P| + q) \leq \mu(G, F, k) - 1 + \alpha \leq \mu(G, F, k)$. ◀

Because reduction rule 3 deletes a vertex of degree two and does not change the degrees of other vertices, it does not change μ. Because reduction rule 4 is applied only when reduction rule 2 cannot be applied, it does not change the degrees of vertices in F, and therefore it does not change μ.

Finally, we analyze the branching rule. Because contraction does not change degrees of other vertices, D never increases by the branching but may decrease. As we did in the analysis of the reduction rules, we analyze as if D does not change by the branching. Recall that U is the set of neighbors of u that are contained in F. Let $f := |U|$ and $\mathbf{d} := \{d_1, \ldots, d_f\}$ be the multiset of degrees of vertices in U. The degree of u' is $d' := D + \sum_{i=1}^{f}(d_i - 2)$. In the former case of the branching, we have

$$\begin{aligned}\Delta_1(D, \mathbf{d}) &:= \mu(G, F, k) - \mu(G - u, F, k - 1) \\ &= 1 - f\frac{\alpha}{D} + \sum_{i=1}^{f}(\beta_{d_i} - \beta_{d_i - 1}).\end{aligned}$$

In the latter case, we have

$$\begin{aligned}\Delta_2(D,\mathbf{d}) &:= \mu(G,F,k) - \mu(G', F - U + u', k)\\ &= -\frac{\alpha}{D}\left(\sum_{i=1}^{f}(d_i - 2) - (d' - 2)\right) + \sum_{i=1}^{f}\beta_{d_i} - \beta_{d'}\\ &= \alpha - 2\frac{\alpha}{D} + \sum_{i=1}^{f}\beta_{d_i} - \beta_{d'}.\end{aligned}$$

Now, we can prove the running time of the algorithm by induction on the height of the search tree as follows. Suppose that the number of leaves in the search tree is bounded by $c^{\mu(G,F,k)}$ for some $c > 1$ when the height is at most h. After the branching, we can bound the total number of leaves by $c^{\mu(G,F,k)-\Delta_1(D,\mathbf{d})} + c^{\mu(G,F,k)-\Delta_2(D,\mathbf{d})}$. Hence, in the induction step for $h+1$, we need to show the inequality

$$c^{\mu(G,F,k)-\Delta_1(D,\mathbf{d})} + c^{\mu(G,F,k)-\Delta_2(D,\mathbf{d})} \leq c^{\mu(G,F,k)},$$

which is equivalent to

$$c^{-\Delta_1(D,\mathbf{d})} + c^{-\Delta_2(D,\mathbf{d})} \leq 1.$$

Therefore, if $c^{-\Delta_1(D,\mathbf{d})} + c^{-\Delta_2(D,\mathbf{d})} \leq 1$ holds for any $(D,\mathbf{d})$ for some $c > 1$, the running time of the algorithm is bounded by $O^*(c^k)$. We now optimize the parameters to minimize c.

The design of our measure

We have reached to our measure by the following three steps.

Step 1. Let gap $:= kD - \sum_{v\in F}(d(v) - 2)$, i.e., the distance to the application of the pruning rule. By analyzing the branching rule, we observe that deleting u decreases k but may not decrease gap, while making u undeletable does not change k but decreases gap. Hence, by taking the convex combination $(1-\alpha)\cdot k + \alpha \cdot \frac{1}{D}\text{gap}$, we can ensure that the measure always decreases. The coefficient $\frac{1}{D}$ is for bounding the measure by k.

Step 2. We observe that deleting u decreases gap if u has neighbors not in F. When u has more than two neighbors in F, making u undeletable decreases the size $|F|$. Because $|F|$ cannot be negative, the worst-case (when all the neighbors of u are contained in F) cannot occur frequently. Hence, by taking $|F|$ into the measure, we can improve the analysis. This corresponds to the simple analysis presented in Section 3.1.

Step 3. We observe that applying reduction rule 3 against a vertex in F does not change gap but decreases $|F|$. Deleting u in the branching rule decreases the degree of its neighbors, and if a neighbor is in F, such a decrease leads to a future application of reduction rule 3. Hence, by using the sum of weights $\sum_{v\in F}\beta_{d(v)}$ depending on the degree instead of the size $|F| = \sum_{v\in F} 1$, we can improve the analysis. This corresponds to the computer-aided analysis presented in Section 3.2.

3.1 Simple Analysis

As a simple analysis whose correctness can be easily checked, we use $\alpha = \log_4 \frac{8}{3} \approx 0.7075$, $\beta_d = \frac{1}{2}\log_4 \frac{3}{2} \approx 0.1462$ for all $d \geq 3$, and $c = 4$. Note that, when applying the branching rule, $D \geq 4$ holds from Lemma 2. For these parameters, we have

$$c^{-\Delta_1(D,\mathbf{d})} \leq 4^{-1+\frac{f}{D}\log_4 \frac{8}{3}} = \frac{1}{4}\cdot\left(\frac{8}{3}\right)^{\frac{f}{D}} \leq \frac{1}{4}\min\left(\frac{8}{3}, \left(\frac{8}{3}\right)^{\frac{f}{4}}\right),$$

$$c^{-\Delta_2(D,\mathbf{d})} = 4^{\left(\frac{2}{D}-1\right)\log_4 \frac{8}{3}+(1-f)\frac{1}{2}\log_4 \frac{3}{2}} \leq \left(\frac{3}{8}\right)^{\frac{1}{2}}\cdot\left(\frac{3}{2}\right)^{\frac{1-f}{2}}.$$

We now show that $c^{-\Delta_1(D,\mathbf{d})} + c^{-\Delta_2(D,\mathbf{d})} \leq 1$ holds by the following case analysis.

$$c^{-\Delta_1(D,\mathbf{d})} + c^{-\Delta_2(D,\mathbf{d})} \leq \begin{cases} \frac{1}{4} + \left(\frac{3}{8}\right)^{\frac{1}{2}}\cdot\left(\frac{3}{2}\right)^{\frac{1}{2}} = 1 & (f=0), \\ \frac{1}{4}\cdot\left(\frac{8}{3}\right)^{\frac{1}{4}} + \left(\frac{3}{8}\right)^{\frac{1}{2}} < 0.932 & (f=1), \\ \frac{1}{4}\cdot\left(\frac{8}{3}\right)^{\frac{2}{4}} + \left(\frac{3}{8}\right)^{\frac{1}{2}}\cdot\left(\frac{2}{3}\right)^{\frac{1}{2}} < 0.909 & (f=2), \\ \frac{1}{4}\cdot\left(\frac{8}{3}\right)^{\frac{3}{4}} + \left(\frac{3}{8}\right)^{\frac{1}{2}}\cdot\frac{2}{3} < 0.930 & (f=3), \\ \frac{1}{4}\cdot\frac{8}{3} + \left(\frac{3}{8}\right)^{\frac{1}{2}}\cdot\left(\frac{2}{3}\right)^{\frac{3}{2}} = 1 & (f\geq 4). \end{cases}$$

▶ **Theorem 8.** *Algorithm 1 solves* FVS *in* $O^*(4^k)$ *time.*

3.2 Measure-and-Conquer Analysis

We use the parameters $\alpha = 0.922863$, β shown in Table 1, and $c = 3.460$. These values are obtained by solving a convex optimization problem to minimize c under the constraints of $c^{-\Delta_1(D,\mathbf{d})} + c^{-\Delta_2(D,\mathbf{d})} \leq 1$ for any $(D, \mathbf{d})$. For details of computer-aided measure-and-conquer analysis and how to solve it as a convex optimization problem, see [13, Chapter 2]. For $d \geq d_{\max} := 30$, we fix $\beta_d = \beta_{d_{\max}}$. This is for bounding the number of vectors $\mathbf{d}$ we need to consider. The running time for solving the convex optimization problem depends on the choice of $d_{\max}$, and we chose $d_{\max} := 30$ because increasing it to 50 did not improve the analysis, and when increasing it to 60, the optimization did not finish within an hour.

▶ **Lemma 9.** $c^{-\Delta_1(D,\mathbf{d})} + c^{-\Delta_2(D,\mathbf{d})} \leq 1$ *holds for any* $(D, \mathbf{d})$ *with* $D \geq 4$.

Proof. Suppose that $d_j \geq 32$ holds for some j. Because $\beta_d = \beta_{30}$ for all $d \geq 30$ and because $d' = D + \sum_i (d_i - 2) \geq D + (d_j - 2) \geq 31$ holds, decreasing d_j by one does not change $\Delta_1(D, \mathbf{d})$ nor $\Delta_2(D, \mathbf{d})$. Therefore, we can focus on the case of $d_i \leq 31$ for all i. We now show that the inequality holds by induction on D. When $D = 4$, we can verify that

$$c^{-\Delta_1(4,\mathbf{d})} + c^{-\Delta_2(4,\mathbf{d})} \leq 1 \ \ (\forall \mathbf{d}) \tag{1}$$

holds by naively enumerating all the possible configurations of $\mathbf{d}$. Assume that, for a fixed D, the inequality holds for any $(D-1, \mathbf{d})$. We show that the inequality also holds for any $(D, \mathbf{d})$.

Table 1 The values of β.

d	β_d	d	β_d	d	β_d
1	0.000000	11	0.384771	21	0.462794
2	0.000000	12	0.396884	22	0.466788
3	0.114038	13	0.408715	23	0.470435
4	0.186479	14	0.418855	24	0.473778
5	0.238143	15	0.427643	25	0.476853
6	0.277239	16	0.435333	26	0.479691
7	0.308030	17	0.442118	27	0.482320
8	0.332974	18	0.448149	28	0.484760
9	0.353536	19	0.453544	29	0.487032
10	0.370540	20	0.458401	≥ 30	0.489153

When $f < D$, we have

$$\Delta_1(D, \mathbf{d}) \geq \Delta_1(D-1, \mathbf{d})$$

and

$$\Delta_2(D, \mathbf{d}) = \Delta_2(D-1, \mathbf{d}) - 2\frac{\alpha}{D} - \beta_{d'} + 2\frac{\alpha}{D-1} + \beta_{d'-1},$$

where $d' = D + \sum_{i=1}^{f}(d_i - 2)$. When $d' > 31$, we have $\Delta_2(D, \mathbf{d}) = \Delta_2(D-1, \mathbf{d}) - 2\frac{\alpha}{D} + 2\frac{\alpha}{D-1} \geq \Delta_2(D-1, \mathbf{d})$. When $d' \leq 31$, we can verify that our parameters satisfy

$$-2\frac{\alpha}{D} - \beta_{d'} + 2\frac{\alpha}{D-1} + \beta_{d'-1} \geq 0 \ \ (\forall (D, d') \text{ with } 5 \leq D \leq d' \leq 31). \tag{2}$$

Therefore, we have $\Delta_2(D, \mathbf{d}) \geq \Delta_2(D-1, \mathbf{d})$. This shows that

$$c^{-\Delta_1(D,\mathbf{d})} + c^{-\Delta_2(D,\mathbf{d})} \leq c^{-\Delta_1(D-1,\mathbf{d})} + c^{-\Delta_2(D-1,\mathbf{d})} \leq 1.$$

When $f = D$, let $\mathbf{d}' := \{d_1, \ldots, d_{f-1}\}$. We have

$$\Delta_1(D, \mathbf{d}) = \Delta_1(D-1, \mathbf{d}') + \beta_{d_f} - \beta_{d_f-1} \geq \Delta_1(D-1, \mathbf{d}')$$

and

$$\begin{aligned}\Delta_2(D, \mathbf{d}) &= \Delta_2(D-1, \mathbf{d}') - 2\frac{\alpha}{D} + 2\frac{\alpha}{D-1} + \beta_{d_f} - \beta_{d'} + \beta_{d'-d_f+2} \\ &\geq \Delta_2(D-1, \mathbf{d}') + \beta_{d_f} - \beta_{d'} + \beta_{d'-d_f+2},\end{aligned}$$

where $d' = D + \sum_{i=1}^{f}(d_i - 2)$. When $d' > 31$, we have $\beta_{d_f} - \beta_{d'} + \beta_{d'-d_f+2} \geq \beta_{d_f} - \beta_{31} + \beta_{31-d_f+2}$, and hence such a case can be reduced to the case when $d' = 31$. Because we can verify that our parameters satisfy

$$\beta_{d_f} - \beta_{d'} + \beta_{d'-d_f+2} \geq 0 \ \ (\forall (d', d_f) \text{ with } 3 \leq d_f < d' \leq 31), \tag{3}$$

we have $\Delta_2(D, \mathbf{d}) \geq \Delta_2(D-1, \mathbf{d}')$. This shows that

$$c^{-\Delta_1(D,\mathbf{d})} + c^{-\Delta_2(D,\mathbf{d})} \leq c^{-\Delta_1(D-1,\mathbf{d}')} + c^{-\Delta_2(D-1,\mathbf{d}')} \leq 1. \quad \blacktriangleleft$$

▶ **Theorem 10.** *Algorithm 1 solves* FVS *in* $O^*(3.460^k)$ *time.*

4 Conclusion

In this paper, we give the theoretical analysis to the empirically fast branching algorithm for FVS. The proved running time is the current fastest among deterministic algorithms. We conclude the paper by giving two open problems.

First, we do not think that our analysis is tight. Can we significantly improve the analysis? Especially, we are interested in whether we can achieve $O^*(4^k)$ time without using the reduction to the linear matroid parity to solve subcubic instances. When allowing randomization, a simple random sampling algorithm runs in $O^*(4^k)$ time [1]. This random sampling algorithm also uses a degree-based argument but does not use the subcubic solver. Without the subcubic solver (i.e., $D \geq 3$ instead of $D \geq 4$), our analysis gives only $O^*(4.59^k)$. If this can be improved to $O^*(4^k)$, using the subcubic solver will further improve the running time.

Second, the proved running time is still far from actual running time against real-world instances. In this paper, we used the standard parameter of the solution size. Can we prove that the algorithm runs in FPT time for some parameter that is smaller in real-world instances. For example, Vertex Cover and Multiway Cut are known to be FPT parameterized by the gap between the solution size and LP lower bounds [8, 21].

References

1 Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized Algorithms for the Loop Cutset Problem. *J. Artif. Intell. Res.*, 12:219–234, 2000. doi:10.1613/jair.638.

2 Hans L. Bodlaender. On Disjoint Cycles. *Int. J. Found. Comput. Sci.*, 5(1):59–68, 1994. doi:10.1142/S0129054194000049.

3 Kevin Burrage, Vladimir Estivill-Castro, Michael R. Fellows, Michael A. Langston, Shev Mac, and Frances A. Rosamond. The Undirected Feedback Vertex Set Problem Has a Poly(k) Kernel. In *IWPEC 2006*, pages 192–202, 2006. doi:10.1007/11847250_18.

4 Yixin Cao. A Naive Algorithm for Feedback Vertex Set. In *SOSA 2018*, pages 1:1–1:9, 2018. doi:10.4230/OASIcs.SOSA.2018.1.

5 Yixin Cao, Jianer Chen, and Yang Liu. On Feedback Vertex Set: New Measure and New Structures. *Algorithmica*, 73(1):63–86, 2015. doi:10.1007/s00453-014-9904-6.

6 Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. Syst. Sci.*, 74(7):1188–1198, 2008. doi:10.1016/j.jcss.2008.05.002.

7 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 *FOCS 2011*, pages 150–159, 2011. doi:10.1109/FOCS.2011.23.

8 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.

9 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 *IPEC 2016*, pages 30:1–30:9, 2016. doi:10.4230/LIPIcs.IPEC.2016.30.

10 Rodney G. Downey and Michael R. Fellows. Fixed Parameter Tractability and Completeness. In *Complexity Theory: Current Research*, pages 191–225, 1992.

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

12 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, 2009. doi:10.1145/1552285.1552286.

13 Serge Gaspers. *Exponential Time Algorithms - Structures, Measures, and Bounds*. VDM, 2010.

14 Kensuke Imanishi and Yoichi Iwata. Feedback Vertex Set solver, 2016. Entry to PACE challenge 2016. URL: `http://github.com/wata-orz/fvs`.

15 Yoichi Iwata. Linear-Time Kernelization for Feedback Vertex Set. In *ICALP 2017*, pages 68:1–68:14, 2017. `doi:10.4230/LIPIcs.ICALP.2017.68`.

16 Yoichi Iwata, Magnus Wahlström, and Yuichi Yoshida. Half-integrality, LP-branching and FPT Algorithms. *SIAM J. Comput.*, 45(4):1377–1411, 2016. `doi:10.1137/140962838`.

17 Yoichi Iwata, Yutaro Yamaguchi, and Yuichi Yoshida. 0/1/All CSPs, Half-Integral A-Path Packing, and Linear-Time FPT Algorithms. In *FOCS 2018*, pages 462–473, 2018. `doi:10.1109/FOCS.2018.00051`.

18 Richard M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations 1972*, pages 85–103, 1972. `doi:10.1007/978-1-4684-2001-2_9`.

19 Krzysztof Kiljan and Marcin Pilipczuk. Experimental Evaluation of Parameterized Algorithms for Feedback Vertex Set. In *SEA 2018*, pages 12:1–12:12, 2018. `doi:10.4230/LIPIcs.SEA.2018.12`.

20 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic Feedback Vertex Set. *Inf. Process. Lett.*, 114(10):556–560, 2014. `doi:10.1016/j.ipl.2014.05.001`.

21 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster Parameterized Algorithms Using Linear Programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, 2014. `doi:10.1145/2566616`.

Subexponential-Time Algorithms for Finding Large Induced Sparse Subgraphs

Jana Novotná
Department of Applied Mathematics, 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
k.okrasa@mini.pw.edu.pl

Michał Pilipczuk
Institute of Informatics, Faculty of Mathematics, Informatics and Mechanics,
University of Warsaw, Poland
michal.pilipczuk@mimuw.edu.pl

Paweł Rzążewski
Faculty of Mathematics and Information Science, Warsaw University of Technology, Poland
p.rzazewski@mini.pw.edu.pl

Erik Jan van Leeuwen
Department of Information and Computing Sciences, Utrecht University, The Netherlands
e.j.vanleeuwen@uu.nl

Bartosz Walczak
Department of Theoretical Computer Science, Faculty of Mathematics and Computer Science,
Jagiellonian University, Kraków, Poland
walczak@tcs.uj.edu.pl

Abstract

Let $\mathcal{C}$ and $\mathcal{D}$ be hereditary graph classes. Consider the following problem: given a graph $G \in \mathcal{D}$, find a largest, in terms of the number of vertices, induced subgraph of G that belongs to $\mathcal{C}$. We prove that it can be solved in $2^{o(n)}$ time, where n is the number of vertices of G, if the following conditions are satisfied:

- the graphs in $\mathcal{C}$ are sparse, i.e., they have linearly many edges in terms of the number of vertices;
- the graphs in $\mathcal{D}$ admit balanced separators of size governed by their density, e.g., $\mathcal{O}(\Delta)$ or $\mathcal{O}(\sqrt{m})$, where Δ and m denote the maximum degree and the number of edges, respectively; and
- the considered problem admits a single-exponential fixed-parameter algorithm when parameterized by the treewidth of the input graph.

This leads, for example, to the following corollaries for specific classes $\mathcal{C}$ and $\mathcal{D}$:

- a largest induced forest in a P_t-free graph can be found in $2^{\tilde{\mathcal{O}}(n^{2/3})}$ time, for every fixed t; and
- a largest induced planar graph in a string graph can be found in $2^{\tilde{\mathcal{O}}(n^{3/4})}$ time.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Problems, reductions and completeness

Keywords and phrases subexponential algorithm, feedback vertex set, P_t-free graphs, string graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.23

Funding *Jana Novotná*: Supported by student grants GAUK 1277018, SVV-2017-260452.
Michał Pilipczuk: This work is a part of project TOTAL that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 677651).
Paweł Rzążewski: Partially supported by Polish National Science Centre grant no. 2018/31/D/ST6/00062.
Bartosz Walczak: Partially supported by Polish National Science Centre grant no. 2015/17/B/ST6/01873.

Acknowledgements The results presented in this paper were obtained during the Parameterized Algorithms Retreat of the algorithms group of the University of Warsaw (PARUW), held in Karpacz in February 2019. This Retreat was financed by the project CUTACOMBS, which has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 714704).

1 Introduction

Many optimization problems in graphs can be expressed as follows: given a graph G, find a largest vertex set A such that $G[A]$, the subgraph of G induced by A, satisfies some property. Examples include Independent Set (the property of being edgeless), Feedback Vertex Set (the property of being acyclic), and Planarization (the property of being planar). Here, Feedback Vertex Set and Planarization are customarily phrased in the complementary form that asks for minimizing the complement of A: given G, find a smallest vertex set X such that $G - X$ has the desired property. While all problems considered in this paper can be viewed in these two ways, for the sake of clarity we focus on the maximization formulation.

Formally, we shall consider the following Max Induced $\mathcal{C}$-Subgraph problem. Fix a graph class $\mathcal{C}$ that is *hereditary*, that is, closed under taking induced subgraphs. Then, given a graph G, the goal is to find a largest vertex subset A such that $G[A] \in \mathcal{C}$. Our focus is on exact algorithms for this problem with running time expressed in terms of n, the number of vertices of G. Clearly, as long as the graphs from $\mathcal{C}$ can be recognized in polynomial time, the problem can be solved in $2^n \cdot n^{\mathcal{O}(1)}$ time by brute-force; we are interested in non-trivial improvements over this approach.

The complexity of Max Induced $\mathcal{C}$-Subgraph was studied as early as in 1980 by Lewis and Yannakakis [19], who proved that when the graph class $\mathcal{C}$ does not contain all graphs, the problem is NP-hard. Recently, Komusiewicz [17] inspected the reduction of Lewis and Yannakakis and concluded that under the Exponential Time Hypothesis (ETH) one can even exclude the existence of *subexponential-time* algorithms for the problem, that is, ones with running time $2^{o(n)}$. While the result of Komusiewicz [17] excludes significant improvements in the running time, there is still room for improvement in the base of the exponent. Indeed, for various classes of graphs $\mathcal{C}$, algorithms with running time $\mathcal{O}((2-\varepsilon)^n)$ for some $\varepsilon > 0$ are known; see e.g. [3, 12, 13, 14, 21] and the references therein.

Another direction, which is of main interest to us, is to impose more conditions on the input graphs G in the hope of obtaining faster algorithms for restricted cases. Formally, we fix another hereditary graph class $\mathcal{D}$ and consider Max Induced $\mathcal{C}$-Subgraph where the input graph G is additionally required to belong to $\mathcal{D}$.

In this line of research, the class $\mathcal{C}$ of edgeless graphs, which corresponds to the classical Max Independent Set (MIS) problem, has been extensively studied. Suppose $\mathcal{D}$ is the class of *H-free graphs*, that is, graphs that exclude some fixed graph H as an induced subgraph. As observed by Alekseev [1], the problem is NP-hard on H-free graphs unless H is a path or a subdivision of the claw ($K_{1,3}$); the reduction of [1] actually excludes the existence of a subexponential-time algorithm under ETH in these cases. On the positive side, the maximal classes for which polynomial-time algorithms are known are the P_6-free graphs [15] and the fork-free graphs [20]. It would be consistent with our knowledge if MIS was polynomial-time solvable on H-free graphs whenever H is a path or a subdivision of the claw.

It turns out that if we only aim at subexponential-time instead of polynomial-time algorithms, many more tractability results can be obtained for MIS, and usually they are also

much simpler conceptually. Bacsó et al. [2] showed that MIS can be solved in $2^{\mathcal{O}(\sqrt{tn \log n})}$ time on P_t-free graphs, for every $t \in \mathbb{N}$. Very recently, Chudnovsky et al. [8] reported a $2^{\mathcal{O}(\sqrt{n \log n})}$-time algorithm on *long-hole-free graphs*, which are graphs that exclude every cycle of length at least 5 as an induced subgraph.

In the light of the results above, it is natural to ask whether structural assumptions on the class $\mathcal{D}$ from which the input is drawn, like e.g. P_t-freeness, can help in the design of subexponential-time algorithms for other maximum induced subgraph problems, beyond $\mathcal{C}$ being the class of edgeless graphs. This is precisely the question we investigate in this work.

Our contribution. We identify three properties that together provide a way to solve the Max Induced $\mathcal{C}$-Subgraph problem on graphs from $\mathcal{D}$ in subexponential time, where $\mathcal{C}$ and $\mathcal{D}$ are hereditary graph classes. They are as follows:

- The class $\mathcal{C}$ should consist of *sparse* graphs. To be specific, let us assume that every n-vertex graph from $\mathcal{C}$ has $\mathcal{O}(n)$ edges.
- The class $\mathcal{D}$ may contain dense graphs, but they should admit balanced separators whose size is somehow governed by the density. To be specific, let us assume that every graph from $\mathcal{D}$ with maximum degree Δ has a balanced separator of size $\mathcal{O}(\Delta)$, or that every graph from $\mathcal{D}$ with m edges has a balanced separator of size $\mathcal{O}(\sqrt{m})$.
- The Max Induced $\mathcal{C}$-Subgraph problem on graphs from $\mathcal{D}$ can be solved in $2^{\tilde{\mathcal{O}}(w)} \cdot n^{\mathcal{O}(1)}$ time, where w is the treewidth of the input graph. Here, notation $\tilde{\mathcal{O}}(\cdot)$ hides polylogarithmic factors.

We show that if these conditions are simultaneously satisfied, then the Max Induced $\mathcal{C}$-Subgraph problem on graphs from $\mathcal{D}$ can be solved in $2^{\tilde{\mathcal{O}}(n^{2/3})}$ time in the presence of balanced separators of size $\mathcal{O}(\Delta)$ and in $2^{\tilde{\mathcal{O}}(n^{3/4})}$ time for balanced separators of size $\mathcal{O}(\sqrt{m})$. The precise statement and proof of this result can be found in Section 2.

The conditions on $\mathcal{C}$ look natural and are satisfied by various specific classes of interest, like forests (corresponding to Feedback Vertex Set) and planar graphs (corresponding to Planarization). On the other hand, the condition on $\mathcal{D}$ looks more puzzling. However, there are certain non-sparse classes of graphs where the existence of such balanced separators has been established. For instance, balanced separators of size $\mathcal{O}(\Delta)$ are known to exist in P_t-free graphs for any fixed $t \in \mathbb{N}$ [2], and in long-hole-free graphs [8]. The existence of balanced separators of size $\mathcal{O}(\sqrt{m})$ is known for *string graphs*, which are intersection graphs of arc-connected subsets of the plane, and more generally for intersection graphs of connected subgraphs in any proper minor-closed class [18]. All these observations yield a number of concrete corollaries to our main result, which are gathered in Section 3. In Section 4, we discuss some lower bounds: we show that if $\mathcal{C}$ is the class of forests (corresponding to the Feedback Vertex Set problem) and $\mathcal{D}$ is characterized by a single excluded induced subgraph, then under the Exponential Time Hypothesis one cannot hope for subexponential-time algorithms in greater generality than provided by our main result.

2 Main result

We use standard graph notation. We assume the reader's familiarity with treewidth. We recall some notation for tree decompositions in Section 5, where it is actually needed.

For a graph G, a set $S \subseteq V(G)$ is a *balanced separator* if every connected component of $G - S$ has at most $\frac{2}{3}|V(G)|$ vertices. It is known that small balanced separators can be used to construct tree decompositions of small width, as made explicit in the following lemma.

▶ **Lemma 1** ([11]). *If every subgraph of a graph G has a balanced separator of size at most k, then the treewidth of G is $\mathcal{O}(k)$.*

Now, we are ready to state and prove our main result.

▶ **Theorem 2.** *Let $\mathcal{C}$ and $\mathcal{D}$ be classes of graphs that satisfy the following conditions:*

(P1) *Every n-vertex graph from $\mathcal{C}$ has $\mathcal{O}(n)$ edges.*

(P2) *The class $\mathcal{D}$ is closed under taking induced subgraphs.*

(P3) *Given a graph $G \in \mathcal{D}$ with n vertices and treewidth w, one can find a largest set $A \subseteq V(G)$ such that $G[A] \in \mathcal{C}$ in $2^{\tilde{\mathcal{O}}(w)} \cdot n^{\mathcal{O}(1)}$ time.*

Furthermore, let the class $\mathcal{D}$ satisfy one of the following conditions:

(P4a) *Every graph in $\mathcal{D}$ with maximum degree Δ has a balanced separator of size $\mathcal{O}(\Delta)$, or*

(P4b) *Every graph in $\mathcal{D}$ with n vertices and maximum degree Δ has a balanced separator of size $\mathcal{O}(\sqrt{n\Delta})$.*

Then, given an n-vertex graph $G \in \mathcal{D}$, one can find a largest set $A \subseteq V(G)$ such that $G[A] \in \mathcal{C}$ in time

(1) *$2^{\tilde{\mathcal{O}}(n^{2/3})}$, if $\mathcal{D}$ satisfies* (P4a)*, or*

(2) *$2^{\tilde{\mathcal{O}}(n^{3/4})}$, if $\mathcal{D}$ satisfies* (P4b)*.*

Proof. Let a constant τ be defined as follows, depending on which of the two conditions is satisfied by $\mathcal{D}$:

$$\tau = \begin{cases} 1/3 & \text{if } \mathcal{D} \text{ satisfies (P4a)}, \\ 1/4 & \text{if } \mathcal{D} \text{ satisfies (P4b)}. \end{cases}$$

We devise a branching algorithm that finds a largest set $A \subseteq V(G)$ such that $G[A] \in \mathcal{C}$ in $2^{\tilde{\mathcal{O}}(n^{1-\tau})}$ time. This matches the complexity bounds from the statement of the theorem.

Let $G \in \mathcal{D}$ be the input graph and n be the number of its vertices. Consider a fixed solution A, that is, a largest set $A \subseteq V(G)$ such that $G[A] \in \mathcal{C}$. Let $A' \subseteq A$ be the set of vertices of degree greater than n^τ in $G[A]$. By property (P1), we have $|A'| = \mathcal{O}(n/n^\tau) = \mathcal{O}(n^{1-\tau})$.

The algorithm guesses the set A' exhaustively, by trying all subsets of $V(G)$ of the appropriate sizes $\mathcal{O}(n^{1-\tau})$, which results in $n^{\mathcal{O}(n^{1-\tau})} = 2^{\tilde{\mathcal{O}}(n^{1-\tau})}$ branches. Fix one such branch and assume, for the purpose of further description of the algorithm, that it corresponds to the true set A' (i.e., the one obtained from the fixed solution A). Let $G' = G - A'$.

Suppose that G' contains a vertex v of degree at least $n^{2\tau}$. If $v \in A$, then v has degree at most n^τ in $G[A]$ (since $v \notin A'$). The algorithm further guesses that $v \notin A$ and discards v (one branch), or it guesses that $v \in A$ and discards all but at most n^τ neighbors of v in G' (at most n^{n^τ} branches). In the latter case, we do *not* fix the assumption that v or any particular neighbor of v belongs to A, so that the vertices that have survived this step can still be discarded in subsequent branching steps.

The step described above is repeated exhaustively. The overall number of branches generated in this way can be bounded as follows, where $k = |V(G')|$:

$$\begin{aligned} F(k) &\leqslant F(k-1) + n^{n^\tau} \cdot F(k - (n^{2\tau} - n^\tau)) \\ &\leqslant F(k-2) + n^{n^\tau} \cdot F(k - (n^{2\tau} - n^\tau)) + n^{n^\tau} \cdot F(k - (n^{2\tau} - n^\tau)) \\ &\leqslant \ldots \leqslant F(k - (n^{2\tau} - n^\tau)) + (n^{2\tau} - n^\tau) \cdot n^{n^\tau} \cdot F(k - (n^{2\tau} - n^\tau)) \\ &= (n^{2\tau} - n^\tau + 1) \cdot n^{n^\tau} \cdot F(k - (n^{2\tau} - n^\tau)) \\ &\leqslant \left((n^{2\tau} - n^\tau + 1) \cdot n^{n^\tau}\right)^{k/(n^{2\tau} - n^\tau)} \\ &\leqslant \left((n^{2\tau} - n^\tau + 1) \cdot n^{n^\tau}\right)^{n/(n^{2\tau} - n^\tau)} = n^{\mathcal{O}(n^{1+\tau-2\tau})} = 2^{\tilde{\mathcal{O}}(n^{1-\tau})}. \end{aligned}$$

Once the branching step can no longer be applied, we obtain an induced subgraph G'' of G' of maximum degree less than $n^{2\tau}$. In the branch where all the choices have been made correctly (i.e., according to the fixed solution A), G'' still contains all vertices from $A \setminus A'$.

By property (P2), we have $G'' \in \mathcal{D}$. Thus G'' satisfies either (P4a) or (P4b), which means that G'' has a balanced separator of size $\mathcal{O}(n^{2/3})$ in the former case or $\mathcal{O}(\sqrt{n \cdot n^{1/2}}) = \mathcal{O}(n^{3/4})$ in the latter case. In both cases, the size of the separator is $\mathcal{O}(n^{1-\tau})$. Moreover, by the same argument, balanced separators of that size also exist in every subgraph of G''. Therefore, by Lemma 1, we conclude that G'' has treewidth $\mathcal{O}(n^{1-\tau})$. Since $|A'| \leqslant \mathcal{O}(n^{1-\tau})$, it follows that the graph $G[V(G'') \cup A']$ also has treewidth $\mathcal{O}(n^{1-\tau})$.

We know that $G[V(G'') \cup A'] \in \mathcal{D}$ and, in the branch where all choices have been made correctly, this graph contains the entire maximum-size solution A. Now, we apply the procedure assumed in (P3) to the graph $G[V(G'') \cup A']$ and observe that in the correct branch it finds some maximum-size solution (possibly different from A). Let us point out that in this step it is not sufficient to consider only the graph G'', as the vertices from A' introduce some additional constraints on the solution we are looking for.

For the time complexity, the algorithm considers $2^{\tilde{\mathcal{O}}(n^{1-\tau})}$ branches and in each of them it executes the procedure assumed in (P3) in $2^{\tilde{\mathcal{O}}(n^{1-\tau})}$ time, which gives the total running time of $2^{\tilde{\mathcal{O}}(n^{1-\tau})}$. ◀

▶ Remark 3. The condition (P1) in the statement of Theorem 2 can be relaxed to "every n-vertex graph from $\mathcal{C}$ has $\mathcal{O}(n^{2-\varepsilon})$ edges, for some constant $\varepsilon > 0$". Then, we can follow the same approach with the following modification: we choose $\tau = 1 - \frac{2}{3}\varepsilon$ in case of (P4a) and $\tau = 1 - \frac{3}{4}\varepsilon$ in case of (P4b), and replace the threshold for branching on high-degree vertices from $n^{2\tau}$ to $n^{2\tau+\varepsilon-1}$. This way, we obtain algorithms with running time $2^{\tilde{\mathcal{O}}(n^{1-\varepsilon/3})}$ for property (P4a) and $2^{\tilde{\mathcal{O}}(n^{1-\varepsilon/4})}$ for property (P4b). This running time is subexponential for every $\varepsilon > 0$.

One can also imagine unifying properties (P4a) and (P4b) into the existence of a balanced separator of size $\mathcal{O}(n^{\alpha}\Delta^{\beta})$, for some constants α, β. However, then, one needs to be careful when choosing τ so that it belongs to the interval $[0, 1]$. As we did not find concrete examples of interesting graph classes $\mathcal{D}$ for which this approach would yield non-trivial results and which would not satisfy either (P4a) or (P4b), we refrain from discussing further details here.

3 Corollaries

In this section, we discuss possible classes $\mathcal{C}$ and $\mathcal{D}$ which satisfy the conditions of Theorem 2. For some choices of $\mathcal{C}$, we obtain well-studied computational problems:

1. for matchings, we obtain Max Induced Matching,
2. for forests, we obtain Max Induced Forest, also known as Feedback Vertex Set,
3. for graphs of maximum degree d, where d is fixed, we obtain Max Induced Degree-d Subgraph,
4. for planar graphs, we obtain Max Induced Planar Subgraph, also known as Planarization,
5. for graphs embeddable in Σ, where the surface Σ is fixed, we obtain Max Induced Σ-Embeddable Subgraph,
6. for graphs of degeneracy at most d, where d is fixed, we obtain Max Induced d-Degenerate Subgraph.

It is clear that all these classes satisfy property 1 of Theorem 2.

Given a graph of treewidth w, its tree decomposition of width at most $4w + 3$ can be computed in $2^{\mathcal{O}(w)} \cdot n^2$ time (see e.g. [9, Section 7.6]). Therefore, for the purpose of verifying property 3, we can assume that a tree decomposition of width $\mathcal{O}(w)$ is additionally provided

on input. While $2^{\tilde{\mathcal{O}}(w)} \cdot n^{\mathcal{O}(1)}$-time algorithms are quite straightforward and well known for the first two problems on the list, this is not necessarily the case for the others. For Max Induced Degree-d Subgraph, an algorithm with running time $2^{\mathcal{O}(w)} \cdot n^{\mathcal{O}(1)}$ can be easily derived from the meta-theorem of Pilipczuk [22]. Algorithms for Max Induced Planar Subgraph and, more generally, Max Induced Σ-Embeddable Subgraph, were provided by Kociumaka and Pilipczuk [16]. Finally, we give a suitable algorithm for Max Induced d-Degenerate Subgraph in Lemma 8 in Section 5.

It may be tempting to consider, as $\mathcal{C}$, the graphs with no even cycle C_{2k} (not necessarily induced), for some fixed integer $k \geqslant 2$. This is because such graphs have $\mathcal{O}(n^{2-\Omega(1/k)})$ edges [5], and thus they satisfy the generalization of property 1 mentioned in Remark 3 for $\varepsilon = \Omega(1/k)$. However, for these classes, property 3 turns out to be problematic: for any fixed $\ell \geqslant 5$, there is no algorithm for a minimum set of vertices hitting all (non-induced) copies of C_ℓ in a graph with treewidth w with running time $2^{o(w^2)} \cdot n^{\mathcal{O}(1)}$ unless the ETH fails [22] (this bound appears to be essentially tight, as the problem can be solved in $2^{\tilde{\mathcal{O}}(w^2)} \cdot n^{\mathcal{O}(1)}$ time [10]). It is unclear whether the additional assumption that the input graph belongs to some class $\mathcal{D}$, considered here, can help.

Now, let us consider classes $\mathcal{D}$. Examples of classes satisfying property a in Theorem 2 come from forbidding some induced subgraphs. Bacsó et al. [2] proved that P_t-free graphs with maximum degree Δ have treewidth $\mathcal{O}(\Delta \cdot t)$. Very recently, Chudnovsky et al. [8] observed that *long-hole-free graphs*, that is, graphs with no induced cycles of length at least 5, also have balanced separators of size $\mathcal{O}(\Delta)$.

An example of a class satisfying property b is the class of string graphs – intersection graphs of arc-connected subsets of the plane. Lee [18] showed that they admit balanced separators of size $\mathcal{O}(\sqrt{m})$, where m is the number of edges. In fact, he proved a more general result that if $\mathcal{M}$ is a class of graphs excluding a fixed graph as a minor, then intersection graphs of connected subgraphs of graphs from $\mathcal{M}$ admit balanced separators of size $\mathcal{O}(\sqrt{m})$. String graphs are precisely the intersection graphs of connected subgraphs of planar graphs.

Summing up, we obtain the following.

▶ **Corollary 4.** *Each of the following problems can be solved in $2^{\tilde{\mathcal{O}}(n^{2/3})}$ time on P_t-free graphs (for every fixed t) and in long-hole-free graphs, and in $2^{\tilde{\mathcal{O}}(n^{3/4})}$ time on string graphs:*

1. Max Induced Matching,
2. Max Induced Forest,
3. Max Induced Degree-d Subgraph, *for every fixed* $d \in \mathbb{N}$,
4. Max Induced Planar Subgraph,
5. Max Induced Σ-Embeddable Subgraph, *for every fixed surface* Σ,
6. Max Induced d-Degenerate Subgraph, *for every fixed* $d \in \mathbb{N}$.

We note that subexponential-time algorithms for Max Induced Matching and Max Induced Forest on string graphs were already known [6], even with a better running time than provided above. As we have argued, in Corollary 4, we can replace string graphs with intersection graphs of connected subgraphs of graphs from $\mathcal{M}$, where $\mathcal{M}$ is any class of graphs excluding a fixed graph as a minor; this is because the result of Lee [18] holds in that generality.

4 Max Induced Forest in H-free graphs

Our original motivation was the Max Induced Forest problem. In the previous section, we discussed a subexponential-time algorithm solving it on P_t-free graphs. We now show that as long as the considered class of inputs $\mathcal{D}$ is characterized by a single excluded induced

subgraph, that is, we investigate Max Induced Forest on H-free graphs for a fixed graph H, we cannot hope for more positive results. Namely, it turns out that if H is not a linear forest (i.e., a collection of vertex-disjoint paths), the problem is unlikely to admit a polynomial-time or even a subexponential-time algorithm on H-free graphs. Specifically, we obtain the following dichotomy.

▶ **Theorem 5.** *Let H be a fixed graph.*

1. *If H is a linear forest, then the* Max Induced Forest *problem can be solved in $2^{\tilde{\mathcal{O}}(n^{2/3})}$ time on H-free graphs with n vertices.*
2. *Otherwise, on H-free graphs, the* Max Induced Forest *problem is NP-complete and cannot be solved in $2^{o(n)}$ time unless the ETH fails.*

Theorem 5 1 follows from Corollary 4, because every linear forest is an induced subgraph of some path. Statement 2 follows from a combination of arguments already existing in the literature. However, since the proof is simple, we include it for the sake of completeness.

We prove Theorem 5 2 in two steps. First, we consider graphs H that contain a cycle or two branch vertices, that is, vertices of degree at least 3. In this case, we can apply the standard argument of subdividing every edge a suitable number of times, cf. [7, Theorem 3].

▶ **Lemma 6.** *Let H be a fixed graph that either contains a cycle or has a connected component with at least two branch vertices. Then* Max Induced Forest *is NP-complete on H-free graphs. Moreover, there is no algorithm solving* Max Induced Forest *in $2^{o(n)}$ time for n-vertex H-free graphs unless the ETH fails.*

Proof. We reduce from Max Induced Forest in graphs with maximum degree 6; it is known that this problem is NP-complete and has no subexponential-time algorithm assuming ETH [9]. Let G be a graph with n vertices and maximum degree 6. Let G^* be the graph obtained from G by subdividing every edge $|V(H)|+1$ times. It is straightforward to observe that G has an induced forest on $n-k$ vertices if and only if G^* has an induced forest on $|G^*|-k$ vertices. Moreover, the number of vertices in G^* is linear in n.

Finally, we show that G^* is H-free. First, observe that if H contains a cycle, then H cannot be a subgraph of G^*, as the girth of G^* is greater than $|V(H)|+1$. On the other hand, the distance between any two branch vertices in G^* is at least $|V(H)|+1$, so G^* does not contain H as a subgraph in case H has two branch vertices in the same connected component. ◀

By Lemma 6, the only graphs H for which we might hope for a polynomial-time or even a subexponential-time algorithm for Max Induced Forest on H-free graphs are collections of disjoint subdivided stars. To resolve this case, we will show that the problem remains hard for line graphs. Recall that the line graph $L(G)$ of a graph G is the graph whose vertices are the edges of G and where the adjacency relation corresponds to the relation of having a common endpoint in G.

Actually, Chiarelli et al. [7] reported that the hardness of Max Induced Forest on line graphs was observed by Speckenmeyer in his PhD thesis [23]. However, we were unable to find this result there. Therefore, we provide the easy proof, which boils down to essentially the same argument as in [7, Theorem 5].

▶ **Lemma 7.** Max Induced Forest *is NP-complete on line graphs. Moreover, there is no algorithm solving* Max Induced Forest *in $2^{o(n)}$ time for n-vertex line graphs unless the ETH fails.*

Proof. We reduce from the Hamiltonian Path problem, which is NP-complete and has no subexponential-time algorithm, even if the input graph has linearly many edges [9]. Let G be a graph, which is the input instance of Hamiltonian Path.

First, note that any induced forest in $L(G)$ corresponds to a collection of vertex-disjoint paths in G. More formally, consider a set $E' \subseteq E(G)$, such that $L(G)[E']$ is a forest. We claim that the subgraph $G' = (V(G), E')$ of G is a collection of vertex-disjoint paths. Suppose not. This means that G' contains a vertex v of degree at least 3 or a cycle C. In the former case, the edges incident to v in G' form a clique in $L(G)[E']$. In the latter case, the edges of the cycle C form a cycle in $L(G)[E']$. In either case, we get a contradiction to the assumption that $L(G)[E']$ is a forest.

We claim that G has a Hamiltonian path if and only if $L(G)$ has an induced forest on $n-1$ vertices. Indeed, the $n-1$ edges of a Hamiltonian path in G induce a path (in particular, a forest) in $L(G)$. For the converse, suppose that $L(G)$ has an induced forest on at least $n-1$ vertices. By the observation above, this induced forest corresponds to a collection of vertex-disjoint paths in G with at least $n-1$ edges in total. This is only possible if this collection consists of a single path of length $n-1$, that is, a Hamiltonian path in G.

Finally, observe that the number of vertices of $L(G)$ is equal to the number of edges of G, which is linear in the number of vertices of G. ◀

Recall that line graphs are claw-free, that is, they contain no induced copy of $K_{1,3}$. Thus Lemma 7 implies that if H contains any star with at least 3 leaves, then Max Induced Forest remains NP-complete and has no subexponential-time algorithm on H-free graphs unless ETH fails. Theorem 5 2 follows from combining Lemma 6 and Lemma 7.

5 Largest induced degenerate subgraph in low-treewidth graphs

This section is devoted to the proof of the following result, which we used in Section 3.

▶ **Lemma 8.** *For every fixed $d \in \mathbb{N}$, there is an algorithm for* Max Induced d-Degenerate Subgraph *with running time $2^{\mathcal{O}(w \log w)} \cdot n$, where w is the treewidth of the input graph and n is the number of its vertices.*

Preliminaries on tree decompositions. First, we introduce some notation and terminology. A *tree decomposition* of a graph G is a tree T together with a mapping $\beta(\cdot)$ that assigns a *bag* $\beta(x)$ to each node x of T in such a way that the following conditions hold:

(T1) for each $u \in V(G)$, the set of nodes x with $u \in \beta(x)$ induces a connected non-empty subtree of T; and

(T2) for each $uv \in E(G)$, there exists a node x such that $\{u, v\} \subseteq \beta(x)$.

The *width* of a tree decomposition (T, β) is $\max_{x \in V(T)} |\beta(x)| - 1$, and the *treewidth* of a graph G is the minimum width of a tree decomposition of G.

Henceforth, all tree decompositions will be *rooted*: the underlying tree T has a prescribed root vertex r. This gives rise a natural ancestor-descendant relation: we write $x \preceq y$ if x is an ancestor of y (where possibly $x = y$). Then, for a node x of T, we define the *component* at x as

$$\alpha(x) = \left(\bigcup_{y \succeq x} \beta(y) \right) \setminus \beta(x).$$

It easily follows from (T1) and (T2) that then $N(\alpha(x)) \subseteq \beta(x)$ for every node x.

A *nice tree decomposition* is a normalized form of a rooted tree decomposition in which every node is of one of the following four kinds.

- **Leaf node**: a node x with no children and with $\beta(x) = \emptyset$.
- **Introduce node**: a node x with one child y such that $\beta(x) = \beta(y) \cup \{u\}$ for some vertex $u \notin \beta(y)$.
- **Forget node**: a node x with one child y such that $\beta(x) = \beta(y) \setminus \{u\}$ for some vertex $u \in \beta(y)$.
- **Join node**: a node x with two children y and z such that $\beta(x) = \beta(y) = \beta(z)$.

Moreover, we require that the root r of the nice tree decomposition satisfies $\beta(r) = \emptyset$.

It is known that any given tree decomposition (T, β) of width k of an n-vertex graph G can be transformed in $k^{\mathcal{O}(1)} \cdot \max(n, |V(T)|)$ time into a nice tree decomposition of G of width at most as large, see [9, Lemma 7.4]. Moreover, given an n-vertex graph G of treewidth w, a tree decomposition of G of width at most $5w+4$ can be computed in $2^{\mathcal{O}(w)} \cdot n$ time [4], and this tree decomposition has at most n nodes. By combining these two results, for the proof of Lemma 8, we can assume that the input graph G is supplied with a nice tree decomposition (T, β) of width $k \leqslant 5w + 4$, where $w = \mathrm{tw}(G)$. From now on, our goal is to design a suitable dynamic programming algorithm working on this decomposition with running time $2^{\mathcal{O}(k \log k)} \cdot n = 2^{\mathcal{O}(w \log w)} \cdot n$.

Dynamic programming states. The main idea behind our dynamic programming algorithm is to view the notion of degeneracy via vertex orderings, as expressed in the following fact.

▶ **Lemma 9** (Folklore). *A graph H is d-degenerate if and only if there is a linear ordering σ of vertices of H such that every vertex of H has at most d neighbors that are smaller in σ.*

Hence, the problem considered in Lemma 8 can be restated as follows: find a largest set $A \subseteq V(G)$ that admits a linear ordering σ in which every vertex of A has at most d neighbors in $G[A]$ that are smaller in σ. Intuitively, our dynamic programming will therefore keep track of the intersection of the bag with A, the restriction of σ to this intersection; and how many smaller neighbors of each vertex from this intersection have been already forgotten.

We now proceed with formal details. For a node x of T, a set $X \subseteq \beta(x)$, a linear ordering σ of X, and a function $f\colon X \to \{0, \ldots, d\}$, we define $\Phi_x[X, \sigma, f] \in \mathbb{N}$ as follows. The value $\Phi_x[X, \sigma, f]$ is the maximum size of a set $Y \subseteq \alpha(x)$ such that $X \cup Y$ admits a linear ordering τ with the following properties: τ restricted to X is equal to σ and for every $a \in X$, there are at most $f(a)$ vertices $b \in Y$ that are adjacent to a and smaller than a in τ. Note that other neighbors of a that belong to X are *not* taken into consideration when verifying the quota imposed by $f(a)$. Note also that such a set Y always exists, as $Y = \emptyset$ satisfies the criteria.

For a fixed node x, the total number of triples (X, σ, f) as above is at most

$$2^{k+1} \cdot (k+1)! \cdot (d+1)^{k+1} \leqslant 2^{\mathcal{O}(k \log k)}.$$

Hence, we now show how to compute the values $\Phi_x[X, \sigma, f]$ in a bottom-up manner, so that the values for a node x are computed based on the values for the children of x in $2^{\mathcal{O}(k \log k)}$ time. The answer to the problem corresponds to the value $\Phi_r[\emptyset, \emptyset, \emptyset]$, where r is the root of T. While $\Phi_r[\emptyset, \emptyset, \emptyset]$ is just the size of a largest feasible solution, an actual solution can be recovered from the dynamic programming tables using standard methods within the same complexity: for every computed value $\Phi_x[X, \sigma, f]$, we store the way this value was obtained, and then we trace back the solution from $\Phi_r[\emptyset, \emptyset, \emptyset]$ in a top-down manner.

Transitions. It remains to provide recursive formulas for the values of $\Phi_x[\cdot, \cdot, \cdot]$. We only present the formulas, while the verification of their correctness, which follows easily from the definition of $\Phi_x[\cdot, \cdot, \cdot]$, is left to the reader. As usual, we distinguish cases depending on the type of x.

- **Leaf node** x. Then we have only one value:

  $$\Phi_x[\emptyset, \emptyset, \emptyset] = 0.$$

- **Introduce node** x with child y such that $\beta(x) = \beta(y) \cup \{u\}$. Then

  $$\Phi_x[X, \sigma, f] = \begin{cases} \Phi_y[X, \sigma, f] & \text{if } u \notin X; \\ \Phi_y[X \setminus \{u\}, \sigma|_{X\setminus\{u\}}, f|_{X\setminus\{u\}}] & \text{if } u \in X. \end{cases}$$

- **Forget node** x with child y such that $\beta(x) = \beta(y) \setminus \{u\}$. Then we have

  $$\Phi_x[X, \sigma, f] = \max\left(\Phi_y[X, \sigma, f],\ 1 + \max_{(\sigma', f') \in S(X,\sigma,f)} \Phi_y[X \cup \{u\}, \sigma', f']\right),$$

  where $S(X, \sigma, f)$ is the set comprising the pairs (σ', f') satisfying the following:
  - σ' is a vertex ordering of $X \cup \{u\}$ whose restriction to X is equal to σ; and
  - $f' \colon X \cup \{u\} \to \{0, \ldots, d\}$ is such that for all $a \in X$ that are adjacent to u and larger than u in σ', we have $f'(a) \leqslant f(a) - 1$, and for all other $a \in X$, we have $f'(a) \leqslant f(a)$. Moreover, we require that $f'(u) \leqslant d - \ell$, where ℓ is the number of vertices $a \in X$ that are adjacent to u and smaller than u in σ'.
- **Join node** x with children y and z. Then

  $$\Phi_x[X, \sigma, f] = \max_{f_y + f_z \leqslant f} \Phi_y[X, \sigma, f_y] + \Phi_z[X, \sigma, f_z],$$

  where $f_y + f_z \leqslant f$ means that $f_y(a) + f_z(a) \leqslant f(a)$ for each $a \in X$.

It is straightforward to see that using the formulas above, each value $\Phi_x[X, \sigma, f]$ can be computed in $2^{\mathcal{O}(k \log k)}$ time based on the values computed for the children of x. This completes the proof of Lemma 8.

References

1 Vladimir E. Alekseev. The effect of local constraints on the complexity of determination of the graph independence number. *Combinatorial-algebraic methods in applied mathematics*, pages 3–13, 1982. (in Russian).

2 Gábor Bacsó, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Zsolt Tuza, and Erik Jan van Leeuwen. Subexponential-Time Algorithms for Maximum Independent Set in P_t-Free and Broom-Free Graphs. *Algorithmica*, 81(2):421–438, 2019.

3 Ivan Bliznets, Fedor V. Fomin, Michał Pilipczuk, and Yngve Villanger. Largest Chordal and Interval Subgraphs Faster than 2^n. *Algorithmica*, 76(2):569–594, 2016. doi:10.1007/s00453-015-0054-2.

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

5 John Adrian Bondy and Miklós Simonovits. Cycles of even length in graphs. *J. Combin. Theory Ser. B*, 16(2):97–105, 1974.

6 Édouard Bonnet and Paweł Rzążewski. Optimality Program in Segment and String Graphs. *Algorithmica*, 81(7):3047–3073, 2019. doi:10.1007/s00453-019-00568-7.

7 Nina Chiarelli, Tatiana Romina Hartinger, Matthew Johnson, Martin Milanič, and Daniël Paulusma. Minimum connected transversals in graphs: New hardness results and tractable cases using the price of connectivity. *Theor. Comput. Sci.*, 705:75–83, 2018. doi:10.1016/j.tcs.2017.09.033.

8 Maria Chudnovsky, Marcin Pilipczuk, Michał Pilipczuk, and Stéphan Thomassé. On the Maximum Weight Independent Set Problem in graphs without induced cycles of length at least five. *CoRR*, abs/1903.04761, 2019. `arXiv:1903.04761`.

9 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`.

10 Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. Hitting forbidden subgraphs in graphs of bounded treewidth. *Inf. Comput.*, 256:62–82, 2017. `doi:10.1016/j.ic.2017.04.009`.

11 Zdeněk Dvořák and Sergey Norin. Treewidth of graphs with balanced separations. *J. Combin. Theory Ser. B*, 137:137–144, 2019. `doi:10.1016/j.jctb.2018.12.007`.

12 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. In *STOC 2016*, pages 764–775. ACM, 2016.

13 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Exact Algorithm for the Maximum Induced Planar Subgraph Problem. In *ESA 2011*, volume 6942 of *LNCS*, pages 287–298. Springer, 2011.

14 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large Induced Subgraphs via Triangulations and CMSO. *SIAM J. Comput.*, 44(1):54–87, 2015.

15 Andrzej Grzesik, Tereza Klimošová, Marcin Pilipczuk, and Michał Pilipczuk. Polynomial-time algorithm for Maximum Weight Independent Set on P_6-free graphs. In *SODA 2019*, pages 1257–1271. SIAM, 2019.

16 Tomasz Kociumaka and Marcin Pilipczuk. Deleting vertices to graphs of bounded genus. *CoRR*, abs/1706.04065, 2017. `arXiv:1706.04065`.

17 Christian Komusiewicz. Tight Running Time Lower Bounds for Vertex Deletion Problems. *ACM Trans. on Comput. Theory (TOCT)*, 10(2):6:1–6:18, 2018.

18 James R. Lee. Separators in Region Intersection Graphs. In *ITCS 2017*, volume 67 of *LIPIcs*, pages 1:1–1:8. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2017.

19 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`.

20 Vadim V. Lozin and Martin Milanič. A polynomial algorithm to find an independent set of maximum weight in a fork-free graph. *J. Discrete Algorithms*, 6(4):595–604, 2008.

21 Marcin Pilipczuk and Michał Pilipczuk. Finding a Maximum Induced Degenerate Subgraph Faster Than 2^n. In *IPEC 2012*, volume 7535 of *LNCS*, pages 3–12. Springer, 2012.

22 Michał Pilipczuk. Problems Parameterized by Treewidth Tractable in Single Exponential Time: A Logical Approach. In *MFCS 2011*, volume 6907, pages 520–531. Springer, 2011.

23 Ewald Speckenmeyer. *Untersuchungen zum Feedback Vertex Set Problem in ungerichteten Graphen*. PhD thesis, Universität Paderborn, 1983. In German.

Beating Treewidth for Average-Case Subgraph Isomorphism

Gregory Rosenthal
University of Toronto, Canada
http://www.cs.toronto.edu/~rosenthal/
rosenthal@cs.toronto.edu

Abstract

For any fixed graph G, the subgraph isomorphism problem asks whether an n-vertex input graph has a subgraph isomorphic to G. A well-known algorithm of Alon, Yuster and Zwick (1995) efficiently reduces this to the "colored" version of the problem, denoted G-SUB, and then solves G-SUB in time $O(n^{tw(G)+1})$ where $tw(G)$ is the treewidth of G. Marx (2010) conjectured that G-SUB requires time $\Omega(n^{\text{const}\cdot tw(G)})$ and, assuming the Exponential Time Hypothesis, proved a lower bound of $\Omega(n^{\text{const}\cdot emb(G)})$ for a certain graph parameter $emb(G) = \Omega(tw(G)/\log tw(G))$. With respect to the size of AC^0 circuits solving G-SUB, Li, Razborov and Rossman (2017) proved an unconditional average-case lower bound of $\Omega(n^{\kappa(G)})$ for a different graph parameter $\kappa(G) = \Omega(tw(G)/\log tw(G))$.

Our contributions are as follows. First, we show that $emb(G)$ is at most $O(\kappa(G))$ for all graphs G. Next, we show that $\kappa(G)$ can be asymptotically less than $tw(G)$; for example, if G is a hypercube then $\kappa(G)$ is $\Theta\left(tw(G)/\sqrt{\log tw(G)}\right)$. Finally, we construct AC^0 circuits of size $O(n^{\kappa(G)+\text{const}})$ that solve G-SUB in the average case, on a variety of product distributions. This improves an $O(n^{2\,\kappa(G)+\text{const}})$ upper bound of Li et al., and shows that the average-case complexity of G-SUB is $n^{o(tw(G))}$ for certain families of graphs G such as hypercubes.

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph algorithms

Keywords and phrases subgraph isomorphism, average-case complexity, AC^0, circuit complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.24

Related Version The full paper is available at https://arxiv.org/abs/1902.06380.

Funding *Gregory Rosenthal*: NSERC (PGS D)

Acknowledgements Thanks to Benjamin Rossman for introducing me to this topic, and for having many helpful discussions about the research and about drafts of this paper. Thanks to Henry Yuen for providing feedback on a draft of this paper as well. Part of this work was done while the author was visiting the Simons Institute for the Theory of Computing.

1 Introduction

The subgraph isomorphism problem asks, given graphs X and G, whether X has a subgraph isomorphic to G. In the "colored" or "partitioned" version of the problem, each vertex of the larger graph X comes with a "color" from the vertex set of G, and we ask whether X has a subgraph that is isomorphic to G with respect to this coloring. We denote the uncolored and colored subgraph isomorphism problems by $G\text{-SUB}_{\text{uncol}}(X)$ and $G\text{-SUB}(X)$ respectively.

Subgraph isomorphism is NP-complete (e.g. if G is a clique or Hamiltonian cycle), so research has focused on algorithms for a variety of special cases in the context of parameterized complexity, surveyed in [12]. If G is a fixed graph on k vertices then $G\text{-SUB}_{\text{uncol}}$ is solvable in time $O(n^k)$ by brute force, where (here and throughout this section) n is the order of the input graph. The color-coding algorithm of Alon, Yuster and Zwick [2] improves on this by efficiently reducing $G\text{-SUB}_{\text{uncol}}$ to G-SUB and solving the latter in time $O(n^{tw(G)+1})$, where $tw(G)$ is the treewidth of the fixed graph G.

The exponent $tw(G)+1$ can sometimes be improved using fast matrix multiplication [14, 5], but no significantly faster algorithm is known for either the colored or uncolored subgraph isomorphism problem. The following was conjectured in [11]:

▶ **Conjecture 1.1.** *There is no class $\mathcal{G}$ of graphs with unbounded treewidth, no algorithm $\mathbb{A}$ that on inputs G and X solves G-SUB(X), and no function f such that if G is in $\mathcal{G}$ then $\mathbb{A}$ runs in time $f(G)n^{o(tw(G))}$.*

Marx [11] came close to proving Conjecture 1.1 assuming the Exponential Time Hypothesis (ETH) [9], which is the hypothesis that solving 3SAT on n variables requires $2^{\Omega(n)}$ time. We state his result in terms of a parameter $emb(G)$ (short for "embedding") which we will define in Section 4:

▶ **Theorem 1.2** ([11])**.** *If there is a class $\mathcal{G}$ of graphs with unbounded treewidth, an algorithm $\mathbb{A}$ that on inputs G and X solves G-SUB(X), and a function f such that if G is in $\mathcal{G}$ then $\mathbb{A}$ runs in time $f(G)n^{o(emb(G))}$, then ETH is false.*

Marx [11] proved that $emb(G)$ is $\Omega(tw(G)/\log tw(G))$, so Theorem 1.2 comes within a logarithmic factor in the exponent of proving Conjecture 1.1. Our main result is a counterexample to an average-case analogue of Conjecture 1.1, in a sense that will be made precise in Section 3. Moreover, our result holds on circuits of depth depending only on G.

Li, Razborov and Rossman [10] proved that for fixed G, the average-case AC^0 complexity of G-SUB is between $n^{\kappa(G)-o(1)}$ and $n^{2\,\kappa(G)+c}$, where $\kappa(G)$ is a graph property defined in Section 3 and c is an absolute constant.[1] We tighten this gap, answering an open problem posed in [10]:

▶ **Theorem 1.3.** *There is a constant $c > 0$ such that for any fixed graph G, the average-case AC^0 complexity of G-SUB is at most $n^{\kappa(G)+c}$.*

We observe that a similar result holds easily on Turing machines, using as a subroutine the *sort-merge join* algorithm from relational algebra. This involves sorting, which cannot be done in AC^0 [7], so our circuit instead uses hashing that relies on concentration of measure for subgraphs of random graphs.

It was also proved in [10] that $\kappa(G)$ is between $\Omega(tw(G)/\log tw(G))$ and $tw(G)+1$, from which it follows that the *worst*-case complexity of G-SUB on bounded-depth circuits is at least $n^{\Omega(tw(G)/\log tw(G))}$. The question was posed in [10] of whether $\kappa(G)$ is $\Theta(tw(G))$; an affirmative answer would have implied that Conjecture 1.1 holds on bounded-depth circuits.

Our main result is a separation of κ from treewidth. The Hamming graph K_q^d has vertex set $\{1,\ldots,q\}^d$ and edges between every two vertices that differ in exactly one coordinate. It is already known that K_q^d has treewidth $\Theta\left(q^d/\sqrt{d}\right)$ [4]. We prove the following:

▶ **Theorem 1.4.** *$\kappa\left(K_q^d\right)$ is $\Theta(q^d/d)$.*

Thus, if G is the hypercube graph K_2^d for example, then $\kappa(G)$ is $\Theta\left(tw(G)/\sqrt{\log tw(G)}\right)$. It follows that an average-case analogue of Conjecture 1.1 is false if $\mathcal{G}$ is taken to be the set of all hypercubes. We also prove the following (for arbitrary graphs G):

▶ **Theorem 1.5.** *$emb(G)$ is $O(\kappa(G))$.*

[1] In [10], the parameter $\kappa(G)$ was called $\kappa_{\mathrm{col}}(G)$.

Because of Theorem 1.5, even if our upper bound generalizes to the worst case, it is still consistent with current knowledge (in particular Theorem 1.2) that ETH is true. Another consequence of Theorem 1.5 is that the lower bound from Theorem 1.2 holds unconditionally in AC^0.

It follows from Theorems 1.4 and 1.5 that if G is a hypercube then $emb(G) \leq O(\kappa(G)) = o(tw(G))$, so proving that Conjecture 1.1 holds under ETH cannot be done by proving that $emb(G)$ is $\Theta(tw(G))$. In fact, this conclusion was already known: Alon and Marx [1] proved that if G is a 3-regular expander then $emb(G)$ is $\Theta(tw(G)/\log tw(G))$. It was proved in [10] that if G is a 3-regular expander then $\kappa(G)$ is $\Theta(tw(G))$, which makes our separation of κ from treewidth more surprising. On the other hand, we will see that Theorem 1.5 is asymptotically tight in the case of Hamming graphs.

We can make a similar statement regarding AC^0. Amano [3] observed that the color-coding algorithm for G-SUB can be implemented by AC^0 circuits of size $O(n^{tw(G)+1})$ for fixed G. Our separation of κ from treewidth implies that if Conjecture 1.1 holds in AC^0, then this cannot be proved using average-case complexity as defined here and in [10].

The paper is organized as follows. In Section 2 we introduce some notation and definitions. In Section 3 we define the average-case problem and $\kappa(G)$, and give an $\tilde{O}(n^{\kappa(G)})$-time algorithm for the average-case problem. In Section 4 we define $emb(G)$ and prove that $emb(G)$ is $O(\kappa(G))$. In Section 5 we prove that $\kappa\left(K_q^d\right)$ is $\Theta(q^d/d)$, and obtain as a corollary that $emb\left(K_q^d\right)$ is $\Theta(q^d/d)$ as well. We also summarize the proof of Chandran and Kavitha [4] that $tw\left(K_q^d\right)$ is $\Theta\left(q^d/\sqrt{d}\right)$. In Section 6 we prove our AC^0 upper bound.

2 Preliminaries

It will be convenient to define $\tilde{O}(f(n)) = f(n)\log^{O(1)} n$. (This differs from the standard notation when $f(n) = n^{o(1)}$.) Let $[k] = \{1, \ldots, k\}$ for $k \in \mathbb{N}$.

We use **boldface** to denote random variables. The indicator variable $\mathbb{1}\{E\}$ equals 1 if the event E occurs and 0 otherwise. Expected value is denoted $\mathbb{E}[\cdot]$. An event occurs *asymptotically almost surely (a.a.s.)* if it occurs with probability $1 - o(1)$ as n goes to infinity.

2.1 Graphs

All graphs we consider are simple and undirected, and may have isolated vertices. If G is a graph then let $V(G)$ and $E(G)$ denote its vertex and edge sets, with respective cardinalities $v(G)$ and $e(G)$. If u and v are adjacent vertices then we denote the edge connecting them by uv or vu. A graph H is a subgraph of G, denoted $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

▶ **Definition 2.1** (Colored subgraph isomorphism problem). *For graphs G and X, where X comes with a coloring $\chi : V(X) \to V(G)$, the problem G-SUB(X) asks whether X has a subgraph G' such that χ (restricted to $V(G')$) is an isomorphism from G' to G. (Note that G' is not required to be an induced subgraph of X.)*

For $U \subseteq V(G)$ let $G[U]$ be the induced subgraph of G on U, and more generally let $G[U_1, \ldots, U_k] = G[U_1 \cup \cdots \cup U_k]$. Let $G - U = G[V(G) - U]$, and for $H \subseteq G$ let $G - H = G - V(H)$.

When the parent graph G is clear in context, let $\deg(u)$ be the degree of a vertex u, and for disjoint $S, T \subseteq V(G)$ let $e(S, T)$ be the number of edges between S and T. Similarly, for vertex-disjoint graphs A and B let $e(A, B) = e(V(A), V(B))$.

Let $G \cap H$ be the graph with vertex set $V(G) \cap V(H)$ and edge set $E(G) \cap E(H)$, and define $G \cup H$ similarly. Note that $G \cap H$ may have isolated vertices even if G and H do not. If $A \subseteq B$ are graphs then let $[A, B] = \{H \mid A \subseteq H \subseteq B\}$, and let $(A, B]$ be the same interval without A, etc.

We denote by K_k the complete graph on k vertices, also called the k-clique. The graph K_q^d has vertex set $[q]^d$, and two vertices are adjacent if and only if they differ in exactly one coordinate. Such graphs are called *Hamming graphs*. A special case is the d-dimensional hypercube $Q_d = K_2^d$; we will use $\{0,1\}^d$ for its vertex set.

Finally, let $\mathbf{ER}(n, p)$ be the Erdős-Rényi graph on n vertices in which each possible edge exists independently with probability p.

3 The Average-Case Problem and the Parameter $\kappa(G)$

3.1 Threshold Random Graphs

First we will define *threshold weightings*, which assign weights to the vertices and edges of a graph subject to certain constraints. Then we will define a family of random graphs for each threshold weighting. The content in this subsection is essentially all from [10].

▶ **Definition 3.1.** *A threshold weighting on a graph G is a pair $(\alpha, \beta) \in [0,1]^{V(G)} \times [0,2]^{E(G)}$ with the following property. For $H \subseteq G$ let $\alpha(H) = \sum_{u \in V(H)} \alpha(u)$ and $\beta(H) = \sum_{e \in E(H)} \beta(e)$, and let $\Delta(H) = \alpha(H) - \beta(H)$. Then, $\Delta(H) \geq 0$ for all $H \subseteq G$, and $\Delta(G) = 0$. Let $\theta(G)$ be the set of threshold weightings on G.*

We will often denote $\Delta = (\alpha, \beta)$ in a slight abuse of notation. (Since $\Delta(u) = \alpha(u)$ if u is a single vertex, the pair (α, β) is uniquely determined by Δ.) The requirement that α be nonnegative is redundant because it's a special case of the requirement that Δ be nonnegative. The requirement that $\beta \leq 2$ is also redundant because for every edge uv,

$$0 \leq \Delta(uv) = \alpha(u) + \alpha(v) - \beta(uv) \leq 2 - \beta(uv).$$

A trivial example is $(\alpha, \beta) = (0, 0)$, i.e. all vertices and edges have a weight of zero. The following example is more general:

▶ **Example 3.2** (Markov Chains)**.** Let $M \in \mathbb{R}_{\geq 0}^{V(G) \times V(G)}$ be a column stochastic matrix (meaning each column sums to 1) such that if $M_{u,v} \neq 0$ then either $u = v$ or $uv \in E(G)$. Let $\alpha(u) = 1 - M_{u,u}$ for all u, and $\beta(uv) = M_{u,v} + M_{v,u}$ for all $uv \in E(G)$. Then for all $H \subseteq G$,

$$\Delta(H) = \sum_{\substack{v \in V(H) \\ uv \in E(G) - E(H)}} M_{u,v} \geq 0, \tag{1}$$

with equality if $H = G$. In fact, in the full paper we prove that every threshold weighting is equivalent to at least one Markov Chain.

The following threshold weighting will be especially important, and can be thought of as representing a uniform random walk on G:

▶ **Definition 3.3.** *If G lacks isolated vertices then let $\Delta_o = (1, \beta_o) \in \theta(G)$ be the threshold weighting generated in Example 3.2 when $M_{u,v} = \mathbb{1}\{uv \in E(G)\}/\deg(v)$. That is, $\Delta_o = (\alpha, \beta)$, where $\alpha(u) = 1$ for all u and $\beta(uv) = 1/\deg(u) + 1/\deg(v)$ for all $u \neq v$. If G is d-regular then this simplifies to $\Delta_o = (1, \beta_o) = (1, 2/d)$.*

Now we define threshold random graphs:

▶ **Definition 3.4.** *For $\Delta = (\alpha, \beta) \in \theta(G)$ let $\mathbf{X}_{\Delta,n}$ be the graph with vertices u_i for $u \in V(G)$ and $i \in [n^{\alpha(u)}]$, and for $uv \in E(G)$, each edge $u_i v_j$ independently with probability $n^{-\beta(uv)}$. The graph $\mathbf{X}_{\Delta,n}$ comes with the coloring to G defined by $u_i \mapsto u$.*

For $H \subseteq G$ and X in the support of $\mathbf{X}_{\Delta,n}$, let $\mathrm{Sub}_X(H)$ be the set of subgraphs $H' \subseteq X$ such that the aforementioned coloring (restricted to $V(H')$) is an isomorphism from H' to H. We say that such a graph H' is "H-colored". Note that $\mathrm{Sub}_X(H)$ can be identified with a subset of $\prod_{u \in V(H)} [n^{\alpha(u)}]$.

▶ **Lemma 3.5.** *If $\Delta \in \theta(G)$ and $H \subseteq G$ then $\mathbb{E}[|\mathrm{Sub}_{\mathbf{X}_{\Delta,n}}(H)|] = n^{\Delta(H)}(1 \pm o(1))$.*

Proof. Let $(\alpha, \beta) = \Delta$. The set $\mathrm{Sub}_{\mathbf{X}_{\Delta,n}}(H)$ contains each of its $n^{\alpha(H)}$ possible elements with probability $n^{-\beta(H)}$, so the result follows from linearity of expectation. (The $1 \pm o(1)$ accounts for having to round $n^{\alpha(\cdot)}$ to an integer.) ◀

Lemma 3.5 motivates the requirements that Δ be nonnegative everywhere and that $\Delta(G) = 0$. Recall that the problem G-SUB(X) asks whether $\mathrm{Sub}_X(G)$ is the empty set. Since $\Delta(G)$ is required to be zero, it follows that $\mathrm{Sub}_{\mathbf{X}_{\Delta,n}}(G)$ has (approximately) one element on average, and the probability that $\mathrm{Sub}_{\mathbf{X}_{\Delta,n}}(G)$ is empty is known to be bounded away from 0 and 1 as n goes to infinity [10].

3.2 The Parameter $\kappa(G)$ and an Algorithm for the Average Case

We now define $\kappa(G)$:

▶ **Definition 3.6** ([10])**.** *Let G be a graph with no isolated vertices. Let $\mathrm{Seq}(G)$ be the set of* union sequences*, meaning sequences $(H_1, \ldots, H_k)$ of distinct subgraphs of G such that $H_k = G$ and each H_i is either an edge or the union of two previous graphs in the sequence. For $\Delta \in \theta(G)$ let $\kappa_\Delta(G) = \min_{S \in \mathrm{Seq}(G)} \max_{H \in S} \Delta(H)$. Finally, let $\kappa(G) = \max_{\Delta \in \theta(G)} \kappa_\Delta(G)$.*

To simplify the exposition, whenever we refer to $\kappa(G)$, the graph G is implicitly assumed to lack isolated vertices. It was proved in [10] that for any fixed G, constant-depth circuits solving G-SUB$(\mathbf{X}_{\Delta,n})$ a.a.s. require size at least $n^{\kappa_\Delta(G) - o(1)}$ and at most $n^{2\kappa_\Delta(G)+c}$ (where c is an absolute constant). The results about average-case complexity described in Section 1 are with respect to a Δ such that $\kappa_\Delta(G) = \kappa(G)$.

▶ **Theorem 3.7.** *The problem G-SUB$(\mathbf{X}_{\Delta,n})$ can be solved in time $\tilde{O}(n^{\kappa_\Delta(G)}) \leq \tilde{O}(n^{\kappa(G)})$ a.a.s. for any fixed G.*

Proof. First we prove a weaker upper bound of $\tilde{O}(n^{2\kappa_\Delta(G)})$, in a manner analogous to the circuit from [10], and then we describe a modification (on Turing machines) that removes the factor of 2 from the exponent. In Section 6 we will remove the factor of 2 in AC^0 using a different approach.

Let S be a union sequence such that $\kappa_\Delta(G) = \max_{H \in S} \Delta(H)$. For any $H \in S$, by Lemma 3.5 and Markov's Inequality, $P\left(|\mathrm{Sub}_{\mathbf{X}_{\Delta,n}}(H)| > n^{\Delta(H)} \log n\right) \leq 1/\log n$. (We will obtain a tighter bound of $P(|\mathrm{Sub}_{\mathbf{X}_{\Delta,n}}(H)| > \tilde{O}(n^{\Delta(H)})) \leq n^{-\omega(1)}$ in Section 6.1.) By a union bound it follows that if $X \sim \mathbf{X}_{\Delta,n}$ then $\max_{H \in S} |\mathrm{Sub}_X(H)| \leq \tilde{O}(n^{\kappa_\Delta(G)})$ a.a.s. Assume this condition holds for X. For each successive H in S, compute $\mathrm{Sub}_X(H)$ as follows. If H is a single edge then this is trivial. Otherwise $H = A \cup B$ for some previous $A, B \in S$, in which case $\mathrm{Sub}_X(H)$ is the set of $\mathcal{A} \cup \mathcal{B}$ such that $\mathcal{A} \in \mathrm{Sub}_X(A), \mathcal{B} \in \mathrm{Sub}_X(B)$ and the projections of $\mathcal{A}$ and $\mathcal{B}$ onto $[n]^{V(A \cap B)}$ are equal. Therefore $\mathrm{Sub}_X(H)$ can be computed by brute force in time $\tilde{O}(|\mathrm{Sub}_X(A)| \cdot |\mathrm{Sub}_X(B)|) \leq \tilde{O}(n^{2\kappa_\Delta(G)})$. Finally, check whether $\mathrm{Sub}_X(G)$ is empty.

We can save a quadratic factor by computing $\mathrm{Sub}_X(H)$ from $\mathrm{Sub}_X(A)$ and $\mathrm{Sub}_X(B)$ as follows. (This is a case of the *sort-merge join* algorithm for computing the *natural join* of two relations, as defined in database theory [20].) Fix an efficiently computable total order on $[n]^{V(A\cap B)}$, e.g. interpret elements of $[n]^{V(A\cap B)}$ as $v(A\cap B)$-digit base-n numbers in increasing order, and then define a partial order on $[n]^{V(A)} \cup [n]^{V(B)}$ by first projecting onto $[n]^{V(A\cap B)}$. Sort $\mathrm{Sub}_X(A)$ and $\mathrm{Sub}_X(B)$ in nondecreasing order, and for convenience add the symbol $\perp$ to the end of both sorted lists. Let $\mathcal{A}$ and $\mathcal{B}$ be the first elements of $\mathrm{Sub}_X(A)$ and $\mathrm{Sub}_X(B)$ respectively, and initialize an empty accumulator (which will ultimately equal $\mathrm{Sub}_X(H)$). While $\mathcal{A} \neq \perp$ and $\mathcal{B} \neq \perp$, do the following. If $\mathcal{A} < \mathcal{B}$ then let $\mathcal{A}$ be the next element of $\mathrm{Sub}_X(A)$. If $\mathcal{B} < \mathcal{A}$ then let $\mathcal{B}$ be the next element of $\mathrm{Sub}_X(B)$. Otherwise, let $\mathcal{B}' = \mathcal{B}$, and while $\mathcal{B}' \neq \perp$ and the projections of $\mathcal{A}$ and $\mathcal{B}'$ onto $[n]^{V(A\cap B)}$ are equal, add $\mathcal{A} \cup \mathcal{B}'$ to the accumulator and let $\mathcal{B}'$ be the next element of $\mathrm{Sub}_X(B)$. Then (once the procedure involving $\mathcal{B}'$ has finished) let $\mathcal{A}$ be the next element of $\mathrm{Sub}_X(A)$.

Sorting $\mathrm{Sub}_X(A)$ and $\mathrm{Sub}_X(B)$ takes $\tilde{O}(|\mathrm{Sub}_X(A)| + |\mathrm{Sub}_X(B)|)$ comparisons, and then computing $\mathrm{Sub}_X(H)$ takes $\tilde{O}(|\mathrm{Sub}_X(A)| + |\mathrm{Sub}_X(B)| + |\mathrm{Sub}_X(H)|) \leq \tilde{O}(n^{\kappa_\Delta(G)})$ time. ◀

We will use the following graph-theoretic properties of $\kappa(G)$:

▶ **Theorem 3.8** ([10][2]). *Let G be a graph with no isolated vertices.*

- **(i)** *There exists $\Delta = (1, \beta) \in \theta(G)$ (meaning $\Delta(u) = 1$ for all vertices u) such that $\kappa(G) = \kappa_\Delta(G)$.*
- **(ii)** *$\kappa(G) \geq v(G)h(G)/(3\max_{u\in V(G)} \deg(u))$, where $h(G)$ is the edge expansion of G.*
- **(iii)** *If G is a minor of some graph H then $\kappa(G) \leq \kappa(H)$.*

The following was observed in [10] as well:

▶ **Corollary 3.9.** *If G is a bounded-degree expander then $\kappa(G)$ is $\Theta(v(G))$.*

Proof. Theorem 3.8(ii) implies that $\kappa(G)$ is $\Omega(v(G))$. Recall from Section 1 that $\kappa(G) \leq tw(G) + 1$ [10], and it is well known that $tw(G) + 1 \leq v(G)$. ◀

4 The Parameter $emb(G)$ and Proof that $emb(G)$ is $O(\kappa(G))$

Recall that $emb(G)$ is significant because of its role in Marx's ETH-hardness result for G-SUB, namely Theorem 1.2.

▶ **Definition 4.1** ($emb(G)$). *Let $G^{(q)}$ be the graph formed by replacing each vertex of G with a q-clique, i.e. it has vertices u_i for all $u \in V(G)$ and $i \in [q]$, and edges u_iv_j for all $u_i \neq v_j$ such that either $u = v$ or $uv \in E(G)$. Let $emb(G)$ be the supremum of all $r > 0$ for which there exists $m_0 = m_0(G, r)$ such that if H is any graph with $m \geq m_0$ edges and no isolated vertices, then H is a minor of $G^{(\lceil m/r\rceil)}$, and furthermore a minor mapping from H to $G^{(\lceil m/r\rceil)}$ can be computed in time $f(G)m^{O(1)}$ for some function f.*

Although the requirement that such a minor mapping be efficiently computable is crucial in Theorem 1.2, none of the other results about $emb(G)$ that we reference or derive depend on this requirement, so we may safely ignore it going forward. The following example illustrates Definition 4.1:

[2] Specifically, Corollary 4.2, Theorem 4.9, and Theorem 5.1 of [10] correspond to Theorems 3.8(i) to 3.8(iii) respectively.

▶ **Example 4.2** ($emb(K_k)$ [11]). Since $K_k^{(\lceil m/r \rceil)} = K_{k\lceil m/r \rceil}$, any graph H with m edges is a minor of $K_k^{(\lceil m/r \rceil)}$ if and only if $v(H) \leq k\lceil m/r \rceil$. If H has no isolated vertices then H could have up to $2m$ vertices, so $2m \leq k\lceil m/r \rceil$. Therefore $emb(K_k) = k/2$: it is sufficient for $2m$ to be at most km/r (i.e. $r \leq k/2$), and no $r > k/2$ satisfies $2m \leq k\lceil m/r \rceil$ for arbitrarily large m.

▶ Remark. The name $emb(G)$ comes from the fact that Marx [11] called a minor mapping from H to $G^{(q)}$ an "embedding of depth q" from H into G. Marx [11] used the notation $G^{(q)}$, but the parameter $emb(G)$ is new in the current paper, all results about $emb(G)$ in [11, 1] having been stated in terms of embeddings of some depth.

The following is used in our proof that $emb(G)$ is $O(\kappa(G))$:

▶ **Lemma 4.3.** $\kappa\left(G^{(q)}\right) \leq q \max(\kappa(G), 2)$.

Proof Sketch. Given a threshold weighting Δ on $G^{(q)}$, collapsing each cluster of q vertices to a single "mega-vertex" induces a threshold weighting Δ' on G. Let S be an optimal union sequence for G with respect to Δ', and project S back onto $G^{(q)}$. ◀

Now we prove that $emb(G)$ is $O(\kappa(G))$ (Theorem 1.5), using an argument similar to the proof by Marx [11] that $emb(G)$ is $O(tw(G))$:

Proof. Let $r > 0$, and assume there exists an arbitrarily large 3-regular expander H that's a minor of $G^{(\lceil e(H)/r \rceil)}$. Then by Corollary 3.9, Theorem 3.8(iii), and Lemma 4.3,

$$e(H) = \Theta(v(H)) = \Theta(\kappa(H)) \leq O\left(\kappa\left(G^{(\lceil e(H)/r \rceil)}\right)\right) \leq O\left(\kappa(G)e(H)/r\right),$$

so r must be $O(\kappa(G))$. ◀

In [10] the question was posed of whether Theorem 1.2 holds with $\kappa(G)$ in place of $emb(G)$. By Theorem 1.5 this would be a stronger bound, which makes the question even more interesting. This problem is open even in the case of 3-regular expanders: recall from Section 1 that if G is a 3-regular expander then $emb(G)$ is $\Theta(tw(G)/\log tw(G))$ and $\kappa(G)$ is $\Theta(tw(G))$ [1, 10].

The fact that $\kappa(G)$ is $\Omega(emb(G))$ gives an alternate proof, besides the one in [10], that $\kappa(G)$ is $\Omega(tw(G)/\log tw(G))$.

5 Separating κ from Treewidth

In Section 5.1 we prove that $\kappa(K_k) = k/4 + O(1)$, which is a special case of the more general result that $\kappa\left(K_q^d\right) = \Theta(q^d/d)$. We obtain tighter multiplicative constants in the case $d = 1$, and it provides an opportunity to illustrate the main ideas of our proof in a simpler setting, but when reading the full paper it may be skipped without penalty. In Section 5.2 we prove that $\kappa\left(K_q^d\right)$ is $O(q^d/d)$ when q is even, which is sufficient to separate κ from treewidth. Again, this case is cleaner than the general case and conveys most of the intuition behind it. In an appendix in the full paper we prove that $\kappa\left(K_q^d\right)$ is $O(q^d/d)$ for all q. In Section 5.3 we prove that $\kappa\left(K_q^d\right)$ is $\Omega(q^d/d)$ in two different ways, completing the proof that $\kappa\left(K_q^d\right)$ is $\Theta(q^d/d)$ (Theorem 1.4), and we obtain as a corollary that $emb\left(K_q^d\right)$ is $\Theta(q^d/d)$ as well. In Section 5.4 we summarize the proof of Chandran and Kavitha [4] that $tw\left(K_q^d\right)$ is $\Theta\left(q^d/\sqrt{d}\right)$.

5.1 Proof that $\kappa(K_k) = k/4 + O(1)$

▶ Remark. It was already observed in [10] that $\kappa(K_k)$ is $\Theta(k)$.

Rossman [16] proved that $\kappa_{\Delta_o}(K_k) \geq k/4$, so it suffices to prove the upper bound. By Theorem 3.8(i) it suffices to prove that $\kappa_{\Delta}(K_k) \leq k/4 + O(1)$ for an arbitrary $\Delta = (1, \beta) \in \theta(G)$. First we construct a sequence $U_1 \subseteq \cdots \subseteq U_k = V(K_k)$ such that U_i is an i-element subset of $V(K_k)$, and $\beta(K_k[U_i]) \geq \beta_o(K_k[U_i])$ for all i. The set $U_k = V(K_k)$ satisfies this requirement because $\beta(K_k)$ and $\beta_o(K_k)$ are both equal to k. Given U_i, let $\mathbf{U}_{i-1}$ be an $(i-1)$-element subset of U_i chosen uniformly at random. Each pair of elements in U_i is included in $\mathbf{U}_{i-1}$ with the same probability p_i $(= 1 - 2/i)$, so it follows from linearity of expectation that

$$\mathbb{E}[\beta(K_k[\mathbf{U}_{i-1}])] = \sum_{e \in E(K_k[U_i])} \beta(e) p_i = p_i \beta(K_k[U_i]) \geq p_i\, \beta_o(K_k[U_i]) = \mathbb{E}[\beta_o(K_k[\mathbf{U}_{i-1}])].$$

Therefore there exists a fixed U_{i-1} such that $\beta(K_k[U_{i-1}]) \geq \beta_o(K_k[U_{i-1}])$.

We construct a union sequence S for K_k as follows. Start by enumerating the edges, and then for i from 1 to $k-1$, append $(K_k[U_i] \cup e_1, K_k[U_i] \cup e_1 \cup e_2, \ldots, K_k[U_{i+1}])$, where $e_1, e_2, \ldots$ are the edges between U_i and $U_{i+1} - U_i$. Then,

$$\max_{H \in S} \Delta(H) \leq \max_i \Delta(K_k[U_i]) + 1 \leq \max_i \Delta_o(K_k[U_i]) + 1.$$

As observed in [16], it follows from Equation (1) that $\Delta_o(K_k[U_i]) = i(k-i)/k$, which is at most $k/4$ (when $i = k/2$). Therefore $\kappa_{\Delta}(K_k) \leq k/4 + 1$.

5.2 Proof that $\kappa\left(K_q^d\right)$ is $O(q^d/d)$ if q is Even

First we reduce this to the case $q = 2$. The graph K_q^d is a subgraph of $Q_d^{\left((q/2)^d\right)}$ (recall Definition 4.1), as explained in the full paper. By Theorem 3.8(iii) and Lemma 4.3, if $\kappa(Q_d)$ is $O(2^d/d)$ then

$$\kappa\left(K_q^d\right) \leq \kappa\left(Q_d^{\left((q/2)^d\right)}\right) \leq O\left(\left(\frac{q}{2}\right)^d \kappa(Q_d)\right) \leq O\left(\left(\frac{q}{2}\right)^d \frac{2^d}{d}\right) = O(q^d/d).$$

Now we prove that $\kappa(Q_d)$ is $O(2^d/d)$, following some brief definitions and a high-level overview of the argument. Fix d. We identify each $u \in \{0,1\}^d$ with $\sum_{i=0}^{d-1} u_i 2^i$. For $0 \leq a \leq 2^d$ let $G(a) = Q_d[0, \ldots, a-1]$. Recall that $\Delta_o = (1, \beta_o) = (1, 2/d)$ is a threshold weighting on Q_d (Definition 3.3). Let $\mu = \max_{0 \leq a \leq 2^d} \Delta_o(G(a))$.

▶ Remark. The intuition behind μ is as follows. The reader may note that $\kappa_{\Delta_o}(Q_d) \leq \mu + 1$, by reasoning analogous to that in Section 5.1. That is, for each vertex u of Q_d in increasing lexicographic order, add to an accumulator all edges uv for which $v < u$.

There is another union sequence captured by μ as well. If a subgraph $B \subseteq Q_d$ isomorphic to Q_k for some k, then since Q_k is isomorphic to $G(2^k)$ (and β_o is uniform) it follows that $\Delta_o(B) \leq \mu$. Consider a depth-d binary tree in which each node at depth k is a subgraph of Q_d isomorphic to Q_{d-k} (in particular, the root is Q_d and the leaves are vertices), and each interior node is the union of its two children along with some additional edges corresponding to a coordinate cut. This tree describes a union sequence S for Q_d: recursively obtain the graphs L and R corresponding to the children of Q_d, and then take $L \cup R$ and add the missing edges. Note that $\max_{H \in S} \Delta_o(H) = 2 \max_{0 \leq k \leq d} \Delta_o(G(2^k)) \leq 2\mu$.

Analogous to Section 5.1, the upper bound is obtained by comparing $\kappa_\Delta(Q_d)$ to μ for each Δ, and bounding μ. For this purpose we will consider the two union sequences mentioned above, as well as hybrids of them.

In the full paper we prove that $\kappa(Q_d)$ is $O(\mu)$. It follows from Equation (1) that $\mu = \max_a \Delta_{\text{o}}(G(a)) = \max_a e(G(a), Q_d - G(a))/d$, and in the full paper we prove that $\max_a e(G(a), Q_d - G(a))$ is $O(2^d)$.

5.3 Proof that $\kappa\left(K_q^d\right)$ is $\Omega(q^d/d)$ and $emb\left(K_q^d\right)$ is $\Theta(q^d/d)$

Alon and Marx [1, Theorem 4.3] proved that $emb\left(K_q^d\right)$ is $\Omega(q^d/d)$, and it follows from Theorem 1.5 that $emb\left(K_q^d\right) \leq O\left(\kappa\left(K_q^d\right)\right) \leq O(q^d/d)$. Therefore $emb\left(K_q^d\right)$ is $\Theta(q^d/d)$.

It is implicit in the above argument that $\kappa\left(K_q^d\right) \geq \Omega\left(emb\left(K_q^d\right)\right) \geq \Omega(q^d/d)$. In the full paper we present a second proof that $\kappa\left(K_q^d\right)$ is $\Omega(q^d/d)$, using Theorem 3.8(ii).

5.4 Proof that $tw\left(K_q^d\right)$ is $\Theta\left(q^d/\sqrt{d}\right)$, Summarized

(See [4] for the full proof.) The proof that $tw\left(K_q^d\right)$ is $O\left(q^d/\sqrt{d}\right)$ reduces to the case $q = 2$ by reasoning analogous to that in the beginning of Section 5.2. For $k \in [d]$ let U_k be the set of vertices of Q_d with exactly k or $k-1$ ones. The path $(U_1, \ldots, U_d)$ is a tree decomposition of Q_d with width approximately $2\binom{d}{d/2}$, and by Stirling's approximation this is $\Theta\left(2^d/\sqrt{d}\right)$.[3]

For a graph G let $\phi(G)$ be the minimum over all $U \subseteq V(G), v(G)/4 \leq |U| \leq v(G)/2$ of the number of vertices in $V(G) - U$ with at least one neighbor in U. From a result of Robertson and Seymour [15] it follows that $tw(G) \geq \phi(G) - 1$, and from a result of Harper [6] it follows that $\phi\left(K_q^d\right)$ is $\Omega\left(q^d/\sqrt{d}\right)$. (Also note the parallels between $tw(G) \geq \phi(G) - 1$ and Theorem 3.8(ii); interestingly, we've sign that both are tight to within a constant factor in the case of K_q^d.)

6 AC^0 Upper Bound

An AC^0 circuit is a constant-depth circuit with polynomially many unbounded-fanin AND and OR gates and NOT gates. Fix a graph G and threshold weighting $\Delta \in \theta(G)$ for the remainder of this section. We prove the following, which is a more precise statement of Theorem 1.3:

▶ **Theorem 6.1.** *There exists a constant-depth circuit with $n^{\kappa_\Delta(G)+c}$ wires that solves G-SUB$(\mathbf{X}_{\Delta,n})$ with probability $1 - n^{\omega(1)}$, where $c > 0$ is an absolute constant.*

Since in any circuit the number of gates is at most one plus the number of wires, the circuit from Theorem 6.1 has size $n^{\kappa_\Delta(G)+O(1)} \leq n^{\kappa(G)+O(1)}$. (In this discussion, all $\pm O(1)$ terms in an exponent are independent of G.) For comparison, it was proved in [10] (building on a line of previous work [16, 3, 17, 13]) that the average-case AC^0 complexity of G-SUB$(\mathbf{X}_{\Delta,n})$ is between $n^{\kappa_\Delta(G)-o(1)}$ and $n^{2\kappa_\Delta(G)+O(1)}$. Another related result, regarding the uncolored k-clique problem, is that the average-case AC^0 complexity of K_k-SUB$_{\text{uncol}}\left(\mathbf{ER}\left(n, n^{-2/(k-1)}\right)\right)$ is at most $n^{k/4+O(1)}$ [3, 18] $(= n^{\kappa(K_k)\pm O(1)}$ by Section 5.1). See [19] for a survey of the average-case circuit complexity of subgraph isomorphism more generally.

[3] Compared to the tree decomposition from [4], this one is a simpler variant whose width is larger by up to a constant factor.

▶ **Definition 6.2.** *Let X be in the support of $\mathbf{X}_{\Delta,n}$, and let $U \subseteq G$ be an arbitrary graph (which we think of as a "universe"). Let $\mathrm{Sub}_n(U)$ be the set of all possible elements of $\mathrm{Sub}_{\mathbf{X}_{\Delta,n}}(U)$; note that this can be identified with $\prod_{v \in V(U)} [n^{\alpha(v)}]$. If $A \subseteq U$ and $\mathcal{A} \in \mathrm{Sub}_n(A)$ then let $\mathcal{A}$* extend to U in X *if there exists a graph $\mathcal{U} \in \mathrm{Sub}_X(U)$ (called a U-*extension of $\mathcal{A}$*) such that $\mathcal{A} \subseteq \mathcal{U}$. (In context, X or $\mathbf{X}$ will be implicit.) Equivalently, $\mathcal{A}$ could be required to be in $\mathrm{Sub}_X(A)$ rather than $\mathrm{Sub}_n(A)$ in the latter definition.*

Let $\Delta^*_U(A) = \min_{A \subseteq H \subseteq U} \Delta(H)$. Let X be good *if for all graphs $U \subseteq G$ and $A \subseteq U$, and for all $\mathcal{A} \in \mathrm{Sub}_n(A)$ and vertices $v \in V(U) - V(A)$, there are $\tilde{O}\left(n^{\Delta^*_U(A \cup v) - \Delta^*_U(A)}\right)$ values of $i \in [n^{\alpha(v)}]$ such that $\mathcal{A} \cup v_i$ extends to U. (Recall our unconventional definition of $\tilde{O}(\cdot)$ from Section 2, e.g. $\tilde{O}(1)$ denotes $\log^{O(1)} n$.) Finally, let an event occur* with high probability (w.h.p.) *if it occurs with probability $1 - n^{-\omega(1)}$.*

We prove the following:

▶ **Theorem 6.3.** *The graph $\mathbf{X}_{\Delta,n}$ is good w.h.p.*

Observe that this is a substantially stronger concentration bound than the application of Markov's Inequality in the proof of Theorem 3.7. In Section 6.1 we prove Theorem 6.3, and then in Section 6.2 we use this result to prove Theorem 6.1.

6.1 Proof of Theorem 6.3

First we derive some algebraic properties of the threshold weighting Δ.

▶ **Lemma 6.4.** *If $A, B \subseteq G$ then $\Delta(A) + \Delta(B) = \Delta(A \cap B) + \Delta(A \cup B)$.*

Proof. Each vertex or edge in one (resp. two) of A and B is also in one (resp. two) of $A \cap B$ and $A \cup B$. ◀

▶ **Definition 6.5.** *For $A \subseteq U \subseteq G$ let $\Gamma_U(A) = \bigcap\{H \in [A, U] \mid \Delta(H) = \Delta^*_U(A)\}$, and let A be a U-*base *if $\Delta(A) = \Delta^*_U(A)$.*

Throughout this subsection, U will be an arbitrary subgraph of G unless additional structure is imposed on it, and missing subscripts on Δ^* and Γ default to U.

▶ **Lemma 6.6.** *If $A \subseteq U$ then $\Delta(\Gamma(A)) = \Delta^*(A)$ and $A \subseteq \Gamma(A)$.*

Proof. It suffices to show that the set $S = \{H \in [A, U] \mid \Delta(H) = \Delta^*(A)\}$ is closed under intersection. Let $B, C \in S$. By the definition of S, Lemma 6.4, and the fact that $A \subseteq B \cup C$,

$$2\Delta^*(A) = \Delta(B) + \Delta(C) = \Delta(B \cap C) + \Delta(B \cup C) \geq \Delta(B \cap C) + \Delta^*(A),$$

so $\Delta(B \cap C) \leq \Delta^*(A)$. On the other hand, $\Delta(B \cap C) \geq \Delta^*(A)$ because $A \subseteq B \cap C$. Therefore $\Delta(B \cap C) = \Delta^*(A)$, so $B \cap C \in S$. ◀

The proofs of the following two lemmas are of a similar flavor, and are included in the full paper.

▶ **Lemma 6.7.** *If $A \subseteq \Gamma(A) \subseteq U' \subseteq U$ then $\Gamma(A)$ is a U'-base.*

▶ **Lemma 6.8.** *If $A \subseteq B \subseteq U$ then $\Gamma(A) \subseteq \Gamma(B)$.*

We now analyze the concentration of $\mathbf{X}_{\Delta,n}$, making liberal use of the fact that if $n^{O(1)}$ events occur with uniformly high probability then their conjunction also occurs w.h.p. by a union bound. For the rest of this subsection, "extensions" are with respect to an implicit $\mathbf{X} \equiv \mathbf{X}_{\Delta,n}$.

▶ **Lemma 6.9.** *If $A \subseteq U$ and $\Gamma_U(A) = U$ (i.e. $\Delta(H) > \Delta(U)$ for all $H \in [A, U)$) then the number of U-extensions of any $\mathcal{A} \in \mathrm{Sub}_n(A)$ is $\tilde{O}(1)$ w.h.p.*

(The above conditions are equivalent because, by the definition of $\Gamma(A)$, we have $\Gamma(A) = U$ if and only if U is the unique $H \in [A, U]$ that minimizes $\Delta(H)$.)

Proof Sketch. Here we prove the weaker claim that the lemma holds with "a.a.s." in place of "w.h.p." There are $n^{\alpha(U)-\alpha(A)}$ possible U-extensions of $\mathcal{A}$, each of which occurs with probability $n^{-\beta(U)+\beta(A)}$, so $\mathcal{A}$ has $n^{\Delta(U)-\Delta(A)}$ U-extensions in expection. Since $\Delta(U) < \Delta(A)$ by assumption, the result follows from Markov's Inequality. ◀

▶ **Lemma 6.10.** *If A is a U-base then any $\mathcal{A} \in \mathrm{Sub}_n(A)$ has $\tilde{O}(n^{\Delta(U)-\Delta(A)})$ U-extensions w.h.p.*

Proof Sketch. Again, if we replace "w.h.p." with "a.a.s." then the claim follows immediately from Markov's Inequality. A similar lower bound is also proved in an appendix in the full paper. ◀

Now we prove that $\mathbf{X}_{\Delta,n}$ is good w.h.p.:

Proof of Theorem 6.3. Let $A \subseteq U$, $\mathcal{A} \in \mathrm{Sub}_n(A)$ and $v \in V(U) - V(A)$. By a union bound it suffices to prove that w.h.p. there are $\tilde{O}(n^{\Delta^*(A\cup v)-\Delta^*(A)})$ values of i such that $\mathcal{A} \cup v_i$ extends to U. The number of such i is at most the number of i such that $\mathcal{A} \cup v_i$ extends to $\Gamma(A \cup v)$, which is at most the number of $\Gamma(A \cup v)$-extensions of $\mathcal{A}$. Since $\Gamma(A) \subseteq \Gamma(A \cup v)$ (Lemma 6.8), this equals the sum over all $\gamma \in \{\Gamma(A)\text{-extensions of } \mathcal{A}\}$ of the number $\mathbf{E}_\gamma$ of $\Gamma(A \cup v)$-extensions of γ.

It follows from Lemma 6.9 that $\mathcal{A}$ has $\tilde{O}(1)$ extensions to $\Gamma(A)$ w.h.p. (To see this, note that if $A \subseteq H \subset \Gamma(A)$ then $\Delta(H) \geq \Delta^*(A) = \Delta(\Gamma(A))$ (Lemma 6.6), and if $\Delta(H) = \Delta^*(A)$ then it follows from the definition of $\Gamma(A)$ that $\Gamma(A) \subseteq H$, a contradiction.) Since $\Gamma(A)$ is a $\Gamma(A \cup v)$-base (Lemma 6.7), it follows from Lemma 6.10 that any $\mathbf{E}_\gamma$ is $\tilde{O}(n^{\Delta(\Gamma(A\cup v))-\Delta(\Gamma(A))})$ w.h.p. $(= \tilde{O}(n^{\Delta^*(A\cup v)-\Delta^*(A)})$ by Lemma 6.6). ◀

6.2 The Circuit

If D is a data structure then let $|D|$ denote the number of bits used to represent it according to whatever schema we describe. When there is a null element we represent it by the all-zeros string.

Proof of Theorem 6.1. Since $\mathbf{X}_{\Delta,n}$ is good w.h.p. (Theorem 6.3) it suffices to prove the existence of a (small, constant-depth) circuit C such that $P_{X\sim\mathbf{X}_{\Delta,n}}(\mathsf{C}(X) = G\text{-}\mathsf{SUB}(X) \mid X \text{ is good}) = 1 - n^{\omega(1)}$. By Yao's Principle [21] it suffices to prove the existence of a (small, constant-depth) random circuit $\mathbf{C}$ such that $P(\mathbf{C}(X) = G\text{-}\mathsf{SUB}(X)) = 1 - n^{-\omega(1)}$ for any fixed good X.

The following result is essentially implicit in [10] (as is the argument above) and helps keep the random circuit small:

▶ **Lemma 6.11** (Random Hashing)**.** *Let S be a set containing a null element, and assume all elements of S are represented using the same number of bits. Let $l = l(n)$ and $m = m(n)$ be polynomially-bounded functions of n. Then there exists a random, constant-depth circuit $\mathbf{C} : S^l \to S^{\tilde{O}(m)}$ such that if A is an array of l values in S, of which all but at most m are null, then $\mathbf{C}$ has at most $|A|n^{o(1)}$ gates and $|A|\tilde{O}(l/m)$ wires, and w.h.p. the multiset of non-null elements of $\mathbf{C}(A)$ is the same as that of A.*

We remark that Lemma 6.11 will only be called with $l \leq \tilde{O}(n)$.

Proof Sketch. The proof uses a Chernoff bound and a result from [8]. ◀

Given $H \subseteq G$ and an ordering $\pi = (\pi^1, \ldots, \pi^{v(H)})$ of $V(H)$, let $\delta_i = \Delta^*_H(\pi^1 \cup \cdots \cup \pi^i)$ for $0 \leq i \leq v(H)$, and let $\phi_i = \delta_{i+1} - \delta_i$ for $0 \leq i < v(H)$. (In context H and π will be implicit.)

▶ **Lemma 6.12.** *$0 \leq \phi_i \leq 1$ for all i.*

Proof. Clearly $\delta_i \leq \delta_{i+1}$. Let $A \subseteq G$ such that $\pi^1, \ldots, \pi^i \in V(A)$ and $\Delta(A) = \delta_i$. Then $\delta_{i+1} \leq \Delta(A \cup \pi^{i+1}) \leq \Delta(A) + \alpha(\pi^{i+1}) \leq \delta_i + 1$. ◀

Let $T = T(H, \pi)$ be a depth-$v(H)$ tree (i.e. the root has depth 0 and the leaves have depth $v(H)$) in which each node at depth $i < v(H)$ has $n^{\phi_i} \log^{c_i} n$ children, where c_i is a sufficiently large constant. Each non-root node N has a *partial label* $\mathcal{L}(N) \in \{\text{null}\} \cup [n]$, and N's (complete) *label* is the sequence of partial labels along the path from the root to N. A label is considered null if it includes any null partial labels. It is required that no two nodes share a non-null label, and there exists a node labeled with $(l_1, \ldots, l_i)$ if and only if[4] $\{\pi^1_{l_1}, \ldots, \pi^i_{l_i}\}$ extends to H.

Let S be an *immediate subtree* of T (resp. of a node N), denoted $S \in T$ (resp. $S \in N$), if S's root is a child of T's root (resp. of N). Any subtree is considered to have the same (partial) label as its root.

If the underlying tree structure of T (that is, everything except the partial labels) is implicit, then we can represent T by an array of values in $\{\text{null}\} \cup [n]$, indexed by the vertices of T. Each of these values can be associated with a bit string in a natural way. We will consider circuits that compute T according to this representation.

▶ **Lemma 6.13.** *$|T|$ is $\tilde{O}(n^{\Delta(H)})$.*

Proof. $\delta_0 = \Delta(\emptyset) = 0$ and $\delta_{v(H)} = \Delta^*_H(V(H)) = \Delta(H)$. It takes $\tilde{O}(1)$ bits to store an element of $[n]^{V(H)}$, and each ϕ_i is nonnegative (Lemma 6.12), so

$$|T| = \tilde{O}\left(\prod_{i=0}^{v(H)-1} n^{\phi_i}\right) = \tilde{O}\left(n^{\sum_{i=0}^{v(H)-1} \phi_i}\right) = \tilde{O}\left(n^{\delta_{v(H)} - \delta_0}\right) = \tilde{O}\left(n^{\Delta(H)}\right). \quad \blacktriangleleft$$

▶ **Lemma 6.14.** *For all $H \subseteq G$ there exists a random, constant-depth circuit with $\tilde{O}(n^{\Delta(H)+2})$ wires, independent of X, that computes $T(H, \pi')$ from $T(H, \pi)$ w.h.p.*

Proof Sketch. Assume that π and π' differ only in positions d and $d+1$. (The general case can be reduced to at most $\binom{v(H)}{2}$ copies of this circuit in succession.) Define δ'_i and ϕ'_i analogously to δ_i and ϕ_i, but with respect to π' rather than π. Clearly $\delta_i = \delta'_i$ for $i \neq d$, so $\phi_i = \phi'_i$ for $i \notin \{d-1, d\}$.

For each depth-$(d-1)$ node N of $T(H, \pi)$, in parallel, do the following. For $\tau \in N, j \in [n]$ let $A_{\tau j}$ be (if this exists) the subtree rooted at a child of τ whose partial label is j. After updating the partial labels at what will become the new depth-d and depth-$(d+1)$ nodes, hash the number of columns of A down to $\tilde{O}(n^{\phi'_{d-1}})$ (using Lemma 6.11), and hash each remaining column down to a set of $\tilde{O}(n^{\phi'_d})$ elements. The remaining columns are the new immediate subtrees of N, and the remaining elements in each column are now the immediate subtrees of that column. ◀

[4] Recall that $(\pi^j)_{l_j}$ is a π^j-colored vertex in X.

For $e \in E(G)$ we can construct $T(e)$ in a similar manner, as explained in the full paper.

▶ **Lemma 6.15.** *For all $H, H' \subseteq G$ there exists a random, constant-depth circuit, independent of X, with $\tilde{O}(n^{\max(\Delta(H),\Delta(H'))+2})$ wires, that computes $T(H \cup H', \hat{\pi})$ from $T(H, \pi)$ and $T(H', \pi')$ w.h.p. for some $\hat{\pi}$.*

Proof Sketch. Let $T = T(H, \pi)$ and $T' = T(H', \pi')$. By Lemma 6.14 we can assume without loss of generality that $\{\pi^1, \ldots, \pi^{v(H \cap H')}\} = V(H \cap H') = V(H) \cap V(H')$, and that $\pi^k = \pi'^k = \hat{\pi}^k$ for $k \in [v(H \cap H')]$. Define ϕ' and $\hat{\phi}$ with respect to (H', π') and $(H \cup H', \hat{\pi})$ respectively.

Let $\psi_i = \min(\phi_i, \phi'_i)$. For $0 \leq d \leq v(H \cap H')$ let S_d be a depth-d tree in which each node at depth $i < d$ (including $i = 0$) has $\tilde{O}(n^{\psi_i})$ children. Each node of S_d has a (partial) label defined the same way as in T, such that no two nodes share a non-null label, and $\{\pi^1_{l_1}, \ldots, \pi^i_{l_i}\}$ extends to both H and H' (but not necessarily to $H \cup H'$) if and only if some node is labeled with l. Each leaf of S_d with a non-null label l is associated with the pair (τ, τ') of subtrees of T and T' respectively whose labels are also l.

The tree S_0 is the single node (T, T'), and we can compute S_{d+1} from S_d as explained in the full paper. Let $S = S_{v(H \cap H')}$. For d from $v(H \cap H') - 1$ down to 0, for each depth-d node N in S, hash (Lemma 6.11) the number of children of N down from $\tilde{O}(n^{\psi_d})$ to $\tilde{O}(n^{\hat{\phi}_d})$, and if all of N's children are null and $d > 0$ then remove N's partial label. Finally, for each leaf (τ, τ') of S, append a copy of τ' to each leaf of τ, and put this in place of (τ, τ') in S. ◀

For each successive H in an optimal union sequence, compute $T(H)$ as described above, and then apply a single OR gate to all leaves of $T(G)$. ◀

References

1. Noga Alon and Dániel Marx. Sparse balanced partitions and the complexity of subgraph problems. *SIAM J. Discrete Math.*, 25(2):631–644, 2011. doi:10.1137/100812653.
2. Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.
3. Kazuyuki Amano. k-subgraph isomorphism on AC^0 circuits. *Comput. Complexity*, 19(2):183–210, 2010. doi:10.1007/s00037-010-0288-y.
4. L. Sunil Chandran and Telikepalli Kavitha. The treewidth and pathwidth of hypercubes. *Discrete Math.*, 306(3):359–365, 2006. doi:10.1016/j.disc.2005.12.011.
5. Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoret. Comput. Sci.*, 326(1-3):57–67, 2004. doi:10.1016/j.tcs.2004.05.009.
6. L. H. Harper. On an isoperimetric problem for Hamming graphs. *Discrete Appl. Math.*, 95(1-3):285–309, 1999. doi:10.1016/S0166-218X(99)00082-7.
7. Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proc. 18th Ann. ACM Symp. on Theory of Computing*, pages 6–20, 1986. doi:10.1145/12130.12132.
8. Johan Håstad, Ingo Wegener, Norbert Wurm, and Sang-Zin Yi. Optimal depth, very small size circuits for symmetric functions in AC^0. *Inform. and Comput.*, 108(2):200–211, 1994. doi:10.1006/inco.1994.1008.
9. Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
10. Yuan Li, Alexander Razborov, and Benjamin Rossman. On the AC^0 complexity of subgraph isomorphism. *SIAM J. Comput.*, 46(3):936–971, 2017. doi:10.1137/14099721X.
11. Dániel Marx. Can you beat treewidth? *Theory Comput.*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.

12 Dániel Marx and Michał Pilipczuk. Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). In *STACS*, volume 25 of *LIPIcs*, pages 542–553. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.542.

13 K. Nakagawa and O. Watanabe. Gap Between Two Combinatorial Measures for Constant Depth Circuit Complexity of Subgraph Isomorphism. Technical report, Tokyo Institute of Technology, 2011.

14 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Comment. Math. Univ. Carolin.*, 26(2):415–419, 1985.

15 Neil Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

16 Benjamin Rossman. On the constant-depth complexity of k-clique. In *Proc. 40th Ann. ACM Symp. on Theory of Computing (STOC)*, pages 721–730, 2008. doi:10.1145/1374376.1374480.

17 Benjamin Rossman. *Average-Case Complexity of Detecting Cliques*. Ph.d., MIT, 2010.

18 Benjamin Rossman. The monotone complexity of k-clique on random graphs. *SIAM J. Comput.*, 43(1):256–279, 2014. doi:10.1137/110839059.

19 Benjamin Rossman. Lower bounds for subgraph isomorphism. In *Proc. Intern. Congress of Mathematicians (ICM)*, volume 3, pages 3409–3430, 2018. URL: https://eta.impa.br/dl/051.pdf.

20 Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill Book Company, 6 edition, 2011.

21 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. 18th Ann. IEEE Symp. on Foundations of Computer Science*, pages 222–227, 1977. doi:10.1109/SFCS.1977.24.

The PACE 2019 Parameterized Algorithms and Computational Experiments Challenge: The Fourth Iteration

M. Ayaz Dzulfikar[1]
University of Indonesia, Kota Depok, Jawa Barat 16424, Indonesia
muhammad.ayaz@ui.ac.id

Johannes K. Fichte
Faculty of Computer Science, TU Dresden, 01062 Dresden, Germany
johannes.fichte@tu-dresden.de

Markus Hecher
Institute of Logic and Computation, TU Wien, Favoritenstraße 9-11, 1040 Wien, Austria
University of Potsdam, Germany
hecher@dbai.tuwien.ac.at

Abstract

The organizers of the 4th Parameterized Algorithms and Computational Experiments challenge (PACE 2019) report on the 4th iteration of the PACE challenge. This year, the first track featured the MinVertexCover problem, which asks given an undirected graph $G = (V, E)$ to output a set $S \subseteq V$ of vertices such that for every edge $vw \in E$ at least one endpoint belongs to S. The exact decision version of this problem is one of the most discussed problem if not even the prototypical problem in parameterized complexity theory. Another two tracks were dedicated to computing the hypertree width of a given hypergraph, which is a certain generalization of tree decompositions to hypergraphs that has widely been applied to problems in databases, constraint programming, and artificial intelligence. On one track we asked for submissions that compute hypertree decompositions of minimum width (MinHypertreeWidth) and on the other track we asked to heuristically compute hypertree decompositions of small width quickly (HeurHypertreeWidth). We received 28 implementations from 26 teams. This year we asked participants to submit solver descriptions in order to count as a submission for the challenge. We received those from 16 teams with overall 33 participants from 10 countries. One team submitted successful solutions to all three tracks.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Complexity theory and logic; Mathematics of computing → Solvers; Mathematics of computing → Graph algorithms; Mathematics of computing → Hypergraphs

Keywords and phrases Parameterized Algorithms, Vertex Cover Problem, Hypertree Decompositions, Implementation Challenge, FPT

Digital Object Identifier 10.4230/LIPIcs.IPEC.2019.25

Category Invited Paper

Funding The work has been supported by the Austrian Science Fund (FWF), Grants Y698 and P26696, and the German Science Fund (DFG), Grant HO 1294/11-1 and Erasmus+ KA107. Hecher is also affiliated with the University of Potsdam, Germany.

Acknowledgements The PACE challenge was supported by *Networks* [5], an NWO Gravitation project of the University of Amsterdam, Eindhoven University of Technology, Leiden University and the Center for Mathematics and Computer Science (CWI), by the *Centre for Information and High Performance Computing (ZIH) of TU Dresden* [3], and by *data-experts* [1]. The prize money (4,000

[1] The author contributed significantly to the realization of PACE 2019 during his internship at TU Dresden.

EUR) was given through the generosity of Networks and data experts. We are grateful to Szymon Wasik and Jan Badura for the fruitful collaboration and for hosting the challenge at optil.io [154]. We like to acknowledge the generous support by the High Performance Computing Center at TU Dresden, who gave us access to the HRSK-II and 80.000 CPU hours from February on to select instances and validate the results [3].

1 Introduction

The Parameterized Algorithms and Computational Experiments Challenge (PACE) was conceived in Fall 2015 to deepen the relationship between parameterized algorithms and practice. It aims to:

1. Bridge the divide between the theory of algorithm design and analysis, and the practice of algorithm engineering.
2. Inspire new theoretical developments.
3. Investigate in how far theoretical algorithms from parameterized complexity and related fields are competitive in practice.
4. Produce universally accessible libraries of implementations and repositories of benchmark instances.
5. Encourage the dissemination of these findings in scientific papers.

The first iteration of PACE was held at IPEC 2016 [40]. There programmers were asked for submissions on two tracks, namely, (a) a treewidth track allowing for exact sequential, exact parallel, heuristic sequential, and heuristic parallel submissions and (b) a feedback vertex set track. PACE 2017 [41] featured (a) a treewidth track allowing for sequential exact or heuristic submissions and (b) a minimum fill-in track. PACE 2018 [26] then asked for submissions that solve the Steiner tree problem. The line of past challenges has inspired a long list of works on the proposed problems [8, 17, 20, 57, 62, 76, 81, 90, 97, 98, 107, 123, 142, 153, 143, 144, 151, 152, 99]. Benchmarks from the PACE challenges have been used for other competitions and evaluations [47, 129, 134]. Various applications have built on top of results from PACE or were inspired by the success of solvers produced for PACE [16, 21, 22, 30, 32, 33, 58, 59, 60, 61, 64, 65, 66, 101, 103, 105, 110, 109, 111, 122, 127, 150, 158]. Finally, PACE challenges have been mentioned in research works [38, 115]. Among the various papers have also been papers that received a best paper award [16, 143].

In this article, we report on the 4th iteration of PACE. The PACE 2019 challenge was announced on November 16, 2018. Format descriptions were posted on November 20, 2018 and the public instances were released on December 5, 2019 (hypertree decompositions) January 4, 2019 (vertex cover) and updated on March 6, 2019 (vertex cover), since the initial instances were not challenging enough for the participants. The final version of the submissions was due on May 6, 2019. We informed the participants of the results on July 14. We released the private instances on July 29 [50, 56] and announced the final results to the public on September 11, during the award ceremony at the International Symposium on Parameterized and Exact Computation (IPEC 2019) in Munich. For the first time in the history of PACE, we had a poster session after the award ceremony, where 4 posters were presented, namely WeGotYouCovered [92], bogdan [156], asc [139] as well as TULongo [124, 125].

PACE 2019 consists of three tracks. The first track featured the MinVertexCover problem, which asks given an undirected graph $G = (V, E)$ to output a set $S \subseteq V$ of vertices such that for every edge $vw \in E$ at least one endpoint belongs to S. The exact decision version of this problem is one of the most discussed problem if not even the prototypical problem in

parameterized complexity theory. Another two tracks were dedicated to compute the hypertree width of a given hypergraph, which is a certain generalization of tree decompositions to hypergraphs that has widely been applied to problems in databases [82], constraint programming [35, 82, 87], and artificial intelligence [106, 108]. On one track we asked for submissions that compute hypertree decompositions of minimum width (MinHypertreeWidth) and on the other track we asked to heuristically compute hypertree decompositions of small width fast (HeurHypertreeWidth). This year's PACE had quite relaxed solver requirements and we even allowed solvers to use external dependencies such as ILP, SAT, and SMT solvers if the external solvers were available under an open source license. We received 28 implementations from 26 teams. This year we asked participants to hand in solver descriptions and received those from 16 teams. If we count only submissions that handed in a solver and a description according to the submission requirements, we had overall 33 participants from 10 countries. One outstanding team submitted successful solutions to all three tracks.

2 The PACE 2019 Challenge Problems

In this section, we give an overview on the PACE 2019 problems. We organized the section by problem and present the well-known vertex cover problem first and then a generalization of treewidth to hypergraphs. For each problem, we start with a quick definition, introduce the tracks and selected instances. We finish with the submission requirements. In the following, we assume that the reader is familiar with basic graph terminology and we refer to standard texts [25, 45] otherwise.

2.1 Vertex Cover (Track 1a)

Computing minimum vertex covers was among the original 21 NP-complete problems by Karp [104] and is probably one of the most famous graph problems. In fact, there are over 537,000 results (queried on 30.07.2019) on Google scholar, dealing with problems related to finding vertex covers and variants thereof. Besides, vertex covers are particularly well-studied in parameterized complexity [37], ranging from studies involving different parameters [31, 126] and related problems [121, 137], over kernelization [52, 112], and also concern concrete applications [28, 46, 51, 96, 131]. We use the following definition.

▶ **Definition 1** (Vertex Cover). *Given an undirected graph $G = (V, E)$. A set $S \subseteq V$ is a* vertex cover *for G, if for every edge $uv \in E$, we have $\{u, v\} \cap S \neq \emptyset$. A vertex cover S is a minimum vertex cover for G if there is no vertex cover S' for G such that $|S'| < |S|$.*

This definition motivates the problem of Track 1a.

Problem:	MinVertexCover (Exact)
Input:	Undirected graph G
Task:	Output a minimum vertex cover for G.

Data Format

The input format for providing a graph (*.gr*) was taken from the PACE 2017 format for graphs [41]. The output format for specifying a vertex cover (*.vc*) was an adaption of the input format in the same style. More details on the format can be found at `pacechallenge.org/2019/vc/vc_format`. There is also a simple checker available at `github.com/hmarkus/vc_validate` in Python (folder: `vc_validate`) and C++ (folder: `cpp`).

Table 1 Information on the origins of our graphs, including download links and relevant converters.

	Name	#	Reference (Download Link)	Converter
1	PACE 2016/Treewidth	17	pacechallenge.org:2016	
2	TransitGraphs	23	github:daajoe/transit_graphs	[53]
3	Road-graphs	5	github.com:ben-strasser	
4	SNAP	15	snap.stanford.edu	
5	frb	41	buaa.edu.cn:kexu/benchmarks	
6	ASP Horn backdoors	1,077	asparagus.cs.uni-potsdam.de	[23, 24, 78, 102, 54]
7	SAT Horn backdoors	83	marco.gario.org	[74, 55]
			tinyurl.com:countingbenchmarks	
8	SAT2VC	8,329	satlib.org	[49]
			tinyurl.com:countingbenchmarks	
			sat2018.tuwien:benchmarks	

Instances for Vertex Cover

In order to establish a suitable set of benchmark instances from various areas, we considered in total 9,591 instances comprising of the following 8 origins.

1. 17 graphs from `PACE 2016/Treewidth` [40];
2. 23 graphs from `TransitGraphs` (Denmark, FlixBus, Israel, Luxembourg, Metro Bilbao, Mexico City, NYC Subway, Pace Bus, Praha, Translink, VBB, WienerLinien) [53];
3. 5 graphs from `Road-graphs` [42, 141];
4. 15 graphs from `SNAP` (Stanford Network Analysis Project) [117] including gnutella [120, 135], social circles from facebook [130], bitcoin OTC trust network [113, 114], and wiki vote [118, 119],
5. 41 graphs from `frb` [155];
6. 1,077 graphs from `ASP Horn backdoors` [63], where a vertex cover of the graph gives a Horn backdoor to the considered logic program (ASP) [27, 100], originating from various ASP competitions [10, 29, 43, 79], in more detail, 7 *SCoreDLP-Mutex* [128] logic programs, 282 *SCore-RLP200* logic programs [157], 200 *MinimalDiagnosis* [29, 77] logic programs, and 588 *automotive* [140] logic programs;
7. 83 graphs from `SAT Horn backdoors` [138], where a vertex cover of the graph gives a Horn backdoor to the considered SAT instance, originating from 15 *dfremont projection* formulas [71] and 68 *Gario* formula collection [73, 75];
8. 8,329 graphs from `SAT2VC`, where we took widely used SAT instances, reduced them into 3-SAT instances [133] and then reduced them to k-vertex cover by means of the well-known reductions by Karp [104] (merge the reductions from SAT to clique and then clique to vertex cover into one). Then, a vertex cover of $k = n + 2m$ exists if and only if the given formula of n variables and m clauses is satisfiable. We took 6,956 SAT instances from the *SATlib* collection [95], 1,187 instances from a popular *#SAT* collection [66, 71], and 186 instances from the *SAT 2018 competition* [94].

We implemented a variety of converters, among those tools where the following: asp_horn_backdoors [54], HornVCBuilder [74], lp2normal [23, 24], gringo [78, 102], a SAT to vertex cover converter [49]. Table 1 gives an overview on the sources and the involved graph converters or programs that implement the reductions described above.

For PACE 2019, we were interested in challenging, large instances that are still within reach for the participants in reasonable development time, but require reasonable efforts to score one of the medals. In order to get a naive classification of the "practical hardness"

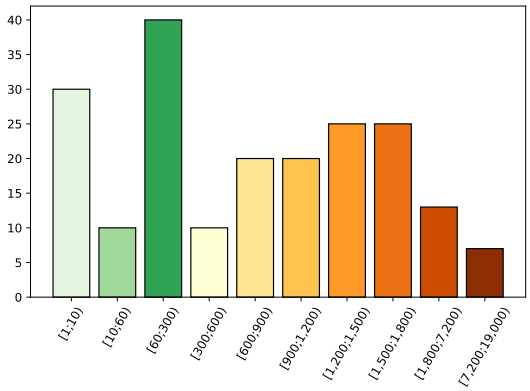


Figure 1 Overview on the number of instances and runtime intervals, which were computed by means of a simple ILP encoding solved by the (M)ILP solver Gurobi, for the selected instances of Track 1a. The x-axis labels the considered intervals, i.e., $[a; b)$ indicates that runtime t was within the interval $a \leq t < b$. The y-axis indicates the number of selected instances.

of our collected benchmark instances, we encoded the MinVertexCover problem into a simple ILP encoding and ran the solver Gurobi [89] on all instances with a timeout of 6 hours. After obtaining initial runtime results, we assigned to each instance a category (easy$\{1, 2, 3\}$, medium$\{1, 2, 3, 4\}$, and hard$\{1, 2, 3\}$). From the classified instances, we picked 200 instances by sampling uniformly at random among the distribution given in Figure 1. Mainly, we dropped an instance if the runtime was below 1s, and picked 80 instances from category easy (runtime within the interval $[1; 300)$), picked 80 instances from category medium (runtime within the interval [300;1,500), and 40 instances from category hard (runtime in the interval [1,500;19,000)). We dropped instances that could not be solved within 6 hours. We numbered the instances from 1 to 200 with increasing hardness, selected the odd numbered instances as public and even numbered instances as private instances. Table 2 shows statistics on the resulting instances. The 100 public instances were released at `pacechallenge.org/2019/vc/vc_exact`. All the selected instances are publicly available at `Zenodo:3368306` [50]. We released the full collection of instances, instance selection scripts, and mapping of selected instances at `github:daajoe/pace_2019_vc_instances`.

2.2 Hypertree Decompositions (Tracks 2a and 2b)

Hypertree decompositions and the resulting measure hypertree width is a prominent structural restriction in the area of constraint satisfaction problems (CSP) [11, 39, 149] and databases [82], such as the commercial database system LogicBlox [12, 15, 6, 7, 132] that uses hypertree decompositions. In the beginning of the 80s, Freuder [72] showed that the CSP is tractable under structural restrictions imposed in terms of bounded treewidth of the constraint graph. The result has later been generalized by Gottlob, Leone, and Scarcello to hypertree width [82], which still renders CSP polynomial-time tractable. In fact, the polynomial-time solvability for bounded hypertree width instances still holds when one is interested in counting the number of satisfying assignments [48], which is also known as sum-of-products, weighted counting, partition function, or probability of evidence [106]. Thus, this problem is also of high interest in artificial intelligence, e.g., to solve probabilistic reasoning [108]. While there are even more general measures [86, 87], hypertree decompositions allow for computing a hypertree decomposition of width at most k (if one exists) in polynomial time for a given fixed integer k. Still hypertree width was mainly of theoretical

Table 2 Basic statistics on the selected PACE 2019 instances for Track 1a (Vertex Cover/Exact). $|V_{\min}|$ and $|E_{\min}|$ refers to the minimum number of vertices and edges, respectively; max refers to the maximum; avg refers to the mean; med refers to the median; $\mathrm{tw}^{\mathrm{ub}}_{\mathrm{med}}$ refers to a heuristically computed upper bound (median over the instances) on the treewidth using the library htd [8].

instances	$\lvert V_{\min}\rvert$	$\lvert V_{\max}\rvert$	$\lvert V_{\mathrm{avg}}\rvert$	$\lvert V_{\mathrm{med}}\rvert$	$\lvert E_{\min}\rvert$	$\lvert E_{\max}\rvert$	$\lvert E_{\mathrm{avg}}\rvert$	$\lvert E_{\mathrm{med}}\rvert$	$\mathrm{tw}^{\mathrm{ub}}_{\mathrm{med}}$
public	198	138.14k	16.44k	14.69k	813	227.24k	30.95k	24.66k	105.0
private	153	98.13k	16.30k	13.59k	625	161.36k	30.50k	27.15k	103.5
all	153	138.14k	16.37k	13.59k	625	227.24k	30.73k	24.66k	107.0

interest due to few practical implementations and missing efficient implementations of heuristics to compute the associated decompositions. The success of PACE 2016 and 2017 and its resulting decomposers for computing tree decompositions, which facilitated lots of follow-up implementations [32, 33, 66, 61], and the hope that PACE also drives advances to the CSP, reasoning, and database community, motivated us to propose this problem for Track 2.

▶ **Definition 2** (Hypergraphs and Tree Decompositions). *A* hypergraph *is a pair $H = (V, E)$ consisting of a set V of* vertices *and a set E of* hyperedges*, where each hyperedge in E is a subset of V. Let $H = (V, E)$ be a hypergraph. A* tree decomposition *[136] of H is a pair $\mathcal{T} = (T, \chi)$ where $T = (N, A)$ is a rooted tree and χ is a mapping that assigns to each node $t \in N$ a set $\chi(t) \subseteq V$, called* bag*, such that the following conditions hold: (i) $V = \bigcup_{t \in N} \chi(t)$, (ii) $E \subseteq \{2^{\chi(t)} \mid t \in N\}$, and (iii) for each $r, s, t \in A$ where s lies on the path from r to t, we have $\chi(s) \subseteq \chi(r) \cap \chi(t)$.*

We follow the definitions of Gottlob, Leone, and Scarcello [82].

▶ **Definition 3** (Hypertree Decompositions [82]). *Let $H = (V, E)$ be a hypergraph and let $S \subseteq V$ be a set of vertices. An* edge cover *$C \subseteq E$ of S is a set of hyperedges, where for every $v \in S$, there is $e \in C$ with $v \in e$. A* hypertree decomposition *of H is a triple $\mathcal{H} = (T, \chi, \lambda)$, where (i) (T, χ) is a tree decomposition of H with $T = (N, A)$, (ii) λ is a mapping that assigns to each node $t \in N$ an edge cover $\lambda(t)$ of $\chi(t)$, and (iii) for every $t \in N$ and every $e \in \lambda(t)$, we have $e \cap \chi_{\leq t} \subseteq \chi(t)$. The set $\chi_{\leq t}$ refers to the set of all vertices occurring in a bag $\chi(t')$ of the subtree $T' = (N', A')$ of T where T' is rooted at t and $t' \in N'$. Then,* width$(\mathcal{H})$ *is the size of the largest edge cover $\lambda(t)$ over all nodes $t \in N$. The* hypertree width $\mathrm{htw}(H)$ *is the smallest width over all hypertree decompositions of H.*

Based on this generalization of tree decompositions to hypergraphs, we defined the following two problems for Track 2a and 2b.

Problem:	MinHypertreeWidth (Exact)
Input:	Hypergraph H
Task:	Output a hypertree decomposition of H of minimum width.

Problem:	HeurHypertreeWidth (Heuristic)
Input:	Hypergraph H
Task:	Output a hypertree decomposition of H of small width.

Table 3 Basic statistics on the selected PACE 2019 instances for Track 2a (MinHypertreeWidth) and Track 2b (HeurHypertreeWidth). $|V_{\min}|$ and $|E_{\min}|$ refers to the minimum number of vertices and hyperedges, respectively; max refers to the maximum; med refers to the median.

Track	instances	$\|V_{\min}\|$	$\|V_{\max}\|$	$\|V_{\text{med}}\|$	$\|E_{\min}\|$	$\|E_{\max}\|$	$\|E_{\text{med}}\|$
Track 2a (Exact)	public	3	130	24.0	3	100	61.5
Track 2a (Exact)	private	10	351	25.0	5	250	60.0
Track 2a (Exact)	all	3	351	24.0	3	250	60.0
Track 2b (Heuristic)	public	12	694	40.0	5	526	84.0
Track 2b (Heuristic)	private	12	694	40.0	5	495	90.0
Track 2b (Heuristic)	all	12	694	40.0	5	526	84.0

Computation of Hypertree Decompositions

Above we mentioned that there are a variety of applications for hypertree decompositions. However, many practical sides are not very well explored. In fact, for tree decompositions both exact as well as heuristic-based decomposers are widely available due to recent PACE challenges, this is not the case for hypertree decompositions. There, only very few implementations are available and the exact implementations are highly prototypical. Fortunately, various theoretical results on computing hypertree decompositions [82, 83, 70, 69, 68] and more general measures [57] are available. Some of these approaches are simply combinatorial backtracking based algorithms, others are heuristics based on bucket elimination, and again others are based on encodings into extensions of SAT. A major obstacle for hypertree decompositions is that Condition (iii) of Definition 3 is expensive and in case of encodings into SAT-related formalisms it blows up the size computation considerably.

Data Format

We designed the input and output format by extending the PACE 2016 formats used for graphs and tree decompositions [40, 41], which are similar to the format used by DIMACS challenges [4]. The input format for hypergraphs (*.hgr*) extends the PACE 2016/2017 graph format to edges of arbitrary arity. The output format for hypertree decompositions (*.htd*) allows in addition to the treewidth format to specify a covering function, i.e., mappings for the bags that map hyperedges to 0 or 1. More details on the format can be found at `https://pacechallenge.org/2019/htd/htd_format/`. We provided a simple checker at `https://github.com/daajoe/htd_validate` in Python (folder: `htd_validate`) and C++ (folder: `cpp`). Both tools already implement reading and outputting the formats.

Instances for Hypertree Decompositions

For our benchmark selection, we considered 2,191 instances, which contain hypergraphs that originate from CSP instances and conjunctive database queries from various sources. All of these instances are part of the hyperbench collection and have been collected and published by Fischl et al. [68, 69] together with different hypergraph properties including various notions of width related measures[2]. The selection consists of eight non-disjoint sets.

[2] The hypergraphs together with the properties have been published at `http://hyperbench.dbai.tuwien.ac.at` and the collection of hypergraphs is also available in a public data repository [57].

15 instances from the set `DaimlerChrysler`, 12 instances from the set `Grid2D`, 24 instances on circuits from the set `ISCAS'89` [85]. 31 instances from `MaxSAT` [19]. 1090 instances and 863 instances, respectively on `csp_application` and `csp_random` of instances from the well known XCSP benchmarks [14]. 82 instances from the set `csp_other`, which have been collected for works on hypertree decompositions[3]. 156 instances from the set `CQ` on various conjunctive queries [13, 18, 80, 88, 116, 145].

In order to obtain a basic classification of the instances we heuristically computed hypertree decompositions with htdecomp [44] and computed generalized hypertree decompositions of smallest width [57]. Generalized hypertree decompositions relax Condition (iii) in Definition 3 and allow certain techniques to find a solution faster. The widths of the thereby obtained decompositions are indeed of minimum width, which is guaranteed by comparing the widths with the matching integer lower bounds obtained by fraSMT [57] – a tool for computing fractional hypertree decompositions, which are more general than hypertree decompositions. After obtaining initial runtime results we assigned to each instance one category out of easy, medium, hard, or dropped the instance. An instance was classified as easy if it could be solved within 60s, as medium if it could be solved within 300 and 900s, and hard if it could not be solved within 7,200s. We dropped instances that could not be solved within 7,200s on purpose, since we were interested in many realistic instances and quite challenging instances for the solvers. From the remaining and classified instances we picked 200 by sampling 20 instances uniform at random from category easy, 60 instances from the category medium, and 120 instances from the category hard. Table 3 shows statistics on the resulting selected instances. We released the public instances at `https://pacechallenge.org/2019/htd/htd_exact/` and `https://pacechallenge.org/2019/htd/htd_heur/`, respectively. The pages also contain a document that contains the mapping of the selected PACE 2019 instances and the original instance of hyperbench. We also published the instances in a public data repository [56].

3 Challenge Settings

In the following, we state the submission requirements for this year's PACE and basic information on the system on which we ran the challenge.

Submission Requirements

We invited people to participate in the three proposed tracks. In order to have common setting we however posted the following *submission requirements*.

1. Both submitted solver and external dependencies have to be *open source*.
2. The source code of the solver is maintained by the submitters on a *public repository*.
3. A *dedicated solver description* of at least two pages has to be submitted.

We choose Requirement 1 fairly permissive, in order to obtain valuable information on the actual efficiency of solving the problem. So we did not prescribe the algorithmic paradigm that had to be used. In that way, we also allowed in principle submissions that relied on an encoding into paradigms such as SAT or SMT. We imposed Requirement 3 to enable other researchers to analyze and compare implemented ideas, get insights into correctness of the other solvers, provide theoreticians with basics ideas on the latest implementations and in the hope to improve on the reproducibility of the submitted solver.

[3] `www.dbai.tuwien.ac.at/proj/hypertree/benchmarks.zip`

Region	Country	Teams	Participants	Tracks
Europe	Austria	3	3	1a, 2a, 2b
	Germany	4	7	1a
	Hungary	1	2	1a
	Norway	2	6	1a
	Poland	2	4	1a
	Russia	1	1	1a
	Scotland	1	2	1a, 2a, 2b
Middle East	Lebanon	1	2	1a
North America	USA	1	1	1a
South Asia	India	2	5	1a
	10	18	33	

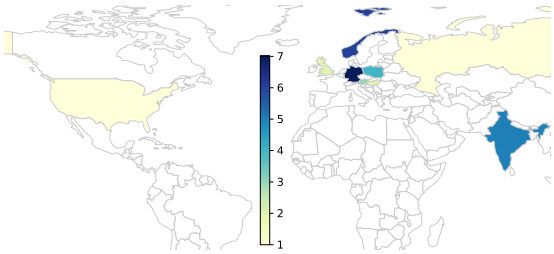


Figure 2 Participation per country based on the easychair registration and submission of both a solver and a description. Details are given by country, tracks, and team (left) and illustrated on the world map (right). Note that more teams and participants uploaded their code on optil.io

In addition, we imposed another main rule for the exact tracks.

E. We expected submissions to be based on a provably optimal algorithm.

While we did not formally check Requirement E, we picked only instances from which we knew the size of a minimum vertex cover or the hypertree width, respectively and checked whether the output was both correct and according to our expected size. If a submission halted on some instance within the allotted time, but produced a solution that was known to be non-optimal, the submission was disqualified.

Limits

Since our evaluation resources were limited and we were interested in the solving behavior on a larger number of instance while allowing the participants to have a "training" phase on public instances, we restricted the runtime to 1,800 seconds and the available main memory to 8GB per instance. Note that in general, a solver is considered to be better than an other solver, if it solves more instances faster than the other solver. For more details about evaluation (criteria), we refer to Section 4.3.

Hardware

Our results were gathered on the cloud evaluation platform optil.io [154] running libc 5.4.0. optil.io evaluates submissions on Intel Xeon CPU E5-2695 v3, which consist of 14 cores running at 2.30GHz. Each submission had access to one core. Since the submissions ran in docker containers and the CPU has only 4 memory channels [2] we repeated the final evaluation 3 times and took the average.

4 Participants and Results

This year, we had 18 teams and 33 participants coming from 10 countries and four regions: Austria, Germany, Hungary, India, Lebanon, Norway, Poland, Russia, Scotland, and USA. Figure 2 provides an overview. The number of teams and participants were little less than half compared to PACE 2018 and at a similar number to PACE 2017. To be precise, the 2019 numbers above correspond to teams and participants who sent a final implementation and a solver description in time whereas in previous challenges registrations were carried out prior to optil.io registration. If we also count people who uploaded some code on the optil.io platform but dropped out of the challenge, the number of teams is 28, which is however also a much smaller number than in the 2018 iteration.

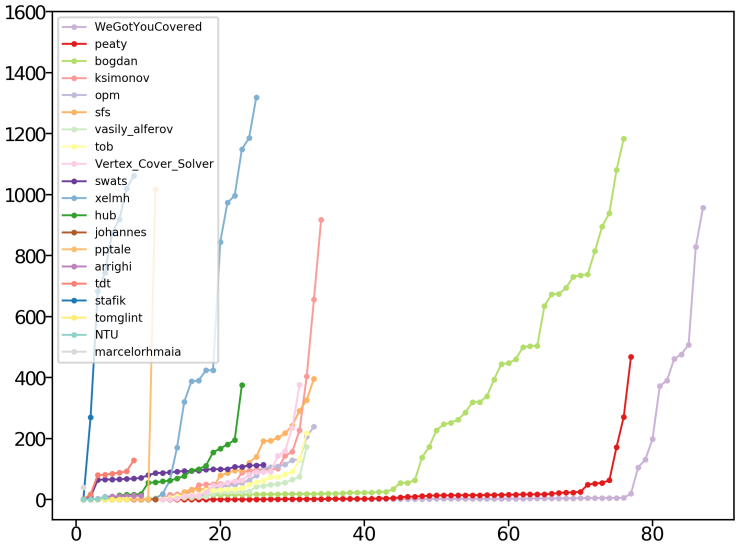


Figure 3 Runtime illustrated as cactus plot for `Track 1a (Vertex Cover/Exact)`. The x-axis labels consecutive integers that identify instances. The y-axis depicts the runtime. The instances are ordered by running time, individually for each solver.

4.1 Track 1a (Vertex Cover/Exact)

Figure 3 illustrates runtime results for all submitted solvers as cactus plot. Table 4 gives a detailed overview on the standings and solvers. We allowed each solver 30 minutes per instance and measured the number of solved instances. If two solvers solved the same number of instances we would in addition also take into account the runtime needed to solve those instances.

Winning Team. Demian Hespe and Sebastian Lamm (both: Karlsruhe University of Technology, Germany), Christian Schulz (University of Vienna, Austria), and Darren Strash (Hamilton College, USA) won Track 1a by solving 87 private instances in overall 1.3 hours and 52.7 seconds solving time on average. Their implementation (WeGotYouCovered) [92, 93] builds on a portfolio of techniques, which include an aggressive kernelization strategy with all known reduction rules, local search, branch-and-reduce, and a state-of-the-art branch-and-bound solver. Surprisingly, they also use several techniques that were not from the literature on the vertex over problem, but originally published to solve the (complementary) maximum independent set and maximum clique problems.

Runner-up. Patrick Prosser and James Trimble from the University of Glasgow, Scotland, scored second with their solver Peaty by solving 77 private instances. In fact, the results looked much closer on the public instances [148]. Interestingly, they also used intensive kernelization, local search, improved branch-and-bound, and a branch-and-bound maximum clique solver, and in addition an exact graph colouring algorithm which can quickly prove the optimality of a solution for some graphs.

Third Place. Sándor Szabó (University of Pecs, Hungary) and Bogdán Zaválnij (Hungarian Academy of Sciences, Hungary) accomplished a safe third place, which was in fact very close to the team that obtained the second place on the private instances. However, on the public

Table 4 Detailed standings of the submitted solvers that solved at least 15 instances for `Track 1a (Vertex Cover/Exact)`. POS refers to the position of the solver where DSQ refers to a disqualification due to a produced wrong answer. # refers to the number of solved private instances and $\#^{\text{all}}$ refers to the number of all instances. TLE refers to the number of instances were the runtime limit was exceeded. RTE contains the number of instances on which we observed a runtime error. Note that if the three sums do not add to 100 we observed a memory overflow on the remaining ones. $t_{\text{sum}}[h]$ states the cumulative runtime over all all solved instances in hours, $t_{\text{avg}}[s]$ contains the average runtime over all solved instances in seconds. In column source, the character H abbreviates github, character L abbreviates gitlab, and Z refers to Zenodo.

POS	Solver	#	$\#^{\text{all}}$	TLE	RTE	$t_{\text{sum}}[h]$	$t_{\text{avg}}[s]$	Source	Reference
1	WeGotYouCovered	87	169	13	0	1.3	52.7	`H:sebalamm/pace-2019`	[92, 93]
2	Peaty	77	157	23	0	0.4	20.6	`H:jamestrimble/peaty`	[148]
3	bogdan	76	147	24	0	4.5	215.2	`H:zbogdan/pace-2019`	[156]
4	ksimonov	34	73	64	2	1.0	102.1	`L:seemann9/pace-2019-vc`	[36]
5	opm	33	75	67	0	0.4	47.8	`Z:3236867#.XW_J9S2B3x8`	[67]
6	sfs	33	74	65	2	0.8	87.0	`L:cg_pace2019/vertex_cover`	[34]
7	vasily_alferov	32	70	68	0	0.2	23.2	`H:vasalf/cheburashka`	[9]
8	hub	23	54	68	9	0.5	79.5	`H:hubhegnel/pace-2019`	[91]
DSQ	Vertex_Cover_Solver	31	69	68	0	0.4	52.1	`H:karamkontar99/Vertex-Cover-Solver`	

instances they solved 9 less than the authors of Peaty. The authors used kernelization and a maximum clique solver by taking the complement graph. The maximum clique solver is based on progressive k-clique search by starting from a heuristically computed maximum clique and increasing until no clique of size k is found.

4.2 Track 2a (Hypertree Width/Exact)

Table 5a summarizes runtime results for all submitted solvers. We allowed each solver 30 minutes per instance and measured the number of solved instances. Much to our regret we received only 3 submissions. We guess that hypertree width is just yet not very popular in the parameterized complexity community.

Winning Team. André Schidler and Stefan Szeider from TU Wien, Austria, won this year's Track 2a by solving 69 private instances in overall 1.4 hours at an average of 69.4 seconds when considering the solved instances. Their implementation (asc) [139] uses an incremental SMT-solving approach. There a first-order logic solver (handling arithmetic constraints) interacts with a SAT solver. Hypertree width and a more general parameter (generalized hypertree width) share Conditions (i) and (ii) from Definition 3. The additional Condition (iii) which is present for hypertree width (special condition), however, blows up the encoding size with a cubic number of clauses resulting in extremely large encodings for generalized hypertree decompositions and very long encoding times. For that reason, the authors implement a two-phase approach. They first use an encoding to obtain a generalized hypertree decomposition and try convert it into a hypertree decomposition satisfying the special condition without increasing the width. Only if that fails, they use the full encoding that includes the special condition.

Runner-up. Davide Mario Longo from TU Wien, Austria, scored second with his solver (TULongo/HdSolveE) by solving 31 private instances in 0.8 hours at an average runtime of 95.9 seconds over the solved instances. He used a combination of algorithms to compute lower bounds by obtaining generalized hypertree decompositions and then running a backtracking-based algorithm to determine a hypertree decomposition. Surprisingly, he solved much less public than private instances.

Table 5 Detailed overview on the results of the Hypertree Width tracks. # refers to the number of solved private instances. $\#^{all}$ comprises the number of solved public and private instances. t_{avg} and t_{sum} refer to average and cumulated runtime of solved private instances, respectively. PAR1 refers to the runtime where all unsolved instances are accounted by 1,800 seonds. Δ_w refers to the sum of the width difference to the resulted output by the judge, i.e., the sum over $w_{\text{solver}} - w_{\text{judge}}$ for each instance I.

(a) `Track 2a (Hypertree Width/Exact)`.

POS	#	$\#^{all}$	Solver	t_{avg}	t_{sum}	Source (github)	Ref.
1	69	144	asc	69.38s	1.32h	`ASchidler/frasmt_pace`	[139]
2	31	48	TULongo	95.96s	0.83h	`TULongo/pace-2019-HD-exact`	[124]
3	1	6	heidi	0.14s	0.00h	`jamestrimble/heidi`	[146]

(b) `Track 2b (Hypertree Width/Heuristic)`.

POS	Score	Solver	#	PAR1	Δ_w	Source (github)	Ref.
–	na	htdecomp	100	na	na		
1	5.0	hypebeast	100	0.1h	501	`jamestrimble/hypebeast`	[147]
2	14.1	TULongo	98	2.3h	20	`TULongo/pace-2019-HD-Heuristic`	[125]
3	128.9	asc	30	27.5	11	`ASchidler/frasmt_pace`	[139]

Third Place. Patrick Prosser and James Trimble from the University of Glasgow, Scotland, received a surprising third place by solving one private instance correctly. They made the most of the situation that we had only three submission on this track. Their solver (heidi) implements an incomplete approach, where they heuristically compute a hypertree decomposition and used simple rules to check whether the width is equal to two.

4.3 Track 2b (Hypertree Width/Heuristic)

Table 5b summarizes runtime results for all submitted solvers. As for Track 2a, we received only 3 submissions, while we were hoping to attract more researchers from the community to this topic. We allowed each solver 30 minutes wall clock time per instance, while ensuring that not more than one core was used. Our aim for the track was to have more decomposers available to foster algorithms that employ decompositions for constraint programming and database applications in the near future. In the past, we observed at multiple occasions that heuristics are often only well applicable if there is a fairly good balance between running time of the computation of the decomposer and width of the decomposition [32, 33, 59, 61, 66]. Improving the width by one pays off if the improvement runs fast and the width is fairly large, however, on instances with small width there is no practical point in spending additional runtime to improve while it might even exceed the running time of the later algorithm that exploits the decomposition. In consequence, we decided to favor submissions that produce a result fairly quickly while still penalizing decompositions that are far from the virtual best results. Since the widths are fairly small, we decided for a very simple score (avoiding exponentially increasing penalties) and compute it per instance I by $(50,000 + t + 50 \cdot (w_{\text{solver}} - w_{\text{judge}}))/1,000,000$ where t refers to the wall clock and w_{solver} refers to the resulting width of the considered solver for I and w_{judge} consists of the width htdecomp [44, 84] produced for I, which we used as judge. The way we selected the score also depended on the situation that the runtime environment optil.io has certain technical restrictions in case of unknown optimal results.

Winning Team. Patrick Prosser and James Trimble from the University of Glasgow, Scotland, obtained the first place by obtaining a score of 5.0 by solving 100 instances in 0.1 hours with a cumulated width of 1104 and an overall difference to the judge by 501. Their solver (hypebeast) implements a very simple bucket elimination strategy by starting from a single tree node containing all hyperedges and then trying to move edges to deeper tree nodes.

Runner-up. Davide Mario Longo from TU Wien, Austria, scored second with 14.1 points by solving 98 private instances in 2.3 hours with a total width difference to the judge of 20. His solver (TULongo/HdSolveH) used a variant of det-k-decomp that prunes the search tree heuristically [85]. While the det-k-decomp algorithm performs well on yes instances, it is slow on no instances. So, Longo decided not to perform a complete search, but to prune the search space by looking only at certain separators. The results were very close to the winning team and only the running time cost him the first position. It is notable that, however, the width is much closer to the result by the judge. More precisely, TULongo outputted better results on 86 instances.

Third Place. André Schidler and Stefan Szeider from TU Wien, Austria, obtained the third position at a score of 128.9 by solving 30 instances using the same technique as above.

5 PACE organization

The composition of the steering committee and program committee during PACE 2019 was as follows.

Steering committee:	Édouard Bonnet	LIP, ENS de Lyon
	Holger Dell	IT University of Copenhagen
	Bart M. P. Jansen (chair)	Eindhoven University of Technology
	Thore Husfeldt	IT Univ. of Copenhagen & Lund Univ.
	Petteri Kaski	Aalto University
	Christian Komusiewicz	Philipps-Universität Marburg
	Frances A. Rosamond	University of Bergen
	Florian Sikora	LAMSADE, Université Paris Dauphine
Program Committee (Tracks 1a, 2a, 2b):	Johannes Fichte	TU Dresden
	Markus Hecher	TU Vienna & University of Potsdam

The Program Committee of **PACE 2020** will consist of Łukasz Kowalik (Univ. of Warsaw).

6 Conclusion and Future Editions of PACE

We thank all the participants for their enthusiasm, strong and interesting contributions. Special thanks go to the participants who also presented at IPEC 2019. We are very happy that this edition attracted many people and that also people from the SAT community showed interest in the latest standings on the vertex cover solvers. While we were hoping to attract more people to the hypertree width measure, which is related to constraint programming and to the database community. We are still happy about strong contributions and hope that this will continue for future editions by considering popular problems to the community or even by repeating previously posted problems.

This year we changed the requirements for submissions by allowing external libraries, but enforcing that these are open source. Further, we asked participants to provide a solver description and to place the source code on a public data library, which is hosted long-term

by a public body, e.g., Zenodo. In line with this, we provided the complete pool of instances from which we selected instances, the instance selection process including references to the original source, as well as the evaluation in a public data library.

We welcome anyone who is interested to add their name to the mailing list on the PACE website to receive updates and join the discussion. We look forward to the next edition. Detailed information will be posted on the website at `pacechallenge.org`.

References

1 data experts gmbh. `https://www.data-experts.de/`.

2 Intel® Xeon® processor E5-2695 v3. `https://ark.intel.com/content/www/us/en/ark/products/81057/intel-xeon-processor-e5-2695-v3-35m-cache-2-30-ghz.html`, 2014.

3 Centre for Information Services and High Performance Computing. `https://tu-dresden.de/zih/hochleistungsrechnen/`, 2019. Project: pacechallage2019.

4 DIMACS Challenge. `http://dimacs.rutgers.edu/programs/challenge/`, 2019.

5 Networks project. `https://www.thenetworkcenter.nl`, 2019.

6 Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. Joins via Geometric Resolutions: Worst-case and Beyond. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'15)*, pages 213–228, Melbourne, Victoria, Australia, 2015. Assoc. Comput. Mach., New York. `doi:10.1145/2745754.2745776`.

7 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: Questions Asked Frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '16)*, pages 13–28, San Francisco, California, USA, 2016. Assoc. Comput. Mach., New York. `doi:10.1145/2902251.2902280`.

8 Michael Abseher, Nysret Musliu, and Stefan Woltran. htd – A Free, Open-Source Framework for (Customized) Tree Decompositions and Beyond. In Domenico Salvagnin and Michele Lombardi, editors, *Proceedings of the 14th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR'17)*, volume 10335 of *Lecture Notes in Computer Science*, pages 376–386, Padova, Italy, June 2017. Springer Verlag. `doi:10.1007/978-3-319-59776-8_30`.

9 Vasily Alferov. Cheburashka Vertex Cover solver. Zenodo, June 2019. `doi:10.5281/zenodo.3236897`.

10 Mario Alviano, Francesco Calimeri, Günther Charwat, Minh Dao-Tran, Carmine Dodaro, Giovambattista Ianni, Thomas Krennwallner, Martin Kronegger, Johannes Oetsch, Andreas Pfandler, Jörg Pührer, Christoph Redl, Francesco Ricca, Patrik Schneider, Martin Schwengerer, LaraKatharina Spendier, Johannes Peter Wallner, and Guohui Xiao. The Fourth Answer Set Programming Competition: Preliminary Report. In Pedro Cabalar and TranCao Son, editors, *Proceedings of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)*, volume 8148 of *Lecture Notes in Computer Science*, pages 42–53. Springer Verlag, Corunna, Spain, September 2013. `doi:10.1007/978-3-642-40564-8_5`.

11 Krzysztof Apt. *Principles of Constraint Programming*. Cambridge University Press, Cambridge, New York, NY, USA, 1st edition, 2009.

12 Molham Aref, Balder ten Cate, Todd J. Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L. Veldhuizen, and Geoffrey Washburn. Design and Implementation of the LogicBlox System. In Susan B. Davidson and Zack Ives, editors, *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD'15)*, pages 1371–1382, Melbourne, Victoria, Australia, 2015. Assoc. Comput. Mach., New York. `doi:10.1145/2723372.2742796`.

13 Patricia C. Arocena, Boris Glavic, Radu Ciucanu, and Renée J. Miller. The iBench Integration Metadata Generator. In Chen Li and Volker Markl, editors, *Proceedings of Very Large Data Bases (VLDB) Endowment*, volume 9:3, pages 108–119. Very Large Data Base Endowment, November 2015. URL: `https://github.com/RJMillerLab/ibench`.

14 G. Audemard, F. Boussemart, C. Lecoutre, and C. Piette. XCSP3: an XML-based format designed to represent combinatorial constrained problems. http://xcsp.org, 2016.

15 Nurzhan Bakibayev, Tomáš Kočiský, Dan Olteanu, and Jakub Závodný. Aggregation and Ordering in Factorised Databases. *Proceedings of Very Large Data Bases Endowment (VLDB'13)*, 6(14):1990–2001, September 2013. doi:10.14778/2556549.2556579.

16 Max Bannach and Sebastian Berndt. Practical Access to Dynamic Programming on Tree Decompositions. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *Proceedings of the 26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:13, Helsinki, Finland, 2018. Dagstuhl Publishing. doi:10.4230/LIPIcs.ESA.2018.6.

17 Max Bannach, Sebastian Berndt, and Thorsten Ehlers. Jdrasil: A Modular Library for Computing Tree Decompositions. In Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman, editors, *Proceedings of the 16th International Symposium on Experimental Algorithms (SEA 2017)*, volume 75 of *Dagstuhl Publishing*, pages 28:1–28:21, London, UK, 2017. Leibniz International Proceedings in Informatics (LIPIcs). doi:10.4230/LIPIcs.SEA.2017.28.

18 Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. Benchmarking the Chase. In Floris Geerts, editor, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'17)*, pages 37–52, Chicago, Illinois, USA, 2017. Assoc. Comput. Mach., New York. URL: https://github.com/dbunibas/chasebench.

19 J. Berg, N. Lodha, M. Järvisalo, and S. Szeider. MaxSAT benchmarks based on determining generalized hypertree-width. Technical report, MaxSAT Evaluation 2017, 2017.

20 Sebastian Berndt. Computing Tree Width: From Theory to Practice and Back. In Florin Manea, Russell G. Miller, and Dirk Nowotka, editors, *Sailing Routes in the World of Computation: Proceedings of the 14th Conference on Computability in Europe (CiE 2018)*, volume 10936 of *Lecture Notes in Computer Science*, pages 81–88, Kiel, Germany, 2018. Springer Verlag. doi:10.1007/978-3-319-94418-0_8.

21 Manuel Bichler, Michael Morak, and Stefan Woltran. Single-shot Epistemic Logic Program Solving. In Jeffrey S. Rosenschein and Jérôme Lang, editors, *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 1714–1720. The AAAI Press, 2018.

22 Johannes Blum and Sabine Storandt. Computation and Growth of Road Network Dimensions. In Lusheng Wang and Daming Zhu, editors, *Proceedings of the 24th International Computing and Combinatorics Conference (COCOON 2018)*, volume 10976 of *Lecture Notes in Computer Science*, pages 230–241, Qing Dao, China, July 2018. Springer Verlag. doi:10.1007/978-3-319-94776-1_20.

23 Jori Bomanson, Martin Gebser, and Tomi Janhunen. Improving the Normalization of Weight Rules in Answer Set Programs. In Eduardo Fermé and João Leite, editors, *Proceedings of the 14th European Conference on Logics in Artificial Intelligence (JELIA'14)*, pages 166–180, Funchal, Madeira, Portugal, September 2014. Springer Verlag.

24 Jori Bomanson, Martin Gebser, and Tomi Janhunen. LP2NORMAL and LP2NORMAL2. https://research.ics.aalto.fi/software/asp/lp2normal/, 2016.

25 John A. Bondy and U. Murty. *Graph theory*, volume 244 of *Graduate Texts in Mathematics*. Springer Verlag, 2008.

26 Édouard Bonnet and Florian Sikora. The PACE 2018 Parameterized Algorithms and Computational Experiments Challenge: The Third Iteration. In Christophe Paul and Michal Pilipczuk, editors, *Proceedings of the 13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:15, Helsinki, Finland, 2019. Dagstuhl Publishing. doi:10.4230/LIPIcs.IPEC.2018.26.

27 G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011. doi:10.1145/2043174.2043195.

28 Shaowei Cai, Wenying Hou, Jinkun Lin, and Yuanjie Li. Improving Local Search for Minimum Weight Vertex Cover by Dynamic Strategies. In *IJCAI*, pages 1412–1418. ijcai.org, 2018.

29 Francesco Calimeri, Giovambattista Ianni, and Francesco Ricca. The third open answer set programming competition. *Theory Pract. Log. Program.*, 14:117–135, January 2014. doi:10.1017/S1471068412000105.

30 Francesco Calimeri, Simona Perri, and Jessica Zangari. Optimizing Answer Set Computation via Heuristic-Based Decomposition. *Theory Pract. Log. Program.*, 19(4):603—-628, 2019. doi:10.1017/S1471068419000036.

31 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.

32 Günther Charwat and Stefan Woltran. Dynamic Programming-based QBF Solving. In Florian Lonsing and Martina Seidl, editors, *Proceedings of the 4th International Workshop on Quantified Boolean Formulas (QBF'16)*, volume 1719, pages 27–40. CEUR Workshop Proceedings (CEUR-WS.org), 2016. co-located with 19th International Conference on Theory and Applications of Satisfiability Testing (SAT'16).

33 Günther Charwat and Stefan Woltran. Expansion-based QBF Solving on Tree Decompositions. *Fundam. Inform.*, 167(1-2):59–92, 2019.

34 Jiehua Chen and Sven Grottke. A fast solver for Minimum Vertex Cover. Zenodo, June 2019. doi:10.5281/zenodo.3236992.

35 David Cohen, Peter Jeavons, and Marc Gyssens. A unified theory of structural tractability for constraint satisfaction problems. *J. of Computer and System Sciences*, 74(5):721–743, 2008.

36 Christophe Crespelle, Eduard Eiben, and Kirill Simonov. Vertex Cover Solver for PACE 2019. Zenodo, May 2019. doi:10.5281/zenodo.3235533.

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

38 Marek Cygan, Łukasz Kowalik, Arkadiusz Socała, and Krzysztof Sornat. Approximation and Parameterized Complexity of Minimax Approval Voting. *J. Artif. Intell. Res.*, 63:495–513, 2018. doi:10.1613/jair.1.11253.

39 Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

40 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, *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:9, Aarhus, Denmark, 2017. Dagstuhl Publishing. doi:10.4230/LIPIcs.IPEC.2016.30.

41 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration. In Daniel Lokshtanov and Naomi Nishimura, editors, *Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, volume 89 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:12, Vienna, Austria, 2018. Dagstuhl Publishing. doi:10.4230/LIPIcs.IPEC.2017.30.

42 Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson. The Shortest Path Problem: Ninth DIMACS Implementation Challenge. http://users.diag.uniroma1.it/challenge9/download.shtml, 2009.

43 Marc Denecker, Joost Vennekens, Stephen Bond, Martin Gebser, and Mirosław Truszczyński. The Second Answer Set Programming Competition. In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Computer Science*, pages 637–654. Springer Verlag, Potsdam, Germany, September 2009. doi:10.1007/978-3-642-04238-6_75.

44 Artan Dermaku, Tobias Ganzow, Georg Gottlob, Ben McMahan, Nysret Musliu, and Marko Samer. Heuristic Methods for Hypertree Decomposition. In Alexander Gelbukh and Eduardo F. Morales, editors, *Procedings of the 7th Mexican International Conference on Artificial Intelligence on Advances in Artificial Intelligence (MICAI 2008)*, volume 5317 of *Lecture Notes in Computer Science*, pages 1–11. Springer Verlag, 2008. doi:10.1007/978-3-540-88636-5_1.

45 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer Verlag, 2012.

46 Diego Delle Donne and Guido Tagliavini. Star Routing: Between Vehicle Routing and Vertex Cover. In *COCOA*, volume 11346 of *Lecture Notes in Computer Science*, pages 522–536. Springer, 2018.

47 Eugene F. Dumitrescu, Allison L. Fisher, Timothy D. Goodrich, Travis S. Humble, Blair D. Sullivan, and Andrew L. Wright. Benchmarking treewidth as a practical component of tensor network simulations. *PLOS ONE*, 13(12):1–19, December 2018. doi:10.1371/journal.pone.0207827.

48 Arnaud Durand and Stefan Mengel. Structural tractability of counting of solutions to conjunctive queries. *Theoretical Computer Science*, 57(4):1202–1249, 2015.

49 M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher. daajoe/sat2vc – A CNF-SAT to VC converter, 2019. URL: https://github.com/daajoe/sat2vc.

50 M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher. PACE2019: Track 1 - vertex cover instances. Zenodo, July 2019. doi:10.5281/zenodo.3354609.

51 Leah Epstein, Asaf Levin, and Gerhard J. Woeginger. Vertex Cover Meets Scheduling. *Algorithmica*, 74(3):1148–1173, 2016.

52 Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What Is Known About Vertex Cover Kernelization? In *Adventures Between Lower Bounds and Higher Altitudes*, volume 11011 of *Lecture Notes in Computer Science*, pages 330–356. Springer, 2018.

53 J. K. Fichte. daajoe/gtfs2graphs – A GTFS transit feed to Graph Format Converter, 2016. URL: https://github.com/daajoe/gtfs2graphs.

54 Johannes K. Fichte. ASP Horn Backdoors. https://github.com/daajoe/asp_horn_backdoors, 2014.

55 Johannes K. Fichte. SAT Horn Backdoors. https://github.com/daajoe/sat_horn_backdoors, 2018.

56 Johannes K. Fichte and Markus Hecher. PACE2019: Track 2a+b - hypertree decomposition instances. Zenodo, July 2019. doi:10.5281/zenodo.3354607.

57 Johannes K. Fichte, Markus Hecher, Neha Lodha, and Stefan Szeider. An SMT Approach to Fractional Hypertree Width. In John Hooker, editor, *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming (CP 2018)*, volume 11008 of *Lecture Notes in Computer Science*, pages 109–127, Lille, France, 2018. Springer Verlag.

58 Johannes K. Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Answer Set Solving with Bounded Treewidth Revisited. In Marcello Balduccini and Tomi Janhunen, editors, *Proceedings of the 14th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'17)*, volume 10377 of *Lecture Notes in Computer Science*, pages 132–145, Espoo, Finland, July 2017. Springer Verlag. doi:10.1007/978-3-319-61660-5_13.

59 Johannes K. Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. DynASP2.5: Dynamic programming on tree decompositions in action. In Daniel Lokshtanov and Naomi Nishimura, editors, *Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC'17)*. Dagstuhl Publishing, 2017. doi:10.4230/LIPIcs.IPEC.2017.17.

60 Johannes K. Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Exploiting Treewidth for Projected Model Counting and its Limits. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Proceedings on the 21th International Conference on Theory and Applications*

of Satisfiability Testing (SAT'18), volume 10929 of *Lecture Notes in Computer Science*, pages 165–184, Oxford, UK, July 2018. Springer Verlag.

61 Johannes K. Fichte, Markus Hecher, and Markus Zisser. gpusat2 – An Improved GPU Model Counter. In Simon de Givry and Thomas Schiex, editors, *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP 2018)*, 2019. To appear.

62 Johannes K. Fichte, Neha Lodha, and Stefan Szeider. SAT-Based Local Improvement for Finding Tree Decompositions of Small Width. In Serge Gaspers and Toby Walsh, editors, *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT 2017)*, volume 10491 of *Lecture Notes in Computer Science*, pages 401–411, Melbourne, VIC, Australia, 2017. Springer Verlag. doi:10.1007/978-3-319-66263-3_25.

63 Johannes K. Fichte and Stefan Szeider. Backdoors to Tractable Answer-Set Programming. *Artificial Intelligence*, 220(0):64–103, 2015. doi:10.1016/j.artint.2014.12.001.

64 Johannes Klaus Fichte and Markus Hecher. Treewidth and Counting Projected Answer Sets. In Marcello Balduccini, Yuliya Lierler, and Stefan Woltran, editors, *Proceedings of the 15th International Conference on Logic Programming and Nonmonotonic Reasoning LPNMR 2019*, volume 11481 of *Lecture Notes in Computer Science*, pages 105–119, Philadelphia, PA, USA, 2019. Springer Verlag. doi:10.1007/978-3-030-20528-7_9.

65 Johannes Klaus Fichte, Markus Hecher, and Arne Meier. Counting Complexity for Reasoning in Abstract Argumentation. In Pascal Van Hentenryck and Zhi-Hua Zhou, editors, *Proceedings of the 33rd AAAI Conference on Artificial Intelligence, AAAI 2019*, pages 2827–2834, Honolulu, Hawaii, USA, 2019. The AAAI Press.

66 Johannes Klaus Fichte, Markus Hecher, Stefan Woltran, and Markus Zisser. Weighted Model Counting on the GPU by Exploiting Small Treewidth. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *Proceedings of the 26th Annual European Symposium on Algorithms (ESA'18)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:16. Dagstuhl Publishing, 2018. doi:10.4230/LIPIcs.ESA.2018.28.

67 Aleksander Figiel. An exact vertex cover solver using kernels. Zenodo, June 2019. doi:10.5281/zenodo.3236867.

68 Wolfgang Fischl, Georg Gottlob, Davide M. Longo, and Reinhard Pichler. HyperBench: a benchmark of hypergraphs. http://hyperbench.dbai.tuwien.ac.at, 2017.

69 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 (PODS'19)*, pages 464–480, Amsterdam, Netherlands, 2019. Assoc. Comput. Mach., New York. doi:10.1145/3294052.3319683.

70 Wolfgang Fischl, Georg Gottlob, and Reinhard Pichler. General and Fractional Hypertree Decompositions: Hard and Easy Cases. In Jan Van den Bussche and Marcelo Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'18)*, pages 17–32, Houston, TX, USA, June 2018. Assoc. Comput. Mach., New York.

71 Daniel Fremont. counting-benchmarks. http://tinyurl.com/countingbenchmarks, 2016.

72 Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *J. of the ACM*, 29(1):24–32, 1982.

73 Marco Gario. Backdoors for SAT. Master's thesis, TU Dresden, 2011. Program sources at https://marco.gario.org/work/master/.

74 Marco Gario. HornVCBuilder. https://marco.gario.org/work/master/, 2011.

75 Marco Gario. Horn backdoor detection via Vertex Cover: Benchmark Description. In Adrian Balint, Anton Belov, Daniel Diepold, Simon Gerber, Matti Järvisalo, and Carsten Sinz, editors, *Proceedings of SAT Challenge 2012; Solver and Benchmark Descriptions*, Helsinki, Finland, 2012. University of Helsinki.

76 Serge Gaspers, Joachim Gudmundsson, Mitchell Jones, Julián Mestre, and Stefan Rümmele. Turbocharging Treewidth Heuristics. In Jiong Guo and Danny Hermelin, editors, *Proceedings*

of the 11th International Symposium on Parameterized and Exact Computation (IPEC'16), volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:13. Dagstuhl Publishing, 2017. doi:10.4230/LIPIcs.IPEC.2016.13.

77 M. Gebser, T. Schaub, S. Thiele, and P. Veber. Detecting Inconsistencies in Large Biological Networks with Answer Set Programming. *Theory Pract. Log. Program.*, 11(2-3):323–360, 2011.

78 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo = ASP + Control: Preliminary Report. *CoRR*, abs/1405.3694, 2014. arXiv:1405.3694.

79 Martin Gebser, Lengning Liu, Gayathri Namasivayam, André Neumann, Torsten Schaub, and Mirosław Truszczyński. The First Answer Set Programming System Competition. In Chitta Baral, Gerhard Brewka, and John Schlipf, editors, *Proceedings of the 9th Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483 of *Lecture Notes in Computer Science*, pages 3–17, Tempe, AZ, USA, May 2007. Springer Verlag. doi:10.1007/978-3-540-72200-7_3.

80 F. Geerts, G. Mecca, P. Papotti, and D. Santoro. Mapping and cleaning. In Isabel Cruz, Elena Ferrari, and Yufei Tao, editors, *Proceedings of the IEEE 30th International Conference on Data Engineering (ICDE'14)*, pages 232–243, March 2014.

81 Martin Josef Geiger. Implementation of a Metaheuristic for the Steiner Tree Problem in Graphs. Medeley Data, 2018. doi:10.17632/yf9vpkgwdr.1.

82 G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decompositions and Tractable Queries. *J. of Computer and System Sciences*, 64(3):579–627, 2002.

83 Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. *Treewidth and Hypertree Width*, pages 3—-38. Cambridge University Press, Cambridge, 2014. doi:10.1017/CBO9781139177801.002.

84 Georg Gottlob, Zoltan Miklos, Nysret Musliu, and Marko Samer. Hypertree Complementary Approaches to Constraint Satisfaction, 2016. URL: https://www.dbai.tuwien.ac.at/proj/hypertree/downloads.html.

85 Georg Gottlob and Marko Samer. A Backtracking-based Algorithm for Hypertree Decomposition. *J. of Experimental Algorithmics*, 13:1:1.1–1:1.19, February 2009.

86 Martin Grohe and Dániel Marx. Constraint Solving via Fractional Edge Covers. In *Proceedings of the of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006)*, pages 289–298. ACM Press, 2006.

87 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms*, 11(1):Art. 4, 20, 2014.

88 Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158–182, 2005.

89 LLC Gurobi Optimization. Gurobi Optimizer Reference Manual, 2019. URL: http://www.gurobi.com.

90 Michael Hamann and Ben Strasser. Graph Bisection with Pareto Optimization. *J. of Experimental Algorithmics*, 23:1.2:1–1.2:34, February 2018. doi:10.1145/3173045.

91 Falko Hegerfeld and Florian Nelles. hubhegnel/pace-2019: Release for zenodo. Zenodo, May 2019. doi:10.5281/zenodo.3234674.

92 Demian Hespe, Sebastian Lamm, Christian Schulz, and Darren Strash. WeGotYouCovered, May 2019. doi:10.5281/zenodo.2816116.

93 Demian Hespe, Sebastian Lamm, Christian Schulz, and Darren Strash. WeGotYouCovered: The winning solver from the PACE 2019 implementation challenge, vertex cover track. *CoRR*, 2019. arXiv:1908.06795.

94 Marijn J. H. Heule, Matti Juhani Järvisalo, and Martin Suda, editors. *Proceedings of the SAT Competition 2018: Solver and Benchmark Descriptions*, volume B. Department of Computer Science, University of Helsinki, University of Helsinki, 2018.

95 Holger H. Hoos and Thomas Stützle. SATLIB: An online resource for research on SAT. In Ian P. Gent, Hans van Maaren, and Toby Walsh, editors, *Proceedings of the 3rd Workshop on*

Satisfiability (SAT'00), pages 283–292, Renesse, The Netherlands, 2000. IOS Press. SATLIB is available online at `www.satlib.org`.

96 Md. Rafiqul Islam, Imran Hossain Arif, and Rifat Hasan Shuvo. Generalized vertex cover using chemical reaction optimization. *Appl. Intell.*, 49(7):2546–2566, 2019.

97 Yoichi Iwata. Linear-Time Kernelization for Feedback Vertex Set. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:14. Dagstuhl Publishing, 2017. `doi:10.4230/LIPIcs.ICALP.2017.68`.

98 Yoichi Iwata and Yusuke Kobayashi. Improved Analysis of Highest-Degree Branching for Feedback Vertex Set. *CoRR*, abs/1905.12233, 2019. `arXiv:1905.12233`.

99 Yoichi Iwata and Takuto Shigemura. Separator-Based Pruned Dynamic Programming for Steiner Tree. In *AAAI*, pages 1520–1527. AAAI Press, 2019.

100 T. Janhunen and I. Niemelä. The Answer Set Programming Paradigm. *AI Magazine*, 37(3):13–24, 2016. `doi:10.1609/aimag.v37i3.2671`.

101 P. Jégou, H. Kanso, and C. Terrioux. On the Relevance of Optimal Tree Decompositions for Constraint Networks. In *Proceedings of the IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI 2018)*, pages 738–743. IEEE Computer Soc., November 2018. `doi:10.1109/ICTAI.2018.00116`.

102 Roland Kaminski. clingo - A grounder and solver for logic programs. `https://github.com/potassco/clingo`, 2019.

103 Kustaa Kangas, Mikko Koivisto, and Sami Salonen. A Faster Tree-Decomposition Based Algorithm for Counting Linear Extensions. In Christophe Paul and Michal Pilipczuk, editors, *Proceedings of the 13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:13. Dagstuhl Publishing, 2019. `doi:10.4230/LIPIcs.IPEC.2018.5`.

104 Richard M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

105 Steven Kelk, Georgios Stamoulis, and Taoyang Wu. Treewidth distance on phylogenetic trees. *Theoretical Computer Science*, 731:99–117, 2018. `doi:10.1016/j.tcs.2018.04.004`.

106 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. *CoRR*, abs/1504.04044, 2017.

107 Krzysztof Kiljan and Marcin Pilipczuk. Experimental Evaluation of Parameterized Algorithms for Feedback Vertex Set. In Gianlorenzo D'Angelo, editor, *Proceedings of the 17th International Symposium on Experimental Algorithms (SEA 2018)*, volume 103 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:12. Dagstuhl Publishing, 2018. `doi:10.4230/LIPIcs.SEA.2018.12`.

108 Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. MIT Press, 2009.

109 Tuukka Korhonen, Jeremias Berg, and Matti Järvisalo. Enumerating Potential Maximal Cliques via SAT and ASP. In Thomas Eiter and Sarit Kraus, editors, *Proceedings of the International Joint Conference on Artificial Intelligence IJCAI 2019*, Macao, China, 2019. The AAAI Press.

110 Tuukka Korhonen, Jeremias Berg, and Matti Järvisalo. Solving Graph Problems via Potential Maximal Cliques: An Experimental Evaluation of the Bouchitté–Todinca Algorithm. *J. of Experimental Algorithmics*, 24(1):1.9:1–1.9:19, February 2019. `doi:10.1145/3301297`.

111 Philipp Klaus Krause, Lukas Larisch, and Felix Salfelder. The tree-width of C. *Discr. Appl. Math.*, 2019. `doi:10.1016/j.dam.2019.01.027`.

112 R. Krithika, Diptapriyo Majumdar, and Venkatesh Raman. Revisiting Connected Vertex Cover: FPT Algorithms and Lossy Kernels. *Theory Comput. Syst.*, 62(8):1690–1714, 2018.

113 Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian. Rev2: Fraudulent user prediction in rating platforms. In Yi Chang and Yan Liu, editors, *Proceedings of the 11th ACM International Conference on Web Search and Data Mining (WSDM'18)*, pages 333–341, Marina Del Rey, CA, USA, 2018. Assoc. Comput. Mach., New York.

114 Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. Edge weight prediction in weighted signed networks. In Francesco Bonchi and Josep Domingo-Ferrer, editors, *Proceedings of the IEEE 16th International Conference on Data Mining (ICDM'16)*, pages 221–230, Barcelona, Catalonia, Spain, 2016. IEEE Computer Soc.

115 Hung V. Le. *Structural Results and Approximation Algorithms in Minor-free Graphs*. PhD thesis, Oregon State University, 2018. URL: `https://ir.library.oregonstate.edu/concern/graduate_thesis_or_dissertations/d217qv924`.

116 Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How Good Are Query Optimizers, Really? *Proceedings of Very Large Data Bases (VLDB) Endowment*, 9(3):204–215, November 2015.

117 Jure Leskovec. Stanford Network Analysis Project. `https://snap.stanford.edu`, 2009.

118 Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Predicting Positive and Negative Links in Online Social Networks. In *Proceedings of the 19th International Conference on World Wide Web (WWW '10)*, pages 641–650, New York, NY, USA, 2010. Assoc. Comput. Mach., New York. `doi:10.1145/1772690.1772756`.

119 Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed Networks in Social Media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*, pages 1361–1370, New York, NY, USA, 2010. Assoc. Comput. Mach., New York. `doi:10.1145/1753326.1753532`.

120 Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph Evolution: Densification and Shrinking Diameters. *ACM Trans. Knowl. Discov. Data*, 1(1), March 2007. `doi:10.1145/1217299.1217301`.

121 Zijie Li and Peter van Beek. Finding Small Backdoors in SAT Instances. In *Canadian Conference on AI*, volume 6657 of *Lecture Notes in Computer Science*, pages 269–280. Springer, 2011.

122 Cong Han Lim and Stephen Wright. k-Support and Ordered Weighted Sparsity for Overlapping Groups: Hardness and Algorithms. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Proceedings of Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pages 284–292. Curran Associates, Inc., 2017.

123 Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. SAT-Encodings for Special Treewidth and Pathwidth. In Serge Gaspers and Toby Walsh, editors, *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT 2017)*, volume 10491 of *Lecture Notes in Computer Science*, pages 429–445, Melbourne, VIC, Australia, 2017. Springer Verlag. `doi:10.1007/978-3-319-66263-3_27`.

124 Davide Mario Longo. Pace2019 Hypertree Width Exact. Zenodo, May 2019. `doi:10.5281/zenodo.3236358`.

125 Davide Mario Longo. Pace2019 Hypertree Width Heuristic. Zenodo, May 2019. `doi:10.5281/zenodo.3236369`.

126 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.

127 Silviu Maniu, Pierre Senellart, and Suraj Jog. An Experimental Study of the Treewidth of Real-World Graph Data. In Pablo Barcelo and Marco Calautti, editors, *Proceedings of the 22nd International Conference on Database Theory (ICDT 2019)*, volume 127 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:18. Dagstuhl Publishing, 2019. `doi:10.4230/LIPIcs.ICDT.2019.12`.

128 Marco Maratea, Francesco Ricca, Wolfgang Faber, and Nicola Leone. Look-back techniques and heuristics in DLV: Implementation, evaluation, and comparison to QBF solvers. *J. Algorithms*, 63(1–3):70–89, 2008. Benchmarks can be found at `http://asparagus.cs.uni-potsdam.de/contest/downloads/benchmarks-score-dlp.tgz`. `doi:10.1016/j.jalgor.2008.02.006`.

129 Thorsten Ehlers Max Bannach, Sebastian Berndt and Dirk Nowotka. SAT-Encodings of Tree Decompositions. In Marijn Heule, Matti Juhani Järvisalo, and Martin Suda, editors, *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*, number Report B-2018-1 in B, page 72. University of Helsinki, Department of Computer Science, 2018.

130 Julian McAuley and Jure Leskovec. Learning to Discover Social Circles in Ego Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS'12)*, pages 539–547, USA, 2012. Curran Associates Inc.

131 Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider. Solving #SAT using vertex covers. *Acta Informatica*, 44(7-8):509–523, 2007.

132 Dan Olteanu and Jakub Závodný. Size Bounds for Factorised Representations of Query Results. *ACM Trans. Database Syst.*, 40(1):2:1–2:44, March 2015. `doi:10.1145/2656335`.

133 Steven Prestwich. CNF Encodings. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, chapter 2, pages 75–97. IOS Press, 2009.

134 Noam Ravid, Dori Medini, and Benny Kimelfeld. Ranked Enumeration of Minimal Triangulations. In Dan Suciu and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2019)*, pages 74–88, Amsterdam, Netherlands, 2019. Assoc. Comput. Mach., New York. `doi:10.1145/3294052.3319678`.

135 Matei Ripeanu, Adriana Iamnitchi, and Ian Foster. Mapping the Gnutella Network. *IEEE Internet Computing*, 6(1):50–57, January 2002. `doi:10.1109/4236.978369`.

136 Neil Robertson and P.D. Seymour. Graph Minors. II. Algorithmic Aspects of Tree-Width. *J. Algorithms*, 7(3):309–322, 1986.

137 Marko Samer and Stefan Szeider. Backdoor Trees. In *AAAI*, pages 363–368. AAAI Press, 2008.

138 Marko Samer and Stefan Szeider. Fixed-Parameter Tractability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, chapter 13, pages 425–454. IOS Press, 2009.

139 André Schidler and Stefan Szeider. HtdSMT - An SMT based solver for hypertree decompositions. Zenodo, May 2019. `doi:10.5281/zenodo.3236333`.

140 C. Sinz, A. Kaiser, and W. Küchlin. Formal methods for the validation of automotive product configuration data. *AI EDAM*, 17(01):75–97, 2003. URL: `http://www-sr.informatik.uni-tuebingen.de/~sinz/DC`.

141 Ben Strasser. Road Graphs as Tree-Decomposition PACE Test Instances, 2016. URL: `https://github.com/ben-strasser/road-graphs-pace16`.

142 Ben Strasser. Computing Tree Decompositions with FlowCutter: PACE 2017 Submission. *CoRR*, abs/1709.08949, 2017. `arXiv:1709.08949`.

143 Hisao Tamaki. Positive-Instance Driven Dynamic Programming for Treewidth. In Kirk Pruhs and Christian Sohler, editors, *Proceedings of the 25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:13, Vienna, Austria, 2017. Dagstuhl Publishing. `doi:10.4230/LIPIcs.ESA.2017.68`.

144 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *J. Comb. Optim.*, 37(4):1283–1311, May 2019. `doi:10.1007/s10878-018-0353-z`.

145 Transaction Processing Performance Council (TPC). TPC-H decision support benchmark. Technical report, TPC, 2014. URL: `http://www.tpc.org/tpch/default.asp`.

146 James Trimble. jamestrimble/heidi: v1.0.0: Pace Challenge 2019 version. Zenodo, June 2019. `doi:10.5281/zenodo.3237427`.

147 James Trimble. jamestrimble/hypebeast: v1.0.0: PACE Challenge 2019 version. Zenodo, May 2019. doi:10.5281/zenodo.3082314.

148 James Trimble. jamestrimble/peaty: v1.0.0: PACE Challenge 2019 entry. Zenodo, May 2019. doi:10.5281/zenodo.3082356.

149 Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

150 René van Bevern, Till Fluschnik, and Oxana Yu. Tsidulko. Parameterized algorithms and data reduction for safe convoy routing. *CoRR*, abs/1806.09540, 2018. arXiv:1806.09540.

151 Tom C. van der Zanden and Hans L. Bodlaender. Computing Treewidth on the GPU. In Daniel Lokshtanov and Naomi Nishimura, editors, *Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, volume 89 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:13, Vienna, Austria, 2018. dp. doi:10.4230/LIPIcs.IPEC.2017.29.

152 Tom Cornelis van der Zanden. *Theory and Practical Applications of Treewidth*. PhD thesis, Utrecht University, 2019.

153 Rim van Wersch and Steven Kelk. ToTo: An open database for computation, storage and retrieval of tree decompositions. *Discr. Appl. Math.*, 217:389–393, 2017. doi:10.1016/j.dam.2016.09.023.

154 Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. Optil.Io: Cloud Based Platform For Solving Optimization Problems Using Crowdsourcing Approach. In Eric Gilbert and Karrie Karahalios, editors, *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*, CSCW '16 Companion, pages 433–436, New York, NY, USA, 2016. Assoc. Comput. Mach., New York. doi:10.1145/2818052.2869098.

155 Ke Xu. BHOSLIB: Benchmarks with Hidden Optimum Solutions for Graph Problems (Maximum Clique, Maximum Independent Set, Minimum Vertex Cover and Vertex Coloring). http://sites.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm, 2014.

156 Bogdan Zavalnij and Sandor Szabo. zbogdan/pace-2019 a. Zenodo, May 2019. doi:10.5281/zenodo.3228802.

157 Yuting Zhao and Fangzhen Lin. Answer Set Programming Phase Transition: A Study on Randomly Generated Programs. In Catuscia Palamidessi, editor, *Proceedings of the 19th International Conference on Logic Programming (ICLP'03)*, volume 2916 of *Lecture Notes in Computer Science*, pages 239–253, Mumbai, India, December 2003. Springer Verlag. doi:10.1007/978-3-540-24599-5_17.

158 Michal Ziobro and Marcin Pilipczuk. Finding Hamiltonian Cycle in Graphs of Bounded Treewidth: Experimental Evaluation. *CoRR*, abs/1803.00927, 2018. arXiv:1803.00927.