11th International Symposium on Parameterized and Exact Computation

IPEC'16, August 24–26, 2016, Aarhus, Denmark

Edited by

Jiong Guo

Danny Hermelin

Editors

Jiong Guo
School of Computer Science and Technology
Shandong University
Jinan
China
jguo@sdu.edu.cn

Danny Hermelin
Department of Industrial Engineering and Management
Ben-Gurion University of the Negev
Beer Sheva
Israel
hermelin@bgu.ac.li

ACM Classification 1998
F.1.3 Complexity Measures and Classes, F.2 Analysis of Algorithms and Problem Complexity, G.2 Discrete Mathematics

ISBN 978-3-95977-023-1

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-023-1.

Publication date
February, 2017

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 http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.IPEC.2016.0

ISBN 978-3-95977-023-1 **ISSN 1868-8969** **http://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

http://www.dagstuhl.de/lipics

Contents

11th International Symposium on Parameterized and Exact Computation (IPEC 2016).
Editors: Jiong Guo and Danny Hermelin
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the papers presented at IPEC 2016: the 11th International Symposium on Parameterized and Exact Computation held during August 24–26, 2016, in Aarhus, Denmark. IPEC was held togeter with seven other algorithms conferences as part of the annual ALGO congress.

The International Symposium on Parameterizd 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.
In response to the call of papers, 48 papers were submitted. Each submission was reviewed by at least 3 reviewers. The reviews came from the 14 members of the program committee, and from 84 external reviewers contributing 107 external reviews. The program committee held electronic meetings through the EasyChair.

Previous IPECs	
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

The program committee felt that the median submission quality was very high, and in the end selected 28 of the submissions for presentation at the symposium and for inclusion in the proceedings volume. The program committee presented the IPEC 2016 Best Paper Awards to Michał Włodarczyk for the paper *Clifford Algebras Meet Tree Decompositions*, Kitty Meeks for the paper *Randomised Enumeration of Small Witnesses Using a Decision Oracle*, and Archontia Giannopoulou, Michał Pilipczuk, Jean-Florent Raymond, Dimitrios Thilikos, and Marcin Wrochna for the paper *Cutwidth: Obstructions and Algorithmic Aspects*. The program committee also presented the IPEC 2016 Excellent Student Paper Awards to Michał Włodarczyk for the paper *Clifford Algebras Meet Tree Decompositions* and R. Krithika, Abhishek Sahu, and Prafullkumar Tale for the paper *Dynamic Parameterized Problems*.

IPEC invited one plenary speaker to the ALGO meeting, Andreas Björklund, as part of the award ceremony for the 2016 EATCS-IPEC Nerode Prize for outstanding papers in the area of multvariate algorithmics. The award was given by a committee consisting of Jan Arne Telle, David Eppstein, and Dániel Marx to Andreas Björklund for his paper *Determinant Sums for Undirected Hamiltonicity* [SIAM J. Comput., 43(1), 2014]. We thank Andreas for accepting our invitation and for contributing an excellent talk to IPEC 2016.

We would like to thank the program committee, together with the external reviewers for their commitment in the whole paper reviewing process. We also thank all authors who submitted their work for consideration. Finally, we are grateful to the local organizers of ALGO, chaired by Gerth Stølting Brodal, for the efforts, which made chairing IPEC an enjoyable experience.

Jiong Guo and Danny Hermelin
Jinan and Beer-Sheva, October 2016

11th International Symposium on Parameterized and Exact Computation (IPEC 2016).
Editors: Jiong Guo and Danny Hermelin
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Cristina Bazgan
LAMSADE, Universite Paris-Dauphine

Yixin Cao
Hong Kong Polytechnic University

Jiong Guo (chair)
Shandong University

Danny Hermelin (chair)
Ben-Gurion University of the Negev

Bart M. P. Jansen
Eindhoven University of Technology

Eun-Jun Kim
CNRS-Paris Dauphine

Christian Komusiewicz
Friedrich-Schiller-Universität Jena

Geevarghese Philip
Max-Planck-Institut für Informatik

Hadas Shachnai
The Technion

Stefan Szeider
TU Wien

Jan Arne Telle
University of Bergen

Meirav Zehavi
The Technion

11th International Symposium on Parameterized and Exact Computation (IPEC 2016).
Editors: Jiong Guo and Danny Hermelin

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

External Reviewers

Faisal Abu-Khzam
Julien Baste
René van Bevern
Hans L. Bodlaender
Robert Bredereck
Hubie Chen
Marek Cygan
Holger Dell
Eduard Eiben
Moran Feldman
Qilong Feng
Henning Fernau
Till Fluschnik
Fedor Formin
Martin Fürer
Serge Gaspers
Petr Golovach
Alexander Grigoriev
Gregory Gutin
Thore Husfeldt
Davis Issac
Dušan Knop
Mikko Koivisto
Sudeshna Kolay
Stavros Kolliopoulos
Lukasz Kowalik
Dieter Kratsch
Stefan Kratsch
R. Krithika
O-Joung Kwon
Michael Lampis
Euiwoong Lee
Wenjun Li
Mathieu Liedloff
Hong Liu
Yang Liu
Daniel Lokshtanov
Amaldev Manuel
Pranabendu Misra
Valia Mitsou
Matthias Mnich
Hendrik Molter
Amer Mouawad
Haiko Müller
Martin Mundhenk
N. S. Narayanaswamy
Jesper Nederlof
André Nichterlein
Rolf Niedermeier
Sebastian Ordyniak
Fahad Panolan
Marcin Pilipczuk
Michał Pilipczuk
M. Praveen
Ashutosh Rai
Venkatesh Raman
M. S. Ramanujan
Jean-Florent Raymond
Felix Reidl
Noy Rotbart
Stefan Rümmele
Michalis Samaris

Ignasi Sau

Saket Saurabh

Roy Schwartz

Yash Raj Shrestha

Friedrich Slivovsky

Manuel Sorge

A. V. Sreejith

Yann Strozecki

Ondřej Suchý

Peng Sun

Prafullkumar Tale

Nimrod Talmon

Thomas C. Van Dijk

Johan M. M. Van Rooij

Stéphane Vialette

Fernando Sánchetz Villaamil

Magnus Waldström

Justin Ward

Oren Weimann

Andreas Wiese

Marcin Wrochna

Yongjie Yang

List of Authors

11th International Symposium on Parameterized and Exact Computation (IPEC 2016).
Editors: Jiong Guo and Danny Hermelin
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Manuel Sorge

Ondřej Suchý

Blair D. Sullivan

Prafullkumar Tale

Hisao Tamaki

Till Tantau

Jan Arne Telle

Eva-Lotta Teutrine

Dimitrios Thilikos

Zsolt Tuza

Andrew van der Poel

Johan M. M. van Rooij

Michał Włodarczyk

Marcin Wrochna

Jules J. H. M. Wulms

Determinant Sums for Hamiltonicity

Andreas Björklund

Department of Computer Science, Lund University, Lund, Sweden
andreas.bjorklund@cs.lth.se

Abstract

The best worst case guarantee algorithm to see if a graph has a Hamiltonian cycle, a closed tour visiting every vertex exactly once, for a long time was based on dynamic programming over all the vertex subsets of the graph. In this talk we will show some algebraic techniques that can be used to see if a graph has a Hamiltonian cycle much faster. These techniques utilize sums over determinants of matrices.

In particular we will show how you can find out if an undirected graph has a Hamiltonian cycle much faster, but we will also talk about some partial results for the directed case and modular counting.

1998 ACM Subject Classification G.2.2 Graph Algorithms, I.1.2 Analysis of Algorithms

Keywords and phrases Hamiltonian cycle, exact algorithm, matrix determinant, algebraic techniques

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.1

Category Invited Talk

11th International Symposium on Parameterized and Exact Computation (IPEC 2016).
Editors: Jiong Guo and Danny Hermelin; Article No. 1; pp. 1:1–1:1

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

Improved Algorithms and Combinatorial Bounds for Independent Feedback Vertex Set*

Akanksha Agrawal[1], Sushmita Gupta[2], Saket Saurabh[3], and Roohani Sharma[4]

1 **Department of Informatics, University of Bergen, Norway**
akanksha.agrawal@uib.no
2 **Department of Informatics, University of Bergen, Norway**
sushmita.gupta@uib.no
3 **Institute of Mathematical Sciences, HBNI, Chennai, India**
saket@imsc.res.in
4 **Institute of Mathematical Sciences, HBNI, Chennai, India**
roohani@imsc.res.in

Abstract

In this paper we study the "independent" version of the classic Feedback Vertex Set problem in the realm of parameterized algorithms and moderately exponential time algorithms. More precisely, we study the Independent Feedback Vertex Set problem, where we are given an undirected graph G on n vertices and a positive integer k, and the objective is to check if there is an *independent feedback vertex set* of size at most k. A set $S \subseteq V(G)$ is called an *independent feedback vertex set (ifvs)* if S is an independent set and $G \setminus S$ is a forest. In this paper we design two deterministic exact algorithms for Independent Feedback Vertex Set with running times $\mathcal{O}^\star(4.1481^k)$[1] and $\mathcal{O}^\star(1.5981^n)$. In fact, the algorithm with $\mathcal{O}^\star(1.5981^n)$ running time finds the smallest sized ifvs, if an ifvs exists. Both the algorithms are based on interesting measures and improve the best known algorithms for the problem in their respective domains. In particular, the algorithm with running time $\mathcal{O}^\star(4.1481^k)$ is an improvement over the previous algorithm that ran in time $\mathcal{O}^\star(5^k)$. On the other hand, the algorithm with running time $\mathcal{O}^\star(1.5981^n)$ is the first moderately exponential time algorithm that improves over the naïve algorithm that enumerates all the subsets of $V(G)$. Additionally, we show that the number of minimal ifvses in any graph on n vertices is upper bounded by 1.7485^n.

1998 ACM Subject Classification G.2.2 Graph Algorithms, I.1.2 Analysis of Algorithms

Keywords and phrases independent feedback vertex set, fixed parameter tractable, exact algorithm, enumeration

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.2

1 Introduction

Feedback Vertex Set (FVS) is one of the classic NP-complete problems. In fact, it is one of the problems in the Karp's famous list of twenty one NP-complete problems [21]. FVS together with several of its variants have been extensively studied from both combinatorial as well as algorithmic view points. Indeed, FVS is one of the central problems in any

* The research leading to these results has received funding from the European Research Council (ERC) via grant PARAPPROX, reference 306992.

[1] The $\mathcal{O}^\star()$ notation suppresses polynomial factors in the running-time expression.

algorithmic paradigm that has to cope with NP-hardness, examples being : approximation algorithms, moderately exponential time algorithms, enumeration algorithms and parameterized algorithms [2, 3, 4, 7, 8, 11, 14, 16, 20, 22, 23, 28, 29, 33]. The goal of this article is to study the independent set version of FVS in the realm of parameterized complexity, moderately exponential time algorithms and combinatorial upper bounds.

We begin by formally defining the problem. The formal description of the problem being studied is as follows.

Independent Feedback Vertex Set (IFVS)	**Parameter:** k
Input: An undirected graph G on n vertices and a positive integer k.	
Question: Is there an independent feedback vertex set of size at most k?	

IFVS and Parameterized Complexity. FVS together with Vertex Cover is one of the most well studied problem in the field of parameterized complexity [3, 4, 22, 28]. The other variants of FVS on undirected graphs that have been studied extensively, include, Subset FVS [8, 23, 33], Group FVS [7, 20, 33], Connected FVS [25], Simultaneous FVS [1] and indeed IFVS [24, 30, 31]. The current champion algorithms for FVS are: a randomized algorithm with running time $\mathcal{O}^\star(3^k)$ [6] and a deterministic algorithm running in time $\mathcal{O}^\star(3.619^k)$ [22]. Misra et al. [24] introduced IFVS in 2011 (in the conference version of the cited paper) as a generlization of FVS and gave an algorithm with running time $\mathcal{O}^\star(5^k)$. They also designed a polynomial kernel of size $\mathcal{O}(k^3)$ for the problem. Later, Song claimed a deterministic algorithm with running time $\mathcal{O}^\star(4^k)$ for the problem [30]. However, the algorithm of Song [30] does not seem to be correct.[2] Tamura et al. [31] studied IFVS on special graph classes and showed that the problem remains NP-complete even on planar bipartite graphs of maximum degree four. They also designed linear time algorithms for graphs of bounded treewidth, chordal graphs and cographs. Finally, they gave an algorithm with running time $\mathcal{O}(2^{\mathcal{O}(\sqrt{k}\log k)}n)$ for IFVS on planar graphs. We refer the reader to the recent book on parameterized algorithms for more details regarding the paradigm of parameterized complexity, as well as about the literature on the FVS problem [5]. Our first main result is the following result regarding IFVS.

▶ **Theorem 1.** *There is an algorithm for* IFVS *running in time* $\mathcal{O}^\star(4.1481^k)$.

Our new algorithm is based on iterative compression and the subroutine for iterative compression is based on branching. The branching algorithm itself exploits (a) the fact that once we select a vertex in the independent feedback vertex set then all its neighbors must be in the forest; and (b) an interesting variation of the measure used for analyzing the fastest known deterministic algorithm for FVS [22]. Finally, we also observe that the randomized algorithm designed for FVS, running in time $\mathcal{O}^\star(3^k)$ [6], can be adapted to design a randomized $\mathcal{O}^\star(3^k)$ time algorithm for IFVS.

IFVS and Moderately Exponential Time Algorithms. In moderately exponential time algorithms (or exact algorithms for short), the objective is to design an algorithm for optimization version of a problem that is better than the naïve brute force algorithm. In particular, for FVS the goal will be to design an algorithm that runs in time c^n, $c < 2$ a constant, and finds a minimum sized set S such that $G - S$ is a forest. We refer to the book

[2] We have approached the author with concrete questions but he has not yet responded. Furthermore, we give a family of counter-examples to his algorithm in the Section 3.2.

of Fomin and Kratsch for more details regarding moderately exponential time algorithms [17]. Obtaining a non-trivial exact algorithm for FVS was open for quite some time before Razgon obtained an algorithm with running time $\mathcal{O}(1.8899^n)$ [29]. Later this algorithm was improved to $\mathcal{O}^\star(1.7347^n)$ [18]. Recently, Fomin et al. [13] obtained an interesting result relating parameterized algorithms and exact algorithms. Roughly speaking, they showed that if a problem (satisfying some constraints) has $\mathcal{O}^\star(c^k)$ time algorithm parameterized by the solution size, then there is an exact algorithm running in time $\mathcal{O}^\star((2-\frac{1}{c})^n)$. Both FVS and IFVS satisfies the required constraints and thus we immediately obtain the following exact algorithm for IFVS: (a) a randomized algorithm running in time $\mathcal{O}^\star((2-\frac{1}{3})^n) = \mathcal{O}^\star(1.6667^n)$; and (b) a deterministic algorithm running in time $\mathcal{O}^\star((2-\frac{1}{4.1481})^n) = \mathcal{O}^\star(1.7590^n)$. We give a recursive algorithm based on classical measure and conquer [15, 17] and design faster algorithm than both the mentioned algorithms. In particular, we prove the following theorem.

▶ **Theorem 2.** *There is an algorithm for* IFVS *running in time* $\mathcal{O}^\star(1.5981^n)$.

Combinatorial Upper Bounds. In our final section we address the following question: How many minimal ifvses are there in any graph on n vertices? Proving an upper bound on the number of combinatorial structures is an old and vibrant area. Some important results in this area include an upper bound of

- $3^{n/3}$ on the number of maximal independent sets in a graph [26].
- 1.667^n on the number of minimal feedback vertex sets in a tournament [13].
- 1.8638^n on the number of minimal feedback vertex sets in a graph [14, 16].

One can easily observe that every minimal ifvs is also a minimal feedback vertex set. Thus, an upper bound of 1.8638^n on the number of minimal ifvses in any graph on n vertices follows by [14]. As our final result, we give an improved upper bound on the number of minimal ifvses in any graph on n vertices. We obtain this result by applying reduction rules and branching rules with a carefully choosen measure. At the base case we prove that counting the number of spanning trees is same as counting the number of minimal ifvses. For bounding the number of spanning trees we use the result of Grimmett [19].

▶ **Theorem 3.** *A graph G on n vertices has at most* 1.7485^n *minimal ifvses.*

Let n be divisible by 3 and G be a graph that is union of $n/3$ vertex disjoint triangles. Then any minimal ifvs must contain exactly one vertex from each of $n/3$ triangles and thus G has $3^{n/3}$ minimal ifvses. Closing the gap between $3^{n/3}$ and 1.7485^n remains an interesting open problem. The proofs of Theorem 2 and 3 are omitted due to space constraints.

2 Preliminaries

In this section, we state some basic definitions and introduce terminology from graph theory and algorithms. We also establish some of the notations that will be used throughout.

We denote the set of natural numbers by $\mathbb{N}$. To describe the running times of our algorithms, we will use the $\mathcal{O}^\star$ notation. Given $f : \mathbb{N} \to \mathbb{N}$, we define $\mathcal{O}^\star(f(n))$ to be $\mathcal{O}(f(n) \cdot p(n))$, where $p(\cdot)$ is some polynomial function. That is, the $\mathcal{O}^\star$ notation suppresses polynomial factors in the running-time expression.

Graphs. We use standard terminology from the book of Diestel [9] for those graph-related terms which are not explicitly defined here. We only consider finite graphs possibly having loops and multi-edges. For a graph G, by $V(G)$ and $E(G)$ we denote the vertex and edge sets of the graph G, respectively. For a vertex $v \in V(G)$, we use $d_G(v)$ to denote the degree of v,

i.e the number of edges incident on v, in the (multi) graph G. We also use the convention that a loop at a vertex v contributes two to its degree. For a vertex subset $S \subseteq V(G)$, $G[S]$ and $G \setminus S$ are the graphs induced on S and $V(G) \setminus S$, respectively. For an edge subset $S \subseteq E(G)$, by $G \setminus S$, we denote the graph obtained after removing edges in S from G. For a vertex subset $S \subseteq V(G)$, we let $N_G(S)$ and $N_G[S]$ denote the open and closed neighbourhood of S in G. That is, $N_G(S) = \{v \mid (u,v) \in E(G), u \in S\} \setminus S$ and $N_G[S] = N_G(S) \cup S$. We drop the sub-script G from $d_G(v), N_G(S), N_G[S]$ whenever the context is clear. For a graph G and an edge $e \in E(G)$, G/e denotes the graph obtained after contracting e in G.

A *path* in a graph is a sequence of distinct vertices $v_0, v_1, \ldots, v_\ell$ such that (v_i, v_{i+1}) is an edge for all $0 \leq i < \ell$. A *cycle* in a graph is a sequence of distinct vertices $v_0, v_1, \ldots, v_\ell$ such that $(v_i, v_{(i+1) \mod (\ell+1)})$ is an edge for all $0 \leq i \leq \ell$. We note that both a double edge and a loop are cycles. A tree T rooted at $r \in V(T)$ is called as a *star* if $E(T) = \{(v,r) \mid v \in V(T) \setminus \{r\}\}$.

Let $W \subseteq V(G)$ and $H = G \setminus W$. We define certain useful vertices in $V(H)$. We call a vertex $v \in V(H)$, a *nice vertex* if $d_H(v) = 0$ and $d_G(v) = 2$, i.e. both the neighbours of v belong to the set W. Similarly, we call a vertex $v \in V(H)$, a *tent* if $d_H(v) = 0$ and $d_G(v) = 3$. A *feedback vertex set* is a subset $S \subseteq V(G)$ such that $G \setminus S$ is a forest.

Parameterized Complexity. A parameterized problem Π is a subset of $\Gamma^* \times \mathbb{N}$, where Γ is a finite alphabet. An instance of a parameterized problem is a tuple (x,k), where x is a classical problem instance, and k is called the parameter. A central notion in parameterized complexity is *fixed-parameter tractability (FPT)* which means, for a given instance (x,k), decidability in time $f(k) \cdot p(|x|)$, where f is an arbitrary function of k and p is a polynomial in the input size. For more details on parameterized complexity, we refer the reader to the books of Downey and Fellows [10], Flum and Grohe [12], Niedermeier [27], and the more recent book by Cygan et al. [5].

When we say that we branch on a vertex v, we mean that we recursively generate two instances, one where v belongs to the solution, the other where v does not belong to the solution. This is a standard method of exhaustive branching.

Bounded Search Trees. The running time of an algorithm that uses bounded search trees can be analyzed as follows (see, e.g., [5, 10]). Suppose that the algorithm executes a branching rule which has ℓ branching options (each leading to a recursive call with the corresponding parameter value), such that, in the i^{th} branch option, the current value of the parameter decreases by b_i. Then, $(b_1, b_2, \ldots, b_\ell)$ is called the *branching vector* of this rule. We say that α is the *root* of $(b_1, b_2, \ldots, b_\ell)$ if it is the (unique) positive real root of $x^{b^*} = x^{b^*-b_1} + x^{b^*-b_2} + \cdots + x^{b^*-b_\ell}$, where $b^* = \max\{b_1, b_2, \ldots, b_\ell\}$. If $r > 0$ is the initial value of the parameter, and the algorithm (a) returns a result when (or before) the parameter is negative, and (b) only executes branching rules whose roots are bounded by a constant $c > 0$, then its running time is bounded by $\mathcal{O}^*(c^r)$.

A reduction rule is a polynomial time algorithm that replaces an instance (I,k) of a parameterized language L by a new instance (I',k'). It is said to be *safe* if $(I,k) \in L$ if and only if $(I',k') \in L$.

3 FPT Algorithm for Independent Feedback Vertex Set

In this section we give an FPT algorithm for IFVS running in time $\mathcal{O}^\star(4.1481^k)$. Given an input (G,k), the algorithm starts by computing a feedback vertex set Z in G. A feedback

vertex set in G of size at most k (if it exists) can be computed in time $\mathcal{O}^\star(3.619^k)$ using the algorithm given in [22]. If there is no feedback vertex set of size at most k, then we conclude that (G, k) is a NO instance of ifvs since an ifvs is also a feedback vertex set in G.

We let $H = G \setminus Z$. The algorithm either outputs an ifvs in G of size at most k or correctly conclude that (G, k) is a NO-instance of IFVS. The algorithm guesses a subset $Z' \subseteq Z$, such that for an ifvs X in G, $X \cap Z = Z'$. For each of the guess Z', the algorithm does the following. If $G[Z']$ is not an independent set then it concludes that there is no ifvs X in G such that $Z' \subseteq X$. Otherwise, $G[Z']$ is an independent set. Let $W = Z \setminus Z'$. If $G[W]$ is not a forest, then their is no ifvs X such that, $X \cap Z = Z'$. Therefore, the guess Z' is not correct and the algorithm rejects this guess. Otherwise, it deletes the vertices in Z' and tries to find an ifvs $S \subseteq V(H) \setminus W$ of size at most $k - |Z'|$. Note that any vertex $v \in N_H(Z')$ cannot be part of the solution. Therefore, the algorithm adds the vertices in $N_H(Z')$ to a set $\mathcal{R}$. The set $\mathcal{R}$ consists of those vertices which cannot be included in ifvs in order to maintain the independence of the vertices included in the solution. The algorithm calls the sub-routine Disjoint Independent Feedback Vertex Set (Dis-IFVS) on the instance $(G \setminus Z', W, \mathcal{R}, k - |Z'|)$ to find an ifvs $X \subseteq V(G \setminus Z') \setminus (W \cup \mathcal{R})$. In Section 3.1 we give an algorithm for Dis-IFVS, which given an instance $(G, W, \mathcal{R}, k)$ either finds an ifvs $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ of size at most k or correctly concludes that there does not exits such an ifvs. Moreover, the algorithm for Dis-IFVS runs in time $\mathcal{O}^\star(3.1481^k)$.

▶ **Theorem 1** (restated). *There is an algorithm for* IFVS *running in time* $\mathcal{O}^\star(4.1481^k)$.

Proof. Given an instance (G, k) of IFVS, the algorithm computes a feedback vertex set Z in G of size at most k (if it exists) in time $\mathcal{O}^\star(3.619^k)$. If there is no feedback vertex set of size at most k, it correctly concludes that (G, k) is a NO instance. Otherwise, for each $Z' \subseteq Z$, either it correctly concludes that Z' is a wrong guess (for extending it to an ifvs) or runs the algorithm for Dis-IFVS on the instance $(G \setminus Z', W, \mathcal{R}, k - |Z'|)$. Here, the instance $(G \setminus Z', W, \mathcal{R}, k - |Z'|)$ is created as described above in the description of the algorithm. The correctness of the algorithm follows from the correctness of the algorithm for Dis-IFVS and the fact that all possible intersections of the solution with Z are considered. The running time of the algorithm is given by the following equation: $3.619^k \cdot n^{\mathcal{O}(1)} + \sum_{i=0}^{k} \binom{k}{i} \cdot 3.1481^{k-i} \cdot n^{\mathcal{O}(1)} \leq 4.1481^k \cdot n^{\mathcal{O}(1)}$. This concludes the proof. ◀

3.1 Algorithm for Disjoint Independent Feedback Vertex Set

We give an algorithm for Disjoint Independent Feedback Vertex Set running in time $\mathcal{O}^\star(3.1481^k)$.

Disjoint Independent Feedback Vertex Set (Dis-IFVS) **Parameter:** k
Input: An undirected (multi) graph G, a fvs W in G, $\mathcal{R} \subseteq V(G) \setminus W$ and, an integer k.
Question: Does G have an ifvs $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ such that $\lvert S \rvert \leq k$?

The algorithm for Dis-IFVS either applies some reduction rules or branches on a vertex in $V(G) \setminus W$. The algorithm branches on a vertex in $V(G) \setminus W$ only when (a) none of the reduction rules are applicable; and (b) we are not in the case where we can solve the problem in polynomial time. Let $H = G \setminus W$. We arbitrary root the trees in H at some vertex (preferably a vertex v with $d_H(v) > 2$). We will be using the following measure μ associated with the instance $(G, W, \mathcal{R}, k)$ to bound the number of nodes of the search tree.

$$\mu = \mu(G, W, \mathcal{R}, k) = 2k + \rho(W) - (\eta + 2\tau)\,.$$

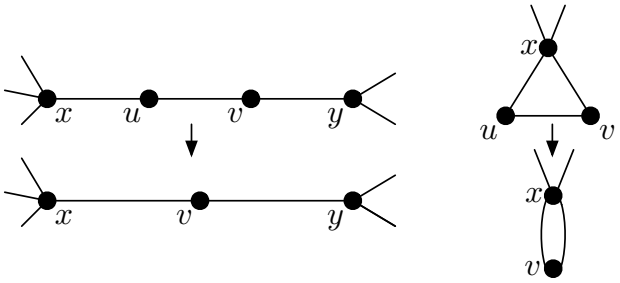


Figure 1 Reduction Rule 2.

Here, $\rho(W)$ is the number of components in W, η denotes the number of nice vertices in $V(H) \setminus \mathcal{R}$ and τ denotes the number of tents in $V(H) \setminus \mathcal{R}$. We note that here nice vertices and tents are defined with respect to the set W. See preliminaries for the definitions of a nice vertex and a tent.

Now we describe all the reduction rules that will be used by the algorithm. The first two reduction rules get rid of vertices of degree at most one and consecutive vertices of degree two in the graph. The safeness of these reduction rules follow from [24].

- Reduction Rule 1. Delete vertices of degree at most one since they do not participate in any cycle.
- Reduction Rule 2. Let u, v be two adjacent degree two vertices in the input graph G which are not nice in H, and x, y be the other neighbors of u, v respectively. Delete the vertex u and add the edge (x, v). Here, if one of u, v belongs to $\mathcal{R}$, say $v \in \mathcal{R}$ then we delete v and add an edge between its neighbors (see Figure 1).
  When applying Reduction Rule 2, if both the degree two vertices belong to $\mathcal{R}$, then the choice of deleting one of them and adding an edge between its neighbors is arbitrary. Observe that the measure μ does not increase after the application of Reduction Rules 1 and 2.
- Reduction Rule 3. If $k < 0$, then return that $(G, W, \mathcal{R}, k)$ is a NO instance.
- Reduction Rule 4. If there is a vertex $v \in \mathcal{R}$ such that v has two neighbors in the same component of W, then return that $(G, W, \mathcal{R}, k)$ is a NO instance.
- Reduction Rule 5. If there is a vertex $v \in \mathcal{R}$ such that v has a neighbor in W, then remove v from $\mathcal{R}$ and add v to W. That is, we solve the instance $(G, W \cup \{v\}, \mathcal{R} \setminus \{v\}, k)$. Observe that by moving v to W we do not increase the number of components of $G[W \cup \{v\}]$.
- Reduction Rule 6. If there is a vertex $v \in V(H) \setminus \mathcal{R}$ such that v has at least two neighbors in the same component of W, then remove v from G and add the vertices in $N_H(v)$ to $\mathcal{R}$. That is, the resulting instance is $(G \setminus \{v\}, W, \mathcal{R} \cup N_H(v), k-1)$. In this case it is easy to observe that v must belong to any ifvs.
- Reduction Rule 7. If there is a vertex $u \in \mathcal{R}$ such that there is a leaf v in H adjacent to u and $d_W(v) \leq 2$. Then remove u from $\mathcal{R}$ and include u in W i.e. the resulting instance is $(G, W \cup \{u\}, \mathcal{R} \setminus \{u\}, k)$. Observe that moving u to W increases the number of components in W, but it also makes v either a nice vertex or a tent.
- Reduction Rule 8. Let T be a tree in H and $u \in V(T) \cap (V(H) \setminus \mathcal{R})$ such that the tree, T_u, rooted at u is a star. That is, all the children of u are leaves of T. Furthermore, each vertex in $V_u = V(T_u) \setminus \{u\}$ (all the children of u) has exactly one neighbor in W and $1 \leq |V_u| \leq 2$. Finally, assume that either $V(T) \setminus V(T_u) = \emptyset$ or the parent x of u is in $\mathcal{R}$. Then include the vertices in $V_u \cup \{x\}$ to W and remove x from $\mathcal{R}$ (if it exists) i.e. the

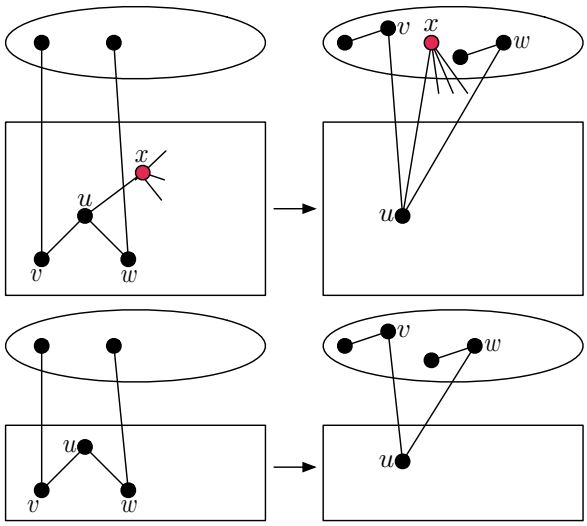


Figure 2 Illustration of Reduction Rule 8.

resulting instance is $(G, W \cup V_u \cup \{x\}, \mathcal{R} \setminus \{x\}, k)$. Here, $\{x\} = \emptyset$, if x does not exists. (see Figure 2)

▶ **Lemma 4.** *Reduction Rule 4 is safe.*

Proof. Let x, y be two neighbors of $v \in \mathcal{R}$ that are present in the same component of W. Since x, y belong to the same component of W, there is a path P in W from x to y. But then, $G[V(P) \cup \{v\}]$ contains a cycle, with v being the only vertex not in W. Therefore, there cannot exist an ifvs, $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ in G. This concludes the proof. ◀

▶ **Lemma 5.** *Reduction Rule 5 is safe. Furthermore, the measure μ does not increase after application of Reduction Rule 5.*

Proof. Let $v \in \mathcal{R}$ be a vertex such that v has a neighbor in W. Note that any ifvs, $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ in G does not contain v. Moreover, since v has a neighbor in W, adding v to W does not increase the number of components in W. This implies that μ does not increase. ◀

▶ **Lemma 6.** *Reduction Rule 6 is safe. Furthermore, the measure μ does not increase after application of Reduction Rule 6.*

Proof. Let $v \in V(H) \setminus \mathcal{R}$ be a vertex such that v has 2 neighbors, say x, y, in the same component of W. Since x, y belong to the same component of W, there is a path P in W from x to y. But then, $G[V(P) \cup \{v\}]$ contains a cycle, with v being the only vertex not in W. Therefore, any ifvs $S \subseteq W$ must include v and hence avoid $N_H(v)$.

When we delete v from G and decrease k by 1, the number of components in W remains the same. If v was either a nice vertex or a tent then $\eta + 2\tau$ can decrease at most by 2. Therefore, the measure μ in the resulting instance can not increase. This concludes the proof. ◀

▶ **Lemma 7.** *Reduction Rule 7 is safe and the measure μ does not increase after its application.*

Proof. Let $u \in \mathcal{R}$ be a vertex such that there is a leaf v in H adjacent to u such that $d_W(v) \leq 2$. Observe that no solution to Dis-IFVS can contain u. Therefore, we only need to show that the measure μ does not increase. When we add u to W, the number of components in W can increase by 1. But then v becomes either a nice vertex or a tent. Therefore, $\eta + 2\tau$ decreases at least by 1. This together with the fact that k remains the same imply that μ cannot increase. ◀

▶ **Lemma 8.** *Reduction Rule 8 is safe and the measure μ does not increase after its application.*

Proof. Let T be a tree in H and $u \in V(T) \cap (V(H) \setminus \mathcal{R})$ such that the tree, T_u, rooted at u is a star. That is, all the children of u are leaves of T. Furthermore, the vertices in $V_u = V(T_u) \setminus \{u\}$ (all the children of u) have exactly one neighbor in W and $1 \leq |V_u| \leq 2$. Also, either $V(T) \setminus V(T_u) = \emptyset$ or the parent x of u is in $\mathcal{R}$. To prove the lemma, we will show that if $(G, W, \mathcal{R}, k)$ is a YES instance of Dis-IFVS then there is an ifvs, $S \subseteq V(H) \setminus (W \cup \mathcal{R})$, of size at most k in G such that $S \cap V_u = \emptyset$. Observe that x (if it exists) cannot belong to S.

Let $S \subseteq V(H) \setminus (W \cup \mathcal{R})$ be an ifvs in G of size at most k. If $S \cap V_u = \emptyset$ then S is the desired solution. Otherwise, let $S' = (S \setminus V_u) \cup \{u\}$. Since $S \cap V_u \neq \emptyset$, we have that u does not belong to S and thus the size of S' is also at most k. We claim that S' is an ifvs of the desired form. Notice that $S' \subseteq V(H) \setminus (W \cup \mathcal{R})$ holds. Also, S' is an independent set since neighbors of u do not belong to S' and $S \setminus V_u$ is an independent set. Therefore, we only need to prove that S' is a feedback vertex set in G. Suppose not, then there is a cycle C in $G \setminus S'$. If C does not contain any vertex from $V_u \cup \{x\}$, then C is also a cycle in $G \setminus S$, contradicting that S in an ifvs in G. If C contains x, but does not contain any other vertex from V_u, then we can conclude that C is a cycle in $G \setminus S$, since $x \notin S$. Otherwise, C contain a vertex say, $v \in V_u$. Note that v is a degree 2 vertex in G. This implies that any cycle containing v must contain both the neighbors of v. But then u belongs to C contradicting that C is a cycle in $G \setminus S'$. This proves the safeness of the reduction rule.

When we add $V_u \cup \{x\}$ to W the number of components can increase at most by 1. Note that none of the vertices in $V_u \cup \{x\}$ is a tent. Therefore, the number of nice vertices or tents does not decrease and u becomes a nice vertex or a tent. This implies that the measure μ does not increase. This concludes the proof. ◀

Algorithm Description. We give an algorithm only for the decision variant of the problem. It is straightforward to modify the algorithm so that it actually finds a solution, provided there exists one.

We will follow a branching strategy with a nontrivial measure function. Let $(G, W, \mathcal{R}, k)$ be the input instance. The algorithm first applies Reduction Rules 1–8, in this order, exhaustively. That is, at any point of time we apply the lowest numbered applicable Reduction Rule. For clarity we denote the reduced instance (the one on which Reduction Rules 1–8 do not apply) by $(G, W, \mathcal{R}, k)$.

We now check whether every vertex in $V(G) \setminus (W \cup \mathcal{R})$ is either a nice vertex or a tent. If this is the case, then in polynomial time we can check whether or not there is an ifvs contained in $V(G) \setminus (W \cup \mathcal{R})$ that is of size at most k; and return accordingly as described by Lemma 9.

▶ **Lemma 9.** *Let (G, X) be an instance of* IFVS *where every vertex in $V(G) \setminus X$ is either a nice vertex or a tent. Then in polynomial time we can find a minimum sized ifvs $S \subseteq V(G) \setminus X$ in G.*

The proof of Lemma 9 follows from Lemma 4.10 in [5], which is based on a polynomial time algorithm for FVS in subcubic graphs by Ueno et at. [32] and the fact that the algorithm described in [5] for finding feedback vertex set on the instances of described type always returns an independent feedback vertex set (if it exists).

Finally, we move to the branching step of the algorithm. We never *branch on a nice vertex or a tent*. We will branch on the vertices in $V(H) \setminus \mathcal{R}$ based on certain criteria. We consider the following three scenarios.

- **Scenario A.** There is a vertex which in not a *tent* and has at least 3 neighbors in W.
- **Scenario B.** There is a leaf which is not a *nice vertex* and has exactly 2 neighbors in W, but no leaf has more than 2 neighbors in W.
- **Scenario C.** All the leaves have exactly one neighbor in W.

Scenario A. If there is a vertex $v \in V(H)$ which is not a *tent* and has at least 3 neighbors in W. Note that $v \notin \mathcal{R}$ as the Reduction Rule 5 is not applicable. In this case we branch on v as follows.

- When v belongs to the solution, then all the vertices in $N_H(v)$ cannot belong to the solution. Therefore, we add all the vertices in $N_H(v)$ to the set $\mathcal{R}$. The resulting instance is $(G \setminus \{v\}, W, \mathcal{R} \cup N(v), k-1)$. In this case k decreases by 1 and $\rho(W), \eta, \tau$ remains the same. Therefore, μ decreases by 2.
- When v does not belong to the solution, then we add v to W. The resulting instance is $(G, W \cup \{v\}, \mathcal{R}, k)$. Note that v cannot have two neighbors in the same component of W, otherwise Reduction Rule 6 would be applicable. Therefore, $G[W \cup \{v\}]$ has at most $\rho(W) - 2$ components. Also, k does not change and η, τ does not decrease. Therefore, μ decreases at least by 2.

The resulting branching vector for this case is $(2, 2)$. When none of the Reduction Rules are applicable and we cannot branch according to Scenario A, then we can assume that there is no vertex $v \in V(H)$, such that v has more than 2 neighbors in W. Of course a tent could have three neighbors in W but as stated before we *never* branch on a nice vertex or a tent. For each tree T (a component) in H, for a vertex $v \in V(T)$ we define the level $\ell(v)$ of v to be the distance of v from the root of T. The root r in a tree has $\ell(r) = 0$. We call a leaf vertex $v \in V(T)$ as a *deep leaf* if $\ell(v) \neq 0$ and for all leaves $v' \in V(T)$, $\ell(v') \leq \ell(v)$.

Scenario B. Let v be a leaf in some tree T in H with the unique neighbor $u \in V(H)$ such that v has exactly two neighbors in W. Observe that $u \notin \mathcal{R}$ since Reduction Rule 7 is not applicable. We branch on u as follows.

- When u belongs to the solution, then all the vertices in $N_H(u)$ cannot belong to the solution. We add all the vertices in $N_H(u) \setminus \{v\}$ to the set $\mathcal{R}$. We add the vertex v to W. The resulting instance is $(G \setminus \{u\}, W \cup \{v\}, \mathcal{R} \cup (N_H(u) \setminus \{v\}), k-1)$. In this case k decreases by 1, and η, τ do not decrease. The number of components in $G[W \cup \{v\}]$ is $\rho(W) - 1$, since v has 2 neighbors in different components of W. Therefore, μ decreases by 3.
- When u does not belong to the solution, then we add u to W. The resulting instance is $(G, W \cup \{u\}, \mathcal{R}, k)$. Note that when we add u to W then v becomes a *tent*. The number of components in $G[W \cup \{u\}]$ is at most $\rho(W) + 1$. Note that k does not increase, η does not decrease and τ increases at least by 1. Therefore, μ decreases by at least 1.

The resulting branching vector for this case is $(3, 1)$.

We now assume that all the leaves in H have exactly one neighbor in W.

Scenario C. Let v be a *deep leaf* in some tree T in H. Let the unique neighbor of v in H be u. We note that the sub-tree T_u rooted at u is a star, i.e. u is the only vertex in T_u which can possibly have degree more than one. This follows from the fact that v is a *deep leaf*. Also, $u \notin \mathcal{R}$ since Reduction Rule 7 is not applicable. We consider the following cases depending on the number of leaves in the sub-tree T_u rooted at u.

Case 1. If T_u has at least two more leaves, say x, y other than v. We branch on the vertex u as follows.

- When u belongs to the solution, then the vertices in $N_H(u)$ does not belong to the solution. We add all the vertices in $N_H(u)$ to the set $\mathcal{R}$. The resulting instance is $(G \setminus \{u\}, W, \mathcal{R} \cup N_H(u), k-1)$. In this case k decreases by 1 and $\eta, \tau, \rho(W)$ does not change. Therefore, μ decreases at least by 2.
- When u does not belong to the solution, we add u to W. The resulting instance is $(G, W \cup \{u\}, \mathcal{R}, k)$. Observe that when we add u to W then, v, x, y becomes nice vertices and the number of components in $G[W \cup \{u\}]$ is at most $\rho(W)+1$. Therefore, μ decreases at least by 2.

The resulting branching vector for this case is $(2, 2)$.

Case 2. If T_u has at most one more leaf other than v. We let x to be the parent of u in T. Note that x exists and $x \notin \mathcal{R}$ because each leaf has exactly one neighbor in W and Reduction Rules 2 and 8 are not applicable. In this case we branch on x.

- When x belongs to the solution, then the vertices in $N_H(x)$ do not belong to the solution. We add all the vertices in $N_H(x) \setminus \{u\}$ to the set $\mathcal{R}$ and add u to the set W. The resulting instance is $(G \setminus \{x\}, W \cup \{u\}, \mathcal{R} \cup (N_H(x) \setminus \{u\}), k-1)$. Observe that Reduction Rule 2 is not applicable. Therefore, at least one of the following holds.
  - u has a neighbor in W.
  - u has one more leaf v' (not in W') adjacent to it in other than v.

  In the former case, when we add u to W, the number of components in $G[W \cup \{u\}]$ is at most $\rho(W)$. Also, v becomes a *nice vertex*. Therefore, η increases at least by 1 and τ does not decrease. Therefore, μ decreases at least by 3. In the latter case when we add u to W, v, v' becomes *nice vertices*. In this case k decreases by 1, η increases by 2, τ does not decrease, and $\rho(W)$ can increase at most by 1. Therefore, μ decreases at least by 3.
- When x does not belong to the solution, we add x to W. But then T_u is a star and u does not have a parent. Therefore, we can apply the Reduction Rule 8. That is, we can add v, v' to W. The resulting instance would be $(G, W \cup \{x, v, v'\}, \mathcal{R}, k)$. Observe that u becomes a *tent*. In this case k, ρ remains the same, while τ increases by 1 and $\rho(W)$ can increase at most by 1. Therefore, μ decreases at least by 1.

The resulting branching vector for this case is $(3, 1)$.

This completes the description of the algorithm.

Analysis and Correctness of the Algorithm. The following Lemma which will be used to prove the correctness of the algorithm.

▶ **Lemma 10.** *For an instance $I = (G, W, \mathcal{R}, k)$ of* Dis-IFVS*, if $\mu < 0$, then I is a NO instance.*

Proof. Let us assume for contradiction that I is a YES instance and $\mu < 0$. Let $S \subseteq V(G) \setminus (W \cup \mathcal{R})$ be an ifvs in G of size at most k. Therefore, $F = G \setminus S$ is a forest. Let $N \subseteq V(G) \setminus (W \cup \mathcal{R})$, $T \subseteq V(G) \setminus (W \cup \mathcal{R})$ be the set of nice vertices and tents in

Table 1 The branch vectors and the corresponding running times.

Scenario	Cases	Branch Vector	c^{μ}
Scenario A		$(2, 2)$	1.4142^{μ}
Scenario B		$(3, 1)$	1.4656^{μ}
Scenario C	Case 1	$(2, 2)$	1.4142^{μ}
	Case 2	$(3, 1)$	1.4656^{μ}

$V(G) \setminus (W \cup \mathcal{R})$, respectively. Since F is a forest we have that $G' = G[(W \cup N \cup T) \setminus S]$ is a forest. In G', we contract each of the components in W to a single vertex to obtain a forest $\tilde{F}$. Observe that $\tilde{F}$ has at most $|V(\tilde{F})| \leq \rho(W) + |N \setminus S| + |T \setminus S|$ vertices and thus can have at most $\rho(W) + |N \setminus S| + |T \setminus S| - 1$ many edges. The vertices in $(N \cup T) \setminus S \subseteq V(G) \setminus (W \cup \mathcal{R})$ forms an independent set in $\tilde{F}$, since they are nice vertices or tents. The vertices in $N \setminus S$ and $T \setminus S$ have degree 2 and degree 3 in $\tilde{F}$, respectively, since their degree cannot drop while contracting the components of $G[W]$. This implies that,

$$2|N \setminus S| + 3|T \setminus S| \quad \leq |E(\tilde{F})| \leq \quad \rho(W) + |N \setminus S| + |T \setminus S| - 1.$$

Therefore, $|N \setminus S| + 2|T \setminus S| < \rho(W)$. But $N \cap T = \emptyset$ and thus

$$|N| + 2|T| < \rho(W) + 2|S| \leq \rho(W) + 2k. \tag{1}$$

However, by our assumption, $\mu(I) = \rho(W) + 2k - (|N| + 2|T|) < 0$ and thus $|N| + 2|T| > \rho(W) + k$. This, contradicts the inequality given in Equation 1 contradicting our assumption that I is a YES instance. ◀

▶ **Lemma 11.** *The algorithm presented for* Dis-IFVS *is correct.*

Proof. Let $I = (G, W, \mathcal{R}, k)$ be an instance of Dis-IFVS. We prove the correctness of the algorithm by induction on $\mu = \mu(I) = 2k + \rho(W) - (\eta + 2\tau)$. The base case occurs in one of the following cases.

- $\mu < 0$. By Lemma 10, when $\mu < 0$, we can correctly conclude that I is a NO instance.
- $k < 0$. By Reduction Rule 3 it follows that when $k < 0$, we can correctly conclude that I is a NO instance.
- When none of the Reduction Rules and Branching Rules are applicable. In this case we are able to solve the instance in polynomial time.

By induction hypothesis we assume that for all $\mu \leq l$, the algorithm is correct. We will now prove that the algorithm is correct when $\mu = l + 1$. The algorithm does one of the following. Either applies one of the Reduction Rules if applicable. By Lemma 4 to Lemma 8 we know that the Reduction Rules correctly concludes that I is a NO instance or produces an equivalent instance I' with $\mu(I') \leq \mu(I)$. If $\mu(I') < \mu(I)$, then by induction hypothesis and safeness of the Reduction Rules the algorithm correctly decides if I is a yes instance or not. Otherwise, $\mu(I') = \mu(I)$. If none of the Reduction Rules are applicable then the algorithm applies one of the Branching Rules. Branching Rules are exhaustive and covers all possible cases. Furthermore, μ decreases in each of the branch by at least one. Therefore, by the induction hypothesis, the algorithm correctly decides whether I is a YES instance or not. ◀

▶ **Theorem 12.** *The algorithm presented solves* Disjoint Independent Feedback Vertex Set *in time* $\mathcal{O}^{\star}(3.1481^{k})$.

Proof. The Reduction Rules 1 to 8 can be applied in time polynomial in the input size. Also, at each of the branch we spend a polynomial amount of time. At each of the recursive calls in a branch, the measure μ decreases at least by 1. When $\mu \leq 0$, then we are able to solve the remaining instance in polynomial time or correctly conclude that the corresponding branch cannot lead to a solution. At the start of the algorithm $\mu \leq 3k$. The worst-case branching vector for the algorithm is $(3, 1)$ (see Table 1). The recurrence for the worst case branching vector is:

$$T(\mu) \leq T(\mu - 3) + T(\mu - 1)\,.$$

The running time corresponding to the above recurrence relation is 3.1481^k. ◀

3.2 A family of counter examples to Song's Algorithm for Independent Feedback Vertex Set

Let $\mathcal{F}$ be the family of even cycles. For any $C \in \mathcal{F}$, let (C_W, C_H) be a bipartition of C. Given a graph G and a feedback vertex set F in G, Lemma 3.1 of [30] claims to output a minimum IFVS in G. But for $G = C$ and $F = C_W$, where $C \in \mathcal{F}$, the algorithm of Lemma 3.1 always returns $\emptyset$.

4 Conclusion

In this paper we studied the Independent Feedback Vertex Set problem in the realm of parameterized algorithms, moderately exponential time algorithms and combinatorial upper bounds. We gave the fastest known deterministic algorithms for the problem running in times $\mathcal{O}^*(4.1481^k)$ and $\mathcal{O}^*(1.5981^n)$, respectively. Finally, we showed that the number of minimal ifvses in any graph on n vertices is upper bounded by 1.7485^n. We also complemented the upper bound result by obtaining a family of graphs where the number of minimal ifvses is at least $3^{n/3}$. Improving running time of all our algorithms is an interesting question. We conclude the paper with few concrete open problems.

- Does Independent Feedback Vertex Set admit a kernel of size $\mathcal{O}(k^2)$?
- Could we close the gap (or even bring closer) between the upper bound and the lower bounds on the number of minimal ifvses in any graph on n vertices?

Acknowledgements. We thank Amer E. Mouawad and Prafullkumar P. Tale for discussions and help in programming.

References

1 Akanksha Agrawal, Daniel Lokshtanov, Amer E. Mouawad, and Saket Saurabh. Simultaneous Feedback Vertex Set: A Parameterized Perspective. In *Proc. of the 33rd Symposium on Theoretical Aspects of Computer Science (STACS'16)*, volume 47 of *LIPIcs*, pages 7:1–7:15. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.7.

2 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999.

3 Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set: New measure and new structures. *Algorithmica*, 73(1):63–86, 2015.

4 Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *Journal of Computer and System Sciences*, 74(7):1188–1198, 2008.

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

6 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 150–159, 2011.

7 Marek Cygan, Marcin Pilipczuk, and Michal Pilipczuk. On group feedback vertex set parameterized by the size of the cutset. *Algorithmica*, 74(2):630–642, 2016.

8 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. *SIAM Journal of Discrete Mathematics*, 27(1):290–309, 2013.

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

10 Rod G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer-Verlag, 1997.

11 Paola Festa, Panos M. Pardalos, and Mauricio G. C. Resende. Feedback set problems. In *Handbook of combinatorial optimization, Supplement Vol. A*, pages 209–258. Kluwer Acad. Publ., Dordrecht, 1999.

12 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

13 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC'16)*, pages 764–775, 2016.

14 Fedor V. Fomin, Serge Gaspers, Artem V. Pyatkin, and Igor Razgon. On the minimum feedback vertex set problem: Exact and enumeration algorithms. *Algorithmica*, 52(2):293–307, 2008.

15 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A Measure & Conquer approach for the analysis of exact algorithms. *Journal of ACM*, 56(5), 2009.

16 Fedor V. Fomin, Pinar Heggernes, Dieter Kratsch, Charis Papadopoulos, and Yngve Villanger. Enumerating minimal subset feedback vertex sets. *Algorithmica*, 69(1):216–231, 2014.

17 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer, 2010. An EATCS Series: Texts in Theoretical Computer Science.

18 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM Journal on Computing*, 44(1):54–87, 2015.

19 G. R. Grimmett. An upper bound for the number of spanning trees of a graph. *Discrete Mathematics*, 16(4):323–324, 1976.

20 Sylvain Guillemot. FPT algorithms for path-transversal and cycle-transversal problems. *Discrete Optimization*, 8(1):61–71, 2011.

21 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Plenum Press, New York, 1972.

22 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Information Processing Letters*, 114(10):556–560, 2014.

23 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. Linear time parameterized algorithms for subset feedback vertex set. In *Automata, Languages, and Programming – 42nd International Colloquium, ICALP*, volume 9134, pages 935–946, 2015.

24 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, and Saket Saurabh. On parameterized independent feedback vertex set. *Theoretical Computer Science*, 461:65–75, 2012.

25 Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh, and Somnath Sikdar. FPT algorithms for connected feedback vertex set. *Journal of Combinatorial Optimization*, 24(2):131–146, 2012.

26 J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28, 1965.

27 Rolf Niedermeier. Invitation to fixed-parameter algorithms, 2006.

28 Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for finding feedback vertex sets. *ACM Transactions on Algorithms*, 2(3):403–415, 2006.

29 Igor Razgon. Exact computation of maximum induced forest. In *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006)*, volume 4059, pages 160–171, 2006.

30 Yinglei Song. An improved parameterized algorithm for the independent feedback vertex set problem. *Theoretical Computer Science*, 535:25–30, 2014.

31 Yuma Tamura, Takehiro Ito, and Xiao Zhou. Algorithms for the independent feedback vertex set problem. *IEICE Transactions*, 98-A(6):1179–1188, 2015.

32 Shuichi Ueno, Yoji Kajitani, and Shin'ya Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Mathematics*, 72(1):355–360, 1988.

33 Magnus Wahlström. Half-integrality, LP-branching and FPT algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1762–1781, 2014.

H-Free Graphs, Independent Sets, and Subexponential-Time Algorithms*

Gábor Bacsó[1], Dániel Marx[2], and Zsolt Tuza[3]

1 **Institute for Computer Science and Control, Hungarian Academy of Sciences, Budapest, Hungary**
2 **Institute for Computer Science and Control, Hungarian Academy of Sciences, Budapest, Hungary**
3 **Alfréd Rényi Institute of Mathematics, Budapest, Hungary; and Department of Computer Science and Systems Technology, University of Pannonia, Veszprém, Hungary**

Abstract

It is an old open question in algorithmic graph theory to determine the complexity of the Maximum Independent Set problem on P_t-free graphs, that is, on graphs not containing any induced path on t vertices. So far, polynomial-time algorithms are known only for $t \leq 5$ [Lokshtanov et al., SODA 2014, pp. 570–581, 2014]. Here we study the existence of subexponential-time algorithms for the problem: by generalizing an earlier result of Randerath and Schiermeyer for $t = 5$ [*Discrete Appl. Math.,* 158 (2010), pp. 1041–1044], we show that for any $t \geq 5$, there is an algorithm for Maximum Independent Set on P_t-free graphs whose running time is subexponential in the number of vertices.

Scattered Set is the generalization of Maximum Independent Set where the vertices of the solution are required to be at distance at least d from each other. We give a complete characterization of those graphs H for which d-Scattered Set on H-free graphs can be solved in time subexponential in the *size of the input* (that is, in the number of vertices plus number of edges):

- If every component of H is a path, then d-Scattered Set on H-free graphs with n vertices and m edges can be solved in time $2^{(n+m)^{1-O(1/|V(H)|)}}$, even if d is part of the input.
- Otherwise, assuming ETH, there is no $2^{o(n+m)}$-time algorithm for d-Scattered Set for any fixed $d \geq 3$ on H-free graphs with n-vertices and m-edges.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases independent set, scattered set, subexponential algorithms, H-free graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.3

1 Introduction

The Maximum Independent Set problem (MIS, for short) is one of the fundamental problems in discrete optimization. It takes a graph G as input, and asks for the maximum number $\alpha(G)$ of mutually nonadjacent (i.e., independent) vertices in G. On unrestricted input, it is not only NP-hard (its decision version "Is $\alpha(G) \geq k$?" being NP-complete), but

* Research of Gábor Bacsó and Dániel Marx was supported by ERC Starting Grant PARAMTIGHT (No. 280152) and OTKA grant NK105645. Research of Zsolt Tuza was supported by the National Research, Development and Innovation Office – NKFIH under the grant SNN 116095.

APX-hard as well, and, in fact, even not approximable within $O(n^{1-\varepsilon})$ in polynomial time for any $\varepsilon > 0$ unless P=NP, as proved by Zuckerman [21]. For this reason, classes of graphs are of definite interest on which MIS becomes tractable. One direction of this area is to study the complexity of MIS on *H-free graphs*, that means graphs not containing any *induced* subgraph isomorphic to a given graph H.

What do we know about the complexity of MIS on H-free graphs? One the hardness side, it is easy to see that if G' is obtained from G by subdividing each edge with $2t$ new vertices, then $\alpha(G') = \alpha(G) + t|E(G)|$ holds. This can be used to show that MIS is NP-hard on H-free graphs whenever H is not a forest, and also if H contains a tree component with at least two vertices of degree larger than 2 (first observed in [2], see, e.g., [11]). As MIS is known to be NP-hard on graphs of maximum degree at most 3, the case when H contains a vertex of degree at least 4 is also NP-hard.

The only case not covered by the above observations is when every component of H is either a path, or a tree with exactly one degree-3 vertex c with three paths of arbitrary lengths starting from c. Even this collection means infinitely many cases. For decades, on these graphs H only partial results have been obtained, proving polynomial-time solvability in some cases. A classical algorithm of Minty [16] and its corrected form by Sbihi [19] solved the problem when H is a claw (3 paths of length 1 in the model above). This happened in 1980. Much later, in 2004, Alekseev [3] generalized this result by an algorithm for H isomorphic to a fork (2 paths of length 1 and one path of length 2).

Somewhat embarrassingly, even the seemingly easy case of P_t-free graphs is poorly understood (where P_t is the path on t vertices). MIS on P_t-free graphs is not known to be NP-hard for any t; for all we know, it could be polynomial-time solvable for every fixed $t \geq 1$. P_4-free graphs (also known as cographs) have very simple structure, which can be used to solve MIS in way that is very simple, but does not generalize to P_t-free graphs for larger t. In 2010, it was a breakthrough when Randerath and Schiermeyer [17] stated that MIS was solvable in subexponential time, more precisely within $O(C^{n^{1-\varepsilon}})$ for any constants $C > 1$ and $\varepsilon < 1/4$, on P_5-free graphs. Designing an algorithm based on deep results, Lokshtanov [11] finally proved that MIS is polynomial-time solvable on P_5-free graphs. More recently, a *quasipolynomial* ($n^{\log^{O(1)} n}$-time) algorithm was found for P_6-free graphs [13].

In this paper, we explore MIS and some variants on H-free graphs from the viewpoint of *subexponential-time algorithms.* That is, instead of aiming for algorithms with running time $n^{O(1)}$ on n-vertex graphs, we ask if $2^{o(n)}$ algorithms are possible. Our first result shows that there is indeed such an algorithm for P_t-free graphs.

▶ **Theorem 1.** *For every fixed $t \geq 5$,* MIS *on n-vertex P_t-free graphs is subexponential, namely, it can be solved by a $2^{O(n^{1-1/\lfloor t/2 \rfloor + o(1)})}$-time algorithm.*

In particular, for $t = 5$, this improves the result of Randerath and Schiermeyer [17]. The algorithm is based on the obsevation that a connected P_t-free graph always has a high-degree vertex, which can be used for efficient branching. However, the algorithm does not seem to be extendable to H-free graphs where H is the subdivision of a $K_{1,3}$, hence the existence of subexponential-time algorithms on such graphs remains an open question.

Scattered Set (also known under other names such as dispersion or distance-d independent set [14, 20, 1, 18, 6, 9]) is the natural generalization of MIS where the vertices of the solution are required to be at distance at least d from each other; the size of the largest such set will be denoted by $\alpha_d(G)$. We can consider with d being part of the input, or assume that $d \geq 2$ is a fixed constant, in which case we call it d-Scattered Set. Clearly, MIS is exactly the same as 2-Scattered Set. Despite its similarity to MIS, the branching algorithm of Theorem 1 cannot be generalized: we give evidence that there is

no subexponential-time algorithm for 3-Scattered Set on P_5-free graphs. For the lower bound, we assume the Exponential-Time Hypothesis (ETH) of Impagliazzo, Paturi, and Zane, which can be informally stated as n-variable 3SAT cannot be solved in $2^{o(n)}$ time (see [7, 12, 10]).

▶ **Theorem 2.** *Assuming ETH, there is no $2^{o(n)}$-time algorithm for d-Scattered Set with $d = 3$ on P_5-free graphs with n vertices.*

In light of the negative result of Theorem 2, we slightly change our objective by aiming for an algorithm that is subexponential in the *size of the input,* that is, in the total number of vertices and edge of the graph G. As the number of edges of G can be up to quadratic in the number of vertices, this is a weaker goal: an algorithm that is subexponential in the number of edges is not necessarily subexponential in the number of vertices. We give a complete characterization when such algorithms are possible for Scattered Set.

▶ **Theorem 3.** *For every fixed graph H, the following holds.*

1. *If every component of H is a path, then d-Scattered Set on H-free graphs with n vertices and m edges can be solved in time $2^{(n+m)^{1-O(1/|V(H)|)}}$, even if d is part of the input.*
2. *Otherwise, assuming ETH, there is no $2^{o(n+m)}$-time algorithm for d-Scattered Set for any fixed $d \geq 3$ on H-free graphs with n-vertices and m-edges.*

The algorithmic side of Theorem 3 is based on the combinatorial observation that the treewidth of P_t-free graphs is sublinear in the number of edges, which means that standard algorithms on bounded-treewidth graphs can be invoked to solve the problem in time subexponential in the number of edges. It has not escaped our notice that this approach is completely generic and could be used for many other problems (e.g., Hamiltonian Cycle, 3-Coloring, ...) where $2^{O(t)} \cdot n^{O(1)}$ or even $2^{t \cdot \log^{O(1)} t} \cdot n^{O(1)}$-time algorithms are known on graphs of treewidth t. For the lower bound part of Theorem 3, we need to examine only two cases: claw-free graphs and C_t-free graphs (where C_t is the cycle on t vertices); the other cases then follow immediately.

The algorithm described in Section 3 implies Theorem 1, while Theorems 2 and 3 are implied by Sections 4 and 5.

2 Preliminaries

This work investigates simple undirected graphs throughout. The vertex set of graph G will be denoted by $V(G)$, the edge set by $E(G)$. When we deal with a fixed graph, we write simply V and E respectively.

A graph is H-free if it does not contain H as an induced subgraph.

A distance-d set (d-scattered) set) in a graph G is a vertex set $S \subseteq V(G)$ such that for every pair of vertices in S, the distance between them is at least d in the graph. For $d = 2$, we obtain the traditional notion of independent set (stable set). For $d > c$, a distance-d set is a distance-c set as well, for example, any distance-d set is independent for $d \geq 2$.

The algorithmic problem Weighted Independent Set is the problem of maximizing the sum of weights in a graph with nonnegative vertex weights w. The maximum is denoted by $\alpha_w(G)$. For a weight w everywhere 1, we obtain the usual problem Independent Set (MIS) with maximum $\alpha(G)$.

Several definitions are used in the literature under the name *subexponential function.* Each of them means some condition: this function (with variable $p > 1$, called the parameter)

may not be larger than some bound, depending on p. Here we use two versions, where the bound is of type $exp(o(p))$ and $exp(p^{1-\epsilon})$ respectively, with some $\epsilon > 0$. (Clearly, the second one is the more strict.) Throughout the paper, we state our results emphasizing, which version we mean.

An algorithm A is *subexponential* in parameter $p > 1$ if the number of steps executed by A is a subexponential function of the parameter p. We will use here this notion for graphs, mostly in the following cases: p is the number n of vertices, the number m of edges, or $p = n + m$ (which is considered to be the size of the input generally).

A problem Π is *subexponential* if there exists some *subexponential* algorithm solving Π.

The notation $d_G(x, y)$ and $\mathrm{diam}(G)$ will have the usual meaning. For a vertex x of G, its *radius* $r_G(x)$ is $\max\{d_G(x, y)|y \in V(G)\}$ and for the *radius* of graph G, $r(G) := \min\{r_G(x)|x \in V(G))\}$. $\Delta(G)$ is the maximal degree in G.

P_t (C_t) is the chordless path (cycle) on t vertices.

3 Algorithm for MIS on P_t-free graphs

The method used here will be similar to that of [17]. There a special dominating set is found (applying [5]), here a vertex of small radius will help. More precisely, the algorithm is based on the observation that a connected P_t-free graph always has a high-degree vertex. The following definition formalizes this property.

▶ **Definition 4.** For a fixed real $\delta > 0$ and a natural number n_0, let $\mathcal{C} := \mathcal{C}(n_0, \delta)$ be the class of graphs G with the following property: For every connected induced subgraph G' of G with $k := |V(G')| \geq n_0$, $\Delta(G') \geq k^\delta$.

Clearly, each class $\mathcal{C} := \mathcal{C}(n_0, \delta)$ is contained in the class of P_t-free graphs for $t = n_0$. But if we extend $\mathcal{C}$, the result below will be stronger than a statement merely for graphs without some long induced path.

▶ **Definition 5.** For a fixed real $\delta > 0$ and a natural number n_0, let $\mathcal{G} := \mathcal{G}(n_0, \delta)$ be the class of graphs G with the following property: For every connected induced subgraph G' of G having maximum degree at least 3, with $k := |V(G')| \geq n_0$, $\Delta(G') \geq k^\delta$.

The following result presents the connection of $P_t$free graphs with the classes above.

▶ **Lemma 6.** *For every $t \geq 5$, every P_t-free graph is in $\mathcal{C}(N_0, \delta)$ (and thus in $\mathcal{G}(N_0, \delta)$ as well) with $\delta = \lfloor t/2 \rfloor^{-1}$ and an appropriate $N_0 = N_0(t)$.*

Proof. Every connected P_t-free graph has radius at most $\mathrm{diam}(G) \leq t - 2$. To obtain stronger constants, we use a result of Erdős, Saks, and Sós [8, Theorem 2.1], which states, in an alternative formulation, that every connected P_t-free graph has radius at most $\lfloor t/2 \rfloor$.[1]

Assuming that G is connected and has maximum degree Δ, the number of vertices at distance i from a vertex c with minimal radius is at most $\Delta \cdot (\Delta - 1)^{i-1}$. Thus, if G is connected, P_t-free, moreover it has n vertices and maximum degree $\Delta = \Delta(G)$, then for any $t \geq 6$, we have

$$n \leq 1 + \Delta \cdot \sum_{i=1}^{r} (\Delta - 1)^{i-1} < \Delta^{\lfloor t/2 \rfloor}, \tag{1}$$

[1] A subset of the present authors [4] established a stronger property which is equivalent to being P_t-free.

Algorithm 1 Algorithm DEGALPHA

Input: a graph G

1. If $|V(G)| = 1$ then $\alpha(G) = 1$.
2. If $|V(G)| > 1$ and G is disconnected:
   a. Determine a connected component G' of G, and set $G'' = G - G'$.
   b. Determine $\alpha(G')$ and $\alpha(G'')$, calling Algorithm DEGALPHA for G' and G'' separately, and write $\alpha(G) = \alpha(G') + \alpha(G'')$.
3. If $|V(G)| > 1$ and G is connected:
   a. Determine a vertex v of maximum degree, $d_G(v) = \Delta(G)$.
   b. $\Delta(G) \leq 2$ then $\alpha(G)$ is the maximal size of independent set in the corresponding path or cycle respectively.
   c. Determine $\alpha(G - v)$ and $\alpha(G - N[v])$ where $N[v]$ is the closed neighborhood of v, calling Algorithm DEGALPHA for $G - v$ and $G - N[v]$ separately, and write $\alpha(G) = \max(\alpha(G - v), \alpha(G - N[v]) + 1)$.

which corresponds to the standard Moore bound (see, e.g., inequality (1) on page 8 of [15]). As a consequence, for $t \geq 6$, we obtain

$$\Delta(G) \geq n^{\lfloor t/2 \rfloor^{-1}} \tag{2}$$

For $t = 5$, we get the slightly weaker bound $n \leq 1 + \Delta + \Delta(\Delta - 1) = \Delta^2 + 1$. However, with additional arguments, we can show that $n \leq \Delta^2$ holds if $\Delta > 2$, thus the statement is true if $n > 5$. (Sketch of the proof: the only way that $n = \Delta^2 + 1$ can hold is when c has exactly Δ neighbors, each of which has exactly $\Delta - 1$ neighbors at distance two from c, and they do not share any of these neighbors. Let u and v be two neighbors of c. If a neighbor $u' \neq c$ of u is nonadjacent to a neighbor $v' \neq c$ of v, then u', u, c, v, v' form an induced P_5. This shows that u' has degree at least $1 + (\Delta - 1)^2$, which is more than Δ if $\Delta > 2$.) ◀

Next we show that subexponential-time algorithms exists for the class $\mathcal{G}(n_0, \delta)$.

▶ Remark. The class $\mathcal{G}(n_0, \delta)$ with appropriate parameters contains non-P_t-free graphs for any t.

▶ **Lemma 7.** *For any fixed real $0 < \delta < 1$ and a natural number n_0, the independent set problem is subexponential (in the strong sense) for the class $\mathcal{G}(n_0, \delta)$, namely, it can be solved by an algorithm executing at most $O(\exp(c(\delta) \cdot n^{1-\delta} \cdot \ln n))$ steps, where $c(\delta)$ is any real constant greater than $\frac{\delta}{1-\delta}$.*

Proof. The conditions lead to a simple exact algorithm solving MIS (see Algorithm 1), which is also the basis for the analysis in [17] (except that here we need not deal with isolated vertices separately) and whose variants also appear in enumeration algorithms for independent sets.

It is a direct consequence of the definitions that Algorithm DEGALPHA properly determines the independence number of G.

Time analysis. We may and will assume that the number n of vertices is larger than a suitably fixed threshold value $n_0 = n_0(\delta)$. Connectivity test and separation of a connected component – as well as the determination of a maximum-degree vertex – can be performed in $O(n^2)$ steps. Therefore, a non-decreasing integer function $f(n)$ surely is a valid upper

bound on the running time of Algorithm 1 on any input graph G on n vertices whenever, for any $n > n_0$ and all integers n' in the range $n/2 \leq n' < n$, we have

$$f(n) \geq kn^2 + f(n') + f(n - n') \tag{3}$$
$$f(n) \geq kn^2 + f(n-1) + f(n - \lceil n^\delta \rceil) \tag{4}$$

where k is a suitably chosen (not large) constant. Throughout this proof, square brackets [ ] will be used as parentheses, with the same meaning as ( ), for making some expressions more transparent.

Note that the time bound in Lemma 7 is superpolynomial, therefore writing f in the form

$$f(n) = g(n) + kn^3/3$$

requires the same growth order for f and g. Let us define

$$g(x) = \exp(h(x))$$

where

$$h(x) = c(\delta) \cdot x^{1-\delta} \cdot \ln x.$$

By the observations above, (3) and (4) will follow if we prove the inequalities

$$g(x) \geq g(x') + g(x - x') \tag{5}$$
$$g(x) \geq g(x-1) + g(x - x^\delta) \tag{6}$$

for every real x large enough and every x' with $x/2 \leq x' \leq x - 1$. We can immediately observe that (5) is a consequence of (6) as

$$g(x) - g(x') \geq g(x) - g(x-1) \geq g(x - x^\delta) \geq g(x/2) \geq g(x - x')$$

if x is large enough with respect to δ, because g is an increasing function and δ is a constant smaller than 1. Therefore only (6) remains to be proved.

We shall need the derivatives of g and h, which can be computed as

$$g'(x) = (\exp[h(x)])' = \exp[h(x)] \cdot h'(x) = g(x) \cdot h'(x)$$

and

$$\begin{aligned} h'(x) &= \left(c(\delta) \cdot x^{1-\delta} \cdot \ln x\right)' \\ &= c(\delta) \cdot x^{-\delta} \cdot [(1-\delta)\ln x + 1]. \end{aligned} \tag{7}$$

It is important to note for later use that

$$h'(x-1) = (1 + o(1)) \cdot h'(x)$$

as $x \to \infty$. Moreover, g and h are increasing, while h' is decreasing, except on a bounded part of the domain.

Next, we apply Cauchy's Mean value theorem in three steps, first for both g and h to estimate $g(x) - g(x-1)$, and second for h to estimate $g(x - x^\delta)$, as follows. For some ξ and ξ' with $x - 1 \leq \xi, \xi' \leq x$ we have

$$\begin{aligned} g(x) - g(x-1) &= g'(\xi) = \exp(h(\xi)) \cdot h'(\xi) \\ &\geq \exp[h(x-1)] \cdot h'(x) \\ &= \exp[h(x) - h'(\xi')] \cdot h'(x) \\ &\geq \exp[h(x) - h'(x-1)] \cdot h'(x) \\ &= \exp\left[h(x) - (1+o(1)) \cdot h'(x)\right] \cdot h'(x). \end{aligned} \tag{8}$$

On the other hand, for some ξ'' with $x - x^\delta \leq \xi'' \leq x$ we have $h(x - x^\delta) = h(x) - x^\delta \cdot h'(\xi'')$, therefore

$$\begin{aligned} g(x - x^\delta) &= \exp[h(x) - x^\delta \cdot h'(\xi'')] \\ &\leq \exp[h(x) - x^\delta \cdot h'(x)]. \end{aligned} \tag{9}$$

Thus, to prove (6), it suffices to show that (8) is not smaller than (9). Taking logarithms this means

$$h(x) - (1 + o(1)) \cdot h'(x) + \ln h'(x) \quad \geq \quad h(x) - x^\delta \cdot h'(x). \tag{10}$$

Or equivalently

$$[x^\delta - 1 - o(1)] \cdot h'(x) \quad \geq \quad -\ln h'(x). \tag{11}$$

Using (7), we obtain that it is enough to prove

$$(c(\delta) + o(1)) \cdot (1 - \delta) \cdot \ln x \quad \geq \quad (\delta + o(1)) \cdot \ln x.$$

This is implied by the condition on $c(\delta)$ (even with strict inequality), completing the proof of the lemma. ◀

Theorem 1 follows immediately from putting together Lemmas 6 and 7.

4 Algorithm for Scattered Set on P_t-free graphs

The algorithm for Scattered Set for P_t-free graphs hinges on the following combinatorial bound.

▶ **Lemma 8.** *For every $t \geq 2$ and for every P_t-free graph with m edges, we have that G has treewidth at most $3m^{1-1/(t+2)}$.*

Proof. Let n be the number of vertices of G. We may ignore components of G that are trees or isolated vertices and hence we can assume that $n \leq m$. We consider two cases. Suppose first that $m \geq n^{1+1/(t+1)}$. Then we have

$$m^{1-1/(t+2)} \geq n^{(1+1/(t+1))(1-1/(t+2))} = n.$$

Obviously, n is an upper bound on the treewidth of G, and hence the claim follows.

Suppose now that $m < n^{1+1/(t+1)}$. Let X be the subset of vertices of G with degree at least $n^{2/(t+1)}$. The degree sum of the vertices in X is at most $2m$, hence we have $|X| \leq 2m/n^{2/(t+1)} < 2n^{1-1/(t+1)}$. By the definition of X, the graph $G - X$ has maximum degree less than $n^{2/(t+1)}$. Thus each component of X is a P_t-free graph with maximum degree less than $n^{2/(t+1)}$ and hence Lemma 6 implies that each component of $G - X$ has at most $n^{(2/(t+1))\lfloor t/2 \rfloor} \leq n^{1-1/(t+1)}$ vertices. In particular, this implies that $G - X$ has treewidth at most $n^{1-1/(t+1)}$. As removing a vertex can decrease treewidth at most by one, it follows that G has treewidth at most $n^{1-1/(t+1)} + |X| = 3n^{1-1/(t+1)} < 3m^{1-1/(t+1)} \leq 3m^{1-1/(t+2)}$. ◀

It is known that Scattered Set can be solved in time $d^{O(w)} \cdot n^{O(1)}$ on graphs of treewidth w using standard dynamic programming techniques (cf. [20, 14]). By Lemma 8, it follows that Scattered Set on P_t-free graphs can be solved in time

$$d^{3m^{1-1/(t+2)}} \cdot n^{O(1)} = 2^{O(m^{1-1/(t+2)} \log m)} = 2^{m^{1-1/(t+2)+o(1)}}$$

(taking into account that we may assume $n = O(m)$ and $d \leq n$). Observe that if every component of H is a path, then H is an induced subgraph of $P_{2|V(H)|}$, which implies that H-free graphs are $P_{2|V(H)|}$-free. Thus the algorithm described here for P_t-free graphs implies the first part of Theorem 3.

5 Lower bounds for Scattered Set

A standard consequence of ETH and the so-called Sparsification Lemma is that there is no subexponential-time algorithm for MIS even on graphs of bounded degree (see, e.g., [7]):

▶ **Theorem 9.** *Assuming ETH, there is no $2^{o(n)}$-time algorithm for* MIS *on n-vertex graphs of maximum degree 3.*

A very simple reduction can reduce MIS to 3-Scattered Set for P_5-free graphs, showing that, assuming ETH, there is no algorithm subexponential in the number of vertices for the latter problem. This proves Theorem 2 stated in the Introduction.

Proof (Theorem 2). Given an n-vertex m-edge graph G with maximum degree 3 and an integer k, we construct a graph G' with $n + m = O(n)$ vertices such that $\alpha(G) = \alpha_3(G')$. This reduction proves that a $2^{o(n)}$-time algorithm for 3-Scattered Set could be used to obtain a $2^{o(n)}$-time algorithm for MIS on graphs of maximum degree 3, and this would violate ETH by Theorem 9.

The graph G' contains one vertex for each vertex of G and additionally one vertex for each edge of G. The m vertices of G' representing the edges of G form a clique. Moreover, if the endpoints of an edge $e \in E(G)$ are $u, v \in V(G)$, then the vertex of G' representing e is connected with the vertices of G' representing u and v. This completes the construction of G'. It is easy to see that G' is P_5-free: an induced path of G' can contain at most two vertices of the clique corresponding to $E(G)$ and the vertices of G' corresponding to the vertices of G form an independent set.

If S is an independent set of G, then we claim that the corresponding vertices of G' are at distance at least 3 from each other. Indeed, no two such vertices have a common neighbor: if $u, v \in S$ and the corresponding two vertices in G' have a common neighbor, then this common neighbor represents an edge e of G whose endpoints are u and v, violating the assumption that S is independent. Conversely, suppose that $S' \subseteq V(G')$ is a set of k vertices with pairwise distance at least 3 in G'. If $k \geq 2$, then all these vertices represent vertices of G: observe that for every edge e of G, the vertex of G' representing e is at distance at most 2 from every other non-isolated vertex of G'. We claim that S' corresponds to an independent set of G. Indeed, if $u, v \in S'$ and there is an edge e in G' with endpoints u and v, then the vertex of G' representing e is a common neighbor of u and v, a contradiction. ◀

Next we give negative results on the existence of algorithms for Scattered Set that have running time subexponential in the number of edges. To rule out such algorithms, we construct instances that have bounded degree: then being subexponential in the number of vertices or the number of edges are the same. We consider first claw-free graphs. The key insight here is that Scattered Set with $d = 3$ in line graphs (which are claw-free) is essentially the Induced Matching problem, for which it is easy to prove hardness results.

▶ **Theorem 10.** *Assuming ETH, d-*Scattered Set *does not have a $2^{o(n)}$ algorithm on n-vertex claw-free graphs of maximum degree 4 for any fixed $d \geq 3$.*

Proof. Given an n-vertex graph G with maximum degree 3, we construct a claw-free graph G' with $O(dn)$ vertices and maximum degree 4 such that $\alpha_d(G') = \alpha(G)$. Then by Theorem 9, a $2^{o(n)}$-time algorithm for d-Scattered Set for n-vertex claw-free graphs of maximum degree 4 would violate ETH.

The construction is slightly different based on the parity of d; let us first consider the case when d is odd. Let us construct the graph G^+ by attaching a path Q_v of $\ell = (d-1)/2$ edges to each vertex $v \in V(G)$; let us denote by $e_{v,1}, \ldots, e_{v,\ell}$ the edges of this path such that $e_{v,1}$ is incident with v. The graph G' is defined as the line graph of G^+, that is, each vertex of G' represents an edge of G^+ and two vertices of G' are adjacent if the corresponding two vertices share an endpoint. It is well known that line graphs are claw-free. As G^+ has $O(dn)$ edges and maximum degree 4 (recall that G has maximum degree 3), the line graph G' has $O(dn)$ vertices an edges. Thus an algorithm for Scattered Set with running time $2^{o(n)}$ on n-vertex claw-free graphs of maximum degree 3 could be used to solve MIS on n-vertex graphs with maximum degree 3 in time $2^{o(n)}$, contradicting ETH.

If there is an independent set S of size k in G, then we claim that the set $S' = \{e_{v,\ell} \mid v \in S\}$ is a d- scattered set of size k in G'. To see this, suppose for a contradiction that there are two vertices $u, v \in S$ such that the vertices of G' representing $e_{u,\ell}$ and $e_{v,\ell}$ are at distance at most $d-1$ from each other. This implies that there is a path in G^+ that has at most d edges and whose first and last edges are $e_{u,\ell}$ and $e_{v,\ell}$, respectively. However, such a path would need to contain all the ℓ edges of path Q_u and all the ℓ edges of Q_v, hence it can contain at most $d - 2\ell = 1$ edges outside these two paths. But u and v are not adjacent in G^+ by assumption, hence more than one edge is needed to complete Q_u and Q_v to a path, a contradiction.

Conversely, let S' be a distance-d scattered set in G', which corresponds to a set S^+ of edges in G^+. Observe that for any $v \in V(G)$, at most one edge of S^+ can be incident to the vertices of Q_v: otherwise, the corresponding two vertices in the line graph G' would have distance at most $\ell < d$. It is easy to see that if S^+ contains an edge incident to a vertex of Q_v, then we can always replace this edge with $e_{v,\ell}$, as this can only move it farther away from the other edges of S^+. Thus we may assume that every edge of S^+ is of the form $e_{v,\ell}$. Let us construct the set $S = \{v \mid e_{v,\ell} \in S^+\}$, which has size exactly k. Then S is independent in G: if $u, v \in S$ are adjacent in G, then there is a path of $2\ell + 1 = d$ edges in G^+ whose first an last edges are $e_{v,\ell}$ and $e_{u,\ell}$, respectively, hence the vertices of G' corresponding to them have distance at most $d-1$.

If $d \geq 4$ is even, then the proof is similar, but we obtain the graph G^+ by first subdividing each edge and attaching paths of length $\ell = d/2 - 1$ to each original vertex. The proof proceeds in a similar way: if u and v are adjacent in G, then G^+ has a path of $2\ell + 2 = d$ edges whose first and last edges are $e_{v,\ell}$ and $e_{u,\ell}$, respectively, hence the vertices of G' corresponding to them have distance at most $d-1$. ◀

There is a well-known and easy way of proving hardness of MIS on graphs with large girth: subdivide edges increases girth and the size of the largest independent set changes in a controlled way.

▶ **Lemma 11.** *If there is an $2^{o(n)}$-time algorithm for* MIS *on n-vertex graphs of maximum degree 3 and girth more than g for any fixed $g > 0$, then ETH fails.*

Proof. Let g be a fixed constant and let G be a simple graph with n vertices, m edges, and maximum degree 3 (hence $m = O(n)$). We construct a graph G' by subdividing each edge with $2g$ new vertices. We have that G' has $n' = O(n + gm) = O(n)$ vertices, maximum degree 3, and girth at least $3(2g+1)$. It is known and easy to show that subdividing the

edges this way increases the size of the maximum independent set exactly by gm. Thus a $2^{o(n')}$- time algorithm for n'-vertex graphs of maximum degree 3 and girth at least g could be used to give a $2^{o(n)}$-time algorithm for n-vertex graphs of maximum degree g, hence ETH would fail by Theorem 9. ◀

We use the lower bound of Lemma 11 to prove lower bounds for Scattered Set on C_t-free graphs.

▶ **Theorem 12.** *Assuming ETH, d-Scattered Set does not have a $2^{o(n)}$ algorithm on n-vertex C_t-free graphs with maximum degree 3 for any fixed $t \geq 3$ and $d \geq 2$.*

Proof. Let G be an n-vertex m-edge graph of maximum degree 3 and girth more than t. We construct a graph G' the following way: we subdivide each edge of G with $d-2$ new vertices to create a path of length $d-1$, and attach a path of length $d-1$ to each of the $(d-2)m = O(dn)$ new vertices created. The resulting graph has maximum degree 3, $O(d^2n)$ vertices and edges, and girth more than $(d-1)t$ (hence it is C_t-free). We claim that $\alpha_d(G') = \alpha(G) + m(d-2)$ holds. This means that an $2^{o(n')}$-time algorithm for Scattered Set n'-vertex C_t-free graphs with maximum degree 3 would give a $2^{o(n)}$-time algorithm for n-vertex graphs of maximum degree 3 and girth more than t and this would violate ETH by Lemma 11.

To see that $\alpha_d(G') = \alpha(G) + m(d-2)$ holds, consider first an independent set S of G. When constructing G', we attached $m(d-2)$ paths of length $d-1$. Let S' contain the degree-1 endpoints of these $m(d-2)$ paths, plus the vertices of G' corresponding to the vertices of S. It is easy to see that any two vertices of S' has distance at least d from each other: S is an independent set in G, hence the corresponding vertices in G' are at distance at least $2(d-1)$ from each other, while the degree-1 endpoints of the paths of length $d-1$ are at distance at least d from every other vertex that can potentially be in S'. This shows $\alpha_d(G') \geq \alpha(G) + m(d-2)$ Conversely, let S' be a set of vertices in G' that are at distance at least d from each other. The set S' contains two types of vertices: let S'_1 be the vertices that correspond to the original vertices of G and let S'_2 be the $m(d-2)d$ new vertices introduced in the construction of G'. Observe that S'_2 can be covered by $m(d-2)$ paths of length $d-1$ and each such path can contain at most one vertex of S', hence at most $m(d-2)$ vertices of S' can be in S'_2. We claim that S'_1 can contain at most $\alpha(G)$ vertices, as $S' \cap S'_1$ corresponds to an independent set of G. Indeed, if u and v are adjacent vertices of G, then the corresponding two vertices of G' are at distance $d-1$, hence they cannot be both present in S'. This shows $\alpha_d(G') \leq \alpha(G) + m(d-2)$, completing the proof of the correctness of the reduction. ◀

As the following corollary shows, putting together Theorems 10 and 12 implies Theorem 3(2).

▶ **Corollary 13.** *If H is a graph having a component that is not a path, then, assuming ETH, d-Scattered Set has no $2^{o(n+m)}$-time algorithm on n-vertex m-edge H-free graphs for any fixed $d \geq 3$.*

Proof. Suppose first that H is not a forest and hence some cycle C_t for $t \geq 3$ appears as an induced subgraph in H. Then the class of H-free graphs is a superset of C_t-free graphs, which means that statement follows from Theorem 12 (which gives a lower bound for a more restricted class of graphs).

Assume therefore that H is a forest. Then it has to have a component that is a tree, but not a path, hence it has a vertex v of degree at least 3. The neighbors of v are independent in

the forest H, which means that the claw $K_{1,3}$ appears in H as an induced subgraph. Then the class of H-free graphs is a superset of claw-free graphs, which means that statement follows from Theorem 10 (which gives a lower bound for a more restricted class of graphs). ◀

6 Conclusion

In spite of our results, it remains an open problem for an infinite class of graphs H, whether a subexponential or even a polynomial algorithm exists for MIS on H-free graphs. Namely, as indicated in the Introduction, among connected graphs these are the ones in which the triple of lengths of paths starting from the unique vertex of degree three is (i, j, k) with $i \leq j \leq k$ and with $(i, j, k) \neq (1, 1, 1), (1, 1, 2)$. Moreover, for paths, it is an unsolved question whether the problem is polynomial-time solvable for $H = P_t$, $t \geq 6$.

Our subexponential algorithm uses simple branching which clearly works for Weighted Independent Set as well.

For Scattered Set, we have seen that on P_t-free graphs there are algorithms subexponential in the number of edges, and Theorem 2 shows that polynomial-time algorithms are unlikely. But can one give a tight lower bound on the subexponential running time, perhaps showing that $1 - O(1/t)$ in the exponent of the exponent is in some sense best possible?

After the acceptance of this manuscript we learned that independently and simultaneously Brause (Ch. Brause, "A subexponential-time algorithm for the Maximum Independent Set in P_t-free graphs", *Discrete Applied Mathematics*, DOI:10.1016/j.dam.2016.06.016) also proved the subexponentiality of MIS on P_t-free graphs. (His time bound is weaker than the one in this paper.) Moreover, an unpublished result of Lokshtanov, Pilipczuk, and van Leuwen yields an algorithm with much better bound on the running time.

References

1 Geir Agnarsson, Peter Damaschke, and Magnús M. Halldórsson. Powers of geometric intersection graphs and dispersion algorithms. *Discrete Applied Mathematics*, 132(1-3):3–16, 2003. doi:10.1016/S0166-218X(03)00386-X.

2 V. E. Alekseev. The effect of local constraints on the complexity of determination of the graph independence number. In *Combinatorial-algebraic methods in applied mathematics*, pages 3–13. Gor′kov. Gos. Univ., Gorki, 1982.

3 Vladimir E. Alekseev. Polynomial algorithm for finding the largest independent sets in graphs without forks. *Discrete Applied Mathematics*, 135(1-3):3–16, 2004. doi:10.1016/S0166-218X(02)00290-1.

4 Gábor Bacsó and Zsolt Tuza. A characterization of graphs without long induced paths. *J. Graph Theory*, 14(4):455–464, 1990. doi:10.1002/jgt.3190140409.

5 Gábor Bacsó and Zsolt Tuza. Dominating cliques in P_5-free graphs. *Period. Math. Hungar.*, 21(4):303–308, 1990. doi:10.1007/BF02352694.

6 Binay K. Bhattacharya and Michael E. Houle. Generalized maximum independent sets for trees in subquadratic time. In *Algorithms and Computation, 10th International Symposium, ISAAC'99, Chennai, India, December 16-18, 1999, Proceedings*, pages 435–445, 1999. doi:10.1007/3-540-46632-0_44.

7 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.

8 Paul Erdős, Michael E. Saks, and Vera T. Sós. Maximum induced trees in graphs. *J. Comb. Theory, Ser. B*, 41(1):61–79, 1986. doi:10.1016/0095-8956(86)90028-6.

9 Hiroshi Eto, Fengrui Guo, and Eiji Miyano. Distance-d independent set problems for bipartite and chordal graphs. *J. Comb. Optim.*, 27(1):88–99, 2014. doi:10.1007/s10878-012-9594-4.

10 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

11 Daniel Lokshantov, Martin Vatshelle, and Yngve Villanger. Independent set in P_5-free graphs in polynomial time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 570–581, 2014. doi:10.1137/1.9781611973402.43.

12 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://albcom.lsi.upc.edu/ojs/index.php/beatcs/article/view/96.

13 Daniel Lokshtanov, Marcin Pilipczuk, and Erik Jan van Leeuwen. Independence and efficient domination on P_6-free graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1784–1803, 2016. doi:10.1137/1.9781611974331.ch124.

14 Dániel Marx and Michal Pilipczuk. Optimal parameterized algorithms for planar facility location problems using Voronoi diagrams. In *Algorithms – ESA 2015 – 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 865–877, 2015. doi:10.1007/978-3-662-48350-3_72.

15 M. Miller and J. Širáň. Moore graphs and beyond: A survey of the degree/diameter problem. *Electronic J. Combinatorics*, 20:1–92, 2013.

16 George J. Minty. On maximal independent sets of vertices in claw-free graphs. *J. Combin. Theory Ser. B*, 28(3):284–304, 1980. doi:10.1016/0095-8956(80)90074-X.

17 Bert Randerath and Ingo Schiermeyer. On maximum independent sets in P_5-free graphs. *Discrete Applied Mathematics*, 158(9):1041–1044, 2010. doi:10.1016/j.dam.2010.01.007.

18 Daniel J. Rosenkrantz, Giri Kumar Tayi, and S. S. Ravi. Facility dispersion problems under capacity and cost constraints. *J. Comb. Optim.*, 4(1):7–33, 2000. doi:10.1023/A:1009802105661.

19 Najiba Sbihi. Algorithme de recherche d'un stable de cardinalité maximum dans un graphe sans étoile. *Discrete Math.*, 29(1):53–76, 1980. doi:10.1016/0012-365X(90)90287-R.

20 Dimitrios M. Thilikos. Fast sub-exponential algorithms and compactness in planar graphs. In *Algorithms – ESA 2011 – 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, pages 358–369, 2011. doi:10.1007/978-3-642-23719-5_31.

21 David Zuckerman. Linear degree extractors and the inapproximability of Max Clique and Chromatic Number. *Theory of Computing*, 3(1):103–128, 2007. doi:10.4086/toc.2007.v003a006.

Parallel Multivariate Meta-Theorems

Max Bannach[1] and Till Tantau[2]

1 **Institute for Theoretical Computer Science, Universität zu Lübeck, Germany**
`bannach@tcs.uni-luebeck.de`
2 **Institute for Theoretical Computer Science, Universität zu Lübeck, Germany**
`tantau@tcs.uni-luebeck.de`

Abstract

Fixed-parameter tractability is based on the observation that many hard problems become tractable even on large inputs as long as certain input parameters are small. Originally, "tractable" just meant "solvable in polynomial time," but especially modern hardware raises the question of whether we can also achieve "solvable in polylogarithmic parallel time." A framework for this study of *parallel fixed-parameter tractability* is available and a number of isolated algorithmic results have been obtained in recent years, but one of the unifying core tools of classical FPT theory has been missing: algorithmic meta-theorems. We establish two such theorems by giving new upper bounds on the circuit depth necessary to solve the model checking problem for monadic second-order logic, once parameterized by the tree width and the formula (this is a parallel version of Courcelle's Theorem) and once by the tree depth and the formula. For our proofs we refine the analysis of earlier algorithms, especially of Bodlaender's, but also need to add new ideas, especially in the context where the parallel runtime is bounded by a function of the parameter and does not depend on the length of the input.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Parallel computation, FPT, meta-theorems, tree width, tree depth

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.4

1 Introduction

Algorithmic meta-theorems bound the computational resources needed to solve problems defined in a certain logic for inputs from a specific class of structures. The prime example is Courcelle's Theorem [5], which states that monadic second-order (MSO) definable problems can be solved in linear time on structures with bounded tree width. This yields, for instance, a linear time algorithm for the feedback vertex set problem on graphs of bounded tree width. Other examples are a "logspace version" [6] or a theorem for structures of bounded tree depth, where constant depth circuits (AC^0) suffice [7]; many more versions can be found in the surveys by Grohe and Kreutzer [10] and Kreutzer [12].

With the rise of multivariate algorithms, algorithmic meta-theorems have become useful tools for establishing parameterized upper bounds. The prime example is again Courcelle's Theorem, which actually gives a linear-time FPT-algorithm when the tree width of the input structure is the parameter. Since the tree width of a graph with a feedback vertex set of size k is at most $k+1$, the theorem shows that the naturally parameterized feedback vertex set problem can be solved in parameterized linear time.

The field of parameterized complexity is renowned for its ability to find algorithms that solve NP- or even PSPACE-complete problems in reasonable time. Unfortunately, "reasonable time" is not quite the same as "fast" and, furthermore, the instances in typical applications for these algorithms are huge. We may thus wish to speedup the computation by taking

advantage of the multiple cores and powerful GPUs present in modern hardware. In order to do so, we need *parallel* fixed-parameter algorithms. A first step in this direction was taken by us in [1], where we showed that the vertex cover problem, among several other problems, allows fast parallel fixed-parameter algorithms; but for many problems, including the feedback vertex set problem, the parallel parameterized complexity remains open. In particular, results concerning the parallel fixed-parameter tractability of problems have been obtained on a problem-by-problem basis without an overarching, unifying approach – which is exactly what the present paper tries to remedy.

Our Contributions. We formulate and prove different *parallel parameterized meta-theorems,* which unify previous results and allow us to obtain new algorithms for natural problems. Our meta-theorems are obtained by translating the logspace and circuit versions of Courcelle's Theorem from [6, 7] into parameterized counter parts, but we must point out already at this point that this is harder than one might expect: Unlike the original linear-time version of Courcelle's Theorem, which is "a theorem about parameterized complexity in disguise," the logspace and circuit versions just state that problems lie in the classes XL and XAC^0. However, these latter classes are presumably not even contained in FPT, let alone in parallel subclasses thereof and, thus, are not the classes we are looking for.

To establish the parallel parameterized meta-theorem, we need to study the parameterized parallel complexity of computing tree decomposition of parameterized width and possibly also depth. At the heart of Courcelle's Theorem and related versions are tree automata that process the tree decomposition of the input. We provide fast parallel algorithms to evaluate parameter-sized tree automata on arbitrary trees and on trees of parameterized depth. By combining these algorithms with the parallel algorithms for computing tree decompositions, we obtain parallel algorithms for monadic second-order model checking on graphs of parameterized tree width or parameterized tree depth ($p_{\phi,\text{td/tw}}$-MC(MSO)). The logic is defined as usual, for instance the following MSO-sentence describes that a graph is colorable with three colors:

$$\begin{aligned}\phi = \exists R\, \exists G\, \exists B\, \forall x \left(R(x) \vee G(x) \vee B(x)\right) \\ \wedge \forall x, y \left(E(x,y) \to (\neg R(x) \vee \neg R(y)) \wedge (\neg G(x) \vee \neg G(y)) \wedge (\neg B(x) \vee \neg B(y))\right).\end{aligned}$$

The model checking problem asks, given a logical structure (for instance a graph), and a logical formula, whether or not the structure is a model for the formula. For example, we have  $\models \phi$, but  $\not\models \phi$. For an introduction to the field, we refer to [8]. Our main results are stated in form of the following theorems (para-$\text{AC}^{0\uparrow}$ contains problems decidable by "FPT-sized" circuits whose depth depends only on the parameter, detailed definition follow later):

▶ **Theorem 1.** $p_{\phi,\text{td}}$-MC(MSO) $\in$ para-$\text{AC}^{0\uparrow}$.

▶ **Theorem 2.** $p_{\phi,\text{tw}}$-MC(MSO) $\in$ para-$\text{NC}^{2+\epsilon}$.

Armed with these new meta-theorems, we settle the parallel parameterized complexity of different natural problems, including the feedback vertex set problem.

Related Work. The prime example of algorithmic meta-theorems is Courcelle's Theorem [5] which becomes powerful in combination with Bodlaender's linear-time algorithm for computing optimal tree decompositions of graphs of bounded tree width [4]. Since the release of this theorem, many other meta-theorems, which place many problems in P, were presented,

see [10, 12] for surveys. For Courcelle's Theorem there are versions for other classes: a LOGCFL-version by Wanke [15], which was later improved to an L-version by Elberfeld, Jakoby, and the last author [6], who also prove an AC^0-version [7]. While most meta-theorems that place problems in P place the parameterized version of the problem in para-P = FPT, this is not the case for the last-mentioned versions: The L- and AC^0-versions of Courcelle's Theorem place problems in XL and XAC^0 and not, as we would like, in para-L and para-AC^0. Early studies on the parallel complexity of computing tree decompositions for graphs of bounded tree width where made by Bodlaender [3]. However, the algorithm does not obtain "FPT-work" in the parameterized setting and only yields a XNC-algorithm. Lagergren provided a parallel $O(\log^3 n)$ time algorithm using $O(n)$ processors for this problem in the CRCW model [13], which translates into a para-NC algorithm for parameterized problems. Bodlaender and Hagerup later provided a parallel algorithm with optimal speedup running in time $O(\log^2 n)$ using $O(n)$ operations on the EREW model [2]. This algorithm readily translates into a para-NC algorithm, but only the careful analysis done in this paper shows that it is actually a para-$NC^{2+\epsilon}$ algorithm.

Organization of This Paper. In Section 2 we define our basic terminology and recap the definition of classes of fixed-parameter parallelism. In Section 3 we provide parallel algorithms to compute tree decompositions of graphs with parameterized tree width or parameterized tree depth. In Section 4 we provide parallel algorithms to evaluate tree automata on arbitrary trees and on trees of parameterized depth. Putting it all together, we provide parallel algorithms for monadic second-order model checking on graphs of parameterized tree width or depth in Section 5. We close the paper by studying the parallel complexity of certain parameterized problems with the help of these meta-theorems in Section 6. Due to lack of space, proofs have been moved to the appendix.

2 Classes of Fixed-Parameter Parallelism

We use standard terminology of parameterized complexity theory, see for instance [8]. A *parameterized problem* is a tuple (Q, κ) of a language $Q \subseteq \Sigma^*$ and a *parameterization* $\kappa\colon \Sigma^* \to \mathbb{N}$. As we deal with small parameterized circuit classes, we require the parameter to be computable in DLOGTIME-uniform AC^0 or, equally, to be first-order computable.[1] We denote parameterized problems by a leading "p-" as in p-VERTEX-COVER, and, whenever the parameter is not clear from the context, we add it as index as in p_{tw}-DISTANCE.

A parameterized problem (Q, κ) is called *fixed-parameter tractable* if there is a language R decidable in polynomial time (P) and a computable function $f\colon \mathbb{N} \to \mathbb{N}$ such that $x \in Q$ if, and only if, $(x, 1^{f(\kappa(x))}) \in R$. That is, the problem is decidable in polynomial time after an arbitrarily complex pre-computation on the parameter. The resulting complexity class is called FPT or para-P. If we replace P in this definition by subclasses of P, we obtain subclasses of FPT, which inherit their inclusion structure from their classical counter parts:

$$\text{para-}AC^0 \subsetneq \text{para-}TC^0 \subseteq \text{para-}NC^1 \subseteq \text{para-L} \subseteq \text{para-NL} \subseteq \text{para-}AC^1 \subseteq \text{para-P}.$$

In order to explicitly define what the parameterized circuit classes contain, we use the definition from [1]:

[1] Sometimes this definition is to restrictive, for instance the tree width of a graph is computable in FPT, but probably not in P, and certainly not in AC^0. In such cases we assume that the input is extended by an upper bound on the parameter, which can easily be extracted in AC^0. However, in this case any algorithm deciding the problem has to verify this parameter by itself.

▶ **Definition 3** (Classes of Parallel Fixed-Parameter Tractability)**.** Let $d\colon \mathbb{N}^2 \to \mathbb{N}$ be a *depth bounding function* and $s\colon \mathbb{N}^2 \to \mathbb{N}$ be a *size bounding function* which both map each pair of an input length and a parameter to a number. We define para-AC$[d, s]$ as the class of parameterized problems (Q, κ) for which there exists a DLOGTIME-uniform[2] family $(C_{n,k})_{n,k\in\mathbb{N}}$ of AC-circuits (only NOT-, AND-, and OR-gates are allowed, AND- and OR- gates may have unbounded fan-in) such that: (1) For all $x \in \Sigma^*$, the circuit $C_{|x|,\kappa(x)}$ evaluates to 1 on input x if, and only if, $x \in Q$. (2) The depth of each $C_{n,k}$ is at most $d(n, k)$. (3) The size of each $C_{n,k}$ is at most $s(n, k)$.

We define the classes para-AC^i as para-AC^0 = para-AC$[O(1), f(\kappa(x)) \cdot |x|^{O(1)}]$ and for $i > 0$ para-AC^i = para-AC$[O(\log^i |x|), f(\kappa(x)) \cdot |x|^{O(1)}]$ (in slight abuse of notation, as its actually the union of this class over all computable functions f). However, there are also interesting new classes, namely the "up-"classes, defined in [1]:

$$\text{para-AC}^{i\uparrow} = \text{para-AC}[f(\kappa(x)) \cdot \log^i |x|, f(\kappa(x)) \cdot |x|^{O(1)}].$$

Note that para-AC$^{0\uparrow}$ captures exactly the problems that can be solved by a circuit of depth depending only on the parameter, and "FPT"-size. Notice, furthermore, that the "up-"classes can be strictly more powerful than the underlying classes (para-AC$^0 \subsetneq$ para-AC$^{0\uparrow}$ [1]), but that a slight increase of the depth in dependence on $|x|$ compensates this effect: We have para-AC$^i \subseteq$ para-AC$^{i\uparrow} \subseteq$ para-AC$^{i+\epsilon}$. These definitions and observations can, of course, also be applied to circuits of bounded fan-in (NC), and to circuits that are equipped with threshold-gates (TC).

To get familiar with parameterized circuits, let us consider an important technique from the design of parallel algorithms: *symmetry breaking*, that is, the ability to find parts of the input that can be processed in parallel. For graph algorithms in the PRAM model, this is often achieved by computing maximal independent-sets. In the lemma, as in the rest of the paper, f is an appropriate computable function and c is an appropriate constant.

▶ **Lemma 4.** *There is a* DLOGTIME*-uniform family of* AC*-circuits of depth* $f(k) + \log^* |V|$ *and size* $f(k) \cdot |V|^c$ *that, on input of an undirected graph* $G = (V, E)$ *and an integer* k*, outputs either that the maximum degree of* G *exceeds* k *or a maximal independent set* I *of* G*.*

Notice that, in sense of circuit classes, the lemma yields a para-AC$^{0+\epsilon} \subseteq$ para-NC$^{1+\epsilon}$ circuit for computing maximal independent sets with respect to the parameter "maximum degree."

3 Parallel Computation of Tree Decompositions

In our algorithmic meta-theorems, the tree *width* and tree *depth* of the input graphs are of special interest: First, they are parameters and, second, our algorithms work on the tree decompositions underlying the input graphs. Thus, it is of particular interest how such tree decompositions can be computed in parallel.

Recall the definition of a tree decomposition (T, ι) of a graph $G = (V, E)$. It is a rooted tree T together with a mapping ι from the nodes of T to subsets of V (which we call *bags*) such that for each vertex $v \in V$ and for each edge $\{v, w\} \in E$ there is (1) at least one node n in T with $v \in \iota(n)$, (2) at least one node n in T with $\{v, w\} \subseteq \iota(n)$, and (3) the set of nodes of

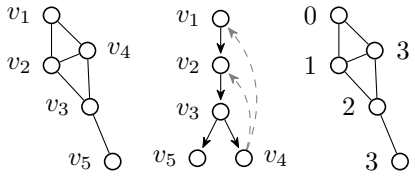
[2] In this context, this means that the circuit $C_{n,k}$ can be computed in time $f(k) + O(\log n)$ by a deterministic Turing machine that obtains $1^n \# 1^k$ as input.

T that contain v in their bag is connected. The *width* of a tree decomposition is the maximum size of its bags minus 1, its *depth* is the maximum of its width and the depth of the tree T. For a graph G, we define $\mathrm{tw}(G)$ to be the minimum width each tree decomposition of G has to have, and we define $\mathrm{td}(G)$ in a similar way for the tree depth.[3] For many algorithms it is useful to have a certain form of a tree decomposition: A *nice* tree decomposition is a tuple (T, ι, η) such that (T, ι) is a tree decomposition and $\eta\colon V(T) \to \{\text{leaf}, \text{introduce}, \text{join}, \text{forget}\}$ is a labeling function of the nodes. The nodes that are labeled as *leaf* are exactly the leafs and the root of T, and the bags of these nodes are empty. *Introduce*- and *forget*-nodes n have one child x such that there is one $v \in V$ with $v \notin \iota(x)$ and $\iota(n) = \iota(x) \cup \{v\}$, or $v \in \iota(x)$ and $\iota(n) = \iota(x) \setminus \{v\}$, respectively. *Join*-nodes n have two children x and y with $\iota(n) = \iota(x) = \iota(y)$. A tree decomposition (T, ι) is called *balanced* if T is a balanced tree; a nice tree decomposition (T, ι, η) is *balanced* if the tree obtained from T by contracting introduce and forget nodes is balanced. We refer to the textbook from Flum and Grohe for a more detailed introduction into the field [8].

Computing Depth-Bounded Tree Decompositions. We first study the case that we deal with graphs parameterized by their tree depth. This class of graphs is well suited for parallel algorithms, as a parallel algorithm can traverse the whole decomposition in time depending only on the parameter. We will see in this section that we can also compute a tree decomposition of parameterized depth within this time bound.

▶ **Theorem 5.** *There is a* DLOGTIME*-uniform family of* AC*-circuits of depth $f(k)$ and size $f(k)\cdot|G|^c$ that, on input of an undirected graph $G = (V, E)$ and an integer k, either determines $\mathrm{td}(G) > k$ or outputs a tree decomposition (T, ι) of G with depth bounded by $O\big(2^{\mathrm{td}(G)}\big)$.*

In order to prove Theorem 5 we will use known facts about the relation of bounded-depth tree decompositions and depth-first search trees [14]. To use these facts, we need a representation of a depth-first search tree that is suitable for our circuit model. Let $G = (V, E)$ be a graph with $s \in V$, and let T be a depth-first search tree of G starting at s, a *depth-first search labeling* is a mapping $\lambda_s\colon V \to \mathbb{N}$ such that $\lambda_s(v)$ is the distance from s to v in T. The figure below shows from left to right: an example graph, a depth-first search tree starting at v_1, and a corresponding depth-first search labeling.



In a similar way, we can define a *breadth-first search labeling* with respect to a breadth-first search tree. Notice that in this case the labeling is actually the (path) distance from s to the other vertices.

▶ **Lemma 6.** *There is a* DLOGTIME*-uniform family of* AC*-circuits of depth $f(k)$ and size $f(k) \cdot |G|^c$ that, on input of an undirected graph $G = (V, E)$, a vertex $s \in V$, and an integer k, either correctly detects that the longest path in G is longer than 2^k, or that output a depth-first and a breadth-first search labeling starting at s.*

[3] Note that the common definition of tree depth is slightly different, but that it is an upper bound for the definition we use.

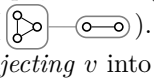
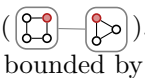
Proof of Theorem 5. It is a well-known fact [14] that the length of the longest path in a graph G is bounded by $2^{\mathrm{td}(G)}$. A direct consequence is that a depth-first search tree can be used to obtain a tree decomposition (T, ι) of width and depth bounded by $2^{\mathrm{td}(G)}$: let us assume G is connected and let T be a depth-first search tree rooted at an arbitrary start vertex $r \in V$. For all $v \in V$ define $\iota(v) = \{\, w \mid w \text{ lies on the unique path from } v \text{ to } r \text{ in } T \,\}$. The depth of T is naturally bounded by $2^{\mathrm{td}(G)}$, and, therefore, we also have $|\iota(v)| \leq 2^{\mathrm{td}(G)}$ for each $v \in V$. Since bags extend along the paths from the root to the leaves of T, all the conditions of a tree decomposition are satisfied by (T, ι).

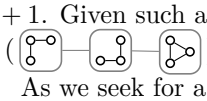
A circuit with the desired size and depth can compute a depth-first search labeling using Lemma 6, and either conclude that the length of the longest path exceeds k, and therefore $\mathrm{td}(G) > k$, or it can compute the bags of the decomposition in parallel. For each $v \in V$ the circuit initializes the bag $\iota(v) = \{v\}$. As long as $r \notin \iota(v)$, the circuit repeats the following sequentially: let $w \in \iota(v)$ the vertex that minimizes $\lambda(w)$ in $\iota(v)$, the circuits adds the unique $w' \in N(w)$ that satisfies $\lambda(w') = \lambda(w) - 1$ to $\iota(v)$. To complete the proof, we have to handle the case that G is not connected. The circuit can compute all connected components of G using a breadth-first search labeling (Lemma 6). Afterwards, the circuit can apply the algorithm from above to each connected component. Finally, the circuit adds a new empty root bag that is connected to the roots of all constructed tree decompositions. This operation does not increase the width and increases the depth only by one. ◀

Computing Width-Bounded Tree Decompositions. We will now handle the case that the input graph is parameterized by tree width. In this case the depth of a tree decomposition is not bounded by any function in the parameter and, thus, it seems unlikely that parallel algorithms running in time depending only on the parameter exist. And, indeed, deciding if a graph has tree width at most k for a fixed k is already L-complete and, hence, the parameterized version of this problem cannot lie in para-AC^0 or para-$\mathrm{AC}^{0\uparrow}$.

▶ **Theorem 7.** *There is a* DLOGTIME*-uniform family of* NC*-circuits with depth* $f(k) + \log^{2+\epsilon} |G|$ *and width* $f(k) \cdot |G|^c$ *that, on input of an undirected graph* $G = (V, E)$ *and an integer* k*, either determines* $\mathrm{tw}(G) > k$ *or outputs a tree decomposition of* G *of width at most* k*.*

The proof of Theorem 7 is essentially a new analysis of a parallel algorithm from Bodlaender and Hagerup [2]. They provide an $O(\log^2 n)$ time and $O(n)$ work algorithm on the EREW-PRAM model to compute optimal tree decompositions of graphs with bounded tree width, from which one can derive that the problem of computing a tree decomposition lies somewhere in para-NC. Our main contribution in the following is a careful analysis regarding the exact circuit class the algorithm achieves: It is para-$\mathrm{NC}^{2+\epsilon}$ for all $\epsilon > 0$.

The idea of the algorithm is as follows: If $G = (V, E)$ is small enough, we can compute an optimal tree decomposition via "brute-force", otherwise we try to reduce the graph until it has a suitable size. We call two vertices $u, v \in V$ *reduction partner* if they are adjacent or twins (). We can reduce the size of G by 1 if we *contract* the two vertices, that is, if we remove v from G after connecting all neighbors of v to u (without creating multi-edges: ). Let G' be the resulting graph, and let (T', ι') be a recursively computed tree decomposition of G' of width at most k (). We can compute a tree decomposition (T, ι) of G of width at most $k + 1$ by *injecting* v into (T', ι'), that is, by adding v to all bags that contain u (). The resulting tree decomposition is most likely not optimal, but its width is bounded by a function in k and we can use it to compute an implicit representation of an optimal tree decomposition of G. This implicit representation, called *path labeled tree*

representation, is a binary tree T in which for every $v \in V$ exactly two vertices are labeled with v, i. e., the vertices of V correspond to paths in T (). If we consider the nodes as bags, each bag that lies on the unique path between two nodes labeled width v will contain v. Each node may be labeled with multiple vertices, but of course with at most $k+1$. Given such an implicit representation, we can compute a tree decomposition of width k ().

As we seek for a circuit of polylogarithmic depth, we can not only contract one reduction pair in every round, as we would require $O(|V|)$ rounds. Fortunately, there are always multiple reduction partners that can be contracted in parallel. The correctness of the algorithm is shown in [2]. For us, it remains to show that we can implement the algorithm in para-$\mathrm{NC}^{2+\epsilon}$, a task for which we have to show that each subfunction of the algorithm can be realized by circuits of logarithmic depth and polynomial size. The following lemmas show that all parts of a *single* iteration of the algorithm can be computed by a para-$\mathrm{NC}^{1+\epsilon}$-circuit.

▶ **Lemma 8.** *There is a* DLOGTIME*-uniform family of* NC*-circuits of depth* $f(k)+\log^{1+\epsilon}|V|$ *and size* $f(k)\cdot|V|^c$ *that, on input of a graph* $G=(V,E)$ *and* $k\in\mathbb{N}$*, outputs a set* I *of* $1/g(k)\cdot|V|$ *pairs of vertices that can be contracted in parallel, or that concludes* $\mathrm{tw}(G)>k$*.*

▶ **Lemma 9.** *There is a* DLOGTIME*-uniform family of* NC*-circuits of depth* $f(k)\cdot\log|V|$ *and size* $f(k)\cdot|V|^c$ *that, on input of a graph* $G=(V,E)$*, a set of pairs of vertices* I*, a graph* $G'=(V',E')$ *that is obtained from* G *by contracting the pairs in* I*, and a tree decomposition* (T',ι') *of* G' *of width* k*, outputs a balanced and nice tree decomposition* (T,ι,η) *of* G *of width at most* $8k+3$ *and depth* $(16k+6)\cdot\log|V|+1$*.*

▶ **Lemma 10.** *There is a* DLOGTIME*-uniform family of* NC*-circuits of depth* $f(k)\cdot\log|V|$ *and size* $f(k)\cdot|V|^c$ *that, on input of a graph* $G=(V,E)$*, an integer* k*, and of a balanced and nice tree decomposition* (T,ι,η) *of* G *of width at most* $\ell\le f(k)$*, outputs either* $\mathrm{tw}(G)>k$ *or a width-*k *tree decomposition of* G*.*

Proof of Theorem 7. The circuit first checks whether the size of the input graph is bounded by k. If this is the case, an optimal tree decomposition can be computed via "brute-force". Otherwise, the circuit computes a set of $1/f(k)\cdot|V|$ reduction pairs using Lemma 8, or concludes that the tree width of G exceeds k. The circuit reduces G to G' by contracting the reduction pairs (the lemma guarantees that this is possible in parallel) and recursively computes a tree decomposition of G'. This tree decomposition can be transformed to a nice and balanced decomposition of G of width bounded by a function in k using Lemma 9. Finally, the circuit can reduce the width of the decomposition to k or conclude $\mathrm{tw}(G)>k$ using Lemma 10.

Since Lemma 8 provides us with $1/f(k)\cdot|V|$ reduction pairs, $f(k)\cdot\log|V|$ rounds of the algorithm are sufficient to reduce the graph to a size depending only on the parameter. Considering each round as a subcircuit, each subcircuit has to execute the algorithms from the lemmas 8, 9, and 10. The most expensive part is Lemma 8, as the circuit needs depth $f(k)+\log^{1+\epsilon}$ here, for the lemmas 9 and 10 circuits of depth $f(k)\cdot\log|V|\le f(k)+\log^{1+\epsilon}|V|$ are sufficient. The complete circuit has, therefore, a total depth of $f(k)\log|V|\cdot\big(f(k)+\log^{1+\epsilon}|V|\big)\le f(k)+\log^{2+\epsilon}|V|$, and is, hence, a para-$\mathrm{NC}^{2+\epsilon}$-circuit. ◀

4 Parallel Evaluation of Tree Automata

A key aspect of modern algorithmic meta-theorems is the simulation of tree automata, since such theorems commonly translate a tree decomposition of the input structure into a labeled

tree that is accepted by a certain tree automaton if, and only if, the structure was a model for the input formula. "Classical" translations produce degree-bounded trees that are then processed by classical tree automata. However, this approach may increase the depth of the tree decomposition by up to a logarithmic factor, which is unacceptable if we wish to handle the tree in parallel time depending on the depth of the tree. As a solution, the authors of [7] suggest the use of *multiset automata*. A *multiset* M is a set S together with a multiplicity function $\#_M \colon S \to \mathbb{N}$. The *multiplicity* of M is $\max_{e \in S} \#_M(e)$. We denote by $\mathcal{P}_\omega(S)$ the class of all multisets of S and by $\mathcal{P}_m(S)$ the class of all multisets of multiplicity at most $m \in \mathbb{N}$ of S. Notice that $\mathcal{P}_1(S)$ is just the standard power set of S. For a multiset $M \in \mathcal{P}_\omega(S)$ and a number $m \in \mathbb{N}$, the *capped version* $M|_m$ of M is defined by setting $\#_M(e) = \min(\#_M(e), m)$ for all $e \in S$.

▶ **Definition 11** (Multiset Automaton)**.** A *nondeterministic* (bottom-up) multiset automaton is a tuple $\mathcal{A} = (\Sigma, Q, Q_a, \Delta, m)$ consisting of an *alphabet* Σ, a *state set* Q with *accepting states* $Q_a \subseteq Q$, a *state transition relation* $\Delta \subseteq \Sigma \times \mathcal{P}_m(Q) \times Q$, and a *multiplicity bound* $m \in \mathbb{N}$. The automaton is *deterministic* if for every $\sigma \in \Sigma$ and every $M \in \mathcal{P}_m(Q)$ there is exactly one $q \in Q$ with $(\sigma, M, q) \in \Delta$; in this case we can view Δ as *state transition function* $\delta \colon \Sigma \times \mathcal{P}_m(Q) \to Q$.

▶ **Definition 12** (Computation of a Multiset Automaton)**.** Let $(\mathfrak{T}, \lambda)$ be a labeled tree, where $\lambda \colon V(\mathfrak{T}) \to \Sigma$ is the labeling function, and let $\mathfrak{A} = (\Sigma, Q, Q_a, \Delta, m)$ be a multiset automaton. A *computation* of $\mathfrak{A}$ on $(\mathfrak{T}, \lambda)$ is a partial assignment $q \colon V(\mathfrak{T}) \to Q$ such that for every node $n \in V(\mathfrak{T})$ for which $q(n)$ is defined, we have that (a) the value $q(c)$ is defined for each child c of n in $\mathfrak{T}$ and (b) for the multiset $M = \{\, q(c) \mid c \text{ is a child of } n \,\}$ we have $(\lambda(n), M\,|_m, q(n)) \in \Delta$. A computation is *accepting* if $q(r) \in Q_a$ holds for the root node r of $\mathfrak{T}$. The *tree language* $\mathrm{L}(\mathfrak{A})$ contains all labeled trees accepted by $\mathfrak{A}$.

▶ **Fact 13** ([7])**.** *The following statements hold and are constructive:*

1. *For all multiset automata $\mathfrak{A}$ and $\mathfrak{B}$ there is a multiset automaton $\mathfrak{C}$ with $\mathrm{L}(\mathfrak{C}) = \mathrm{L}(\mathfrak{A}) \cap \mathrm{L}(\mathfrak{B})$;*
2. *For every nondeterministic multiset automaton $\mathfrak{A}$ there is a deterministic multiset automaton $\mathfrak{B}$ with $\mathrm{L}(\mathfrak{A}) = \mathrm{L}(\mathfrak{B})$;*
3. *For every multiset automaton $\mathfrak{A}$ there is a multiset automaton $\mathfrak{B}$ accepting the complement of $\mathrm{L}(\mathfrak{A})$.*

The actual aim of this section is to study the parallel parameterized complexity of the simulation of a multiset automaton. Since we will need such simulations in different scenarios, instead of classifying the problem into complexity classes, we identify circuit families depending on different parameters.

▶ **Lemma 14.** *Let $S_{k,d}$ be the set of labeled trees $(\mathfrak{T}, \lambda)$ of maximal depth d and maximal degree k. There is a* DLOGTIME*-uniform family of circuits over the standard base (only* AND*-,* OR*-, and* NOT*-gates) with fan-in k, depth $f(|\mathfrak{A}|) \cdot d$ and size $f(|\mathfrak{A}|) \cdot |\mathfrak{T}|^c$ that, on input of a labeled tree $(\mathfrak{T}, \lambda) \in S_{k,d}$ and a multiset automata $\mathfrak{A} = (\Sigma, Q, Q_a, \Delta, m)$, decides whether or not $(\mathfrak{T}, \lambda) \in \mathrm{L}(\mathfrak{A})$ holds.*

As used later on, we will mention two special cases of Lemma 14: The simulation of multiset automata can be performed (a) in para-$\mathrm{AC}^{0\uparrow}$ for trees of depth bounded by the parameter and (b) in para-$\mathrm{NC}^{1\uparrow}$ for balanced binary trees. Here, the size of the automata is the parameter.

5 Parallel Second-Order Model Checking

The goal of this section is to actually prove Theorem 1 and Theorem 2. The classical way of proving variants of Courcelle's Theorem is as follows: On input of a logical structure $\mathcal{S}$ and a MSO-formula ϕ, we first compute a tree decomposition (T, ι) of $\mathcal{S}$. This tree decomposition is then translated into a s-tree-structure $\mathcal{T}$ and ϕ is translated to a new MSO-formula ψ such that $\mathcal{S} \models \phi \Leftrightarrow \mathcal{T} \models \psi$. To decide $\mathcal{T} \models \psi$, the s-tree-structure $\mathcal{T}$ is transformed into a labeled tree $(\mathfrak{T}, \lambda)$ and ψ is turned into a multiset automata $\mathfrak{A}$ such that $\mathcal{T} \models \psi \Leftrightarrow (\mathfrak{T}, \lambda) \in \mathrm{L}(\mathfrak{A})$. Here, an s-tree-structure is a structure $\mathcal{T} = (V, E^{\mathcal{T}}, P_1^1, \dots, P_s^{\mathcal{T}})$ over the signature $\tau_{s\text{-tree}} = (E^2, P_1^1, \dots, P_s^1)$ where $(V, E^{\mathcal{T}})$ is a directed tree.

▶ **Fact 15** (Implicit in [7]). *There are functions h_1, h_2, h_3 and h_4 performing the following mappings:*

1. *The input for h_1 are a structure $\mathcal{S}$ together with a width-w tree decomposition (T, ι) of $\mathcal{S}$ and an* MSO*-formula ϕ. The output is an s-tree-structure $\mathcal{T}$.*
2. *The input for h_2 are an* MSO*-formula ϕ and a tree width w. The output is an* MSO*-formula formula ψ.*
3. *The input for h_3 are an s-tree-structure $\mathcal{T}$ and an* MSO*-formula ψ. The output is a labeled tree $(\mathfrak{T}, \lambda)$ of the same depth.*
4. *The input for h_4 is an* MSO*-formula ψ. The output is a multiset automaton $\mathfrak{A}$.*

The following holds for the values computed by these functions:

$$\mathcal{S} \models \phi \iff \mathcal{T} \models \psi \iff (\mathfrak{T}, \lambda) \in \mathrm{L}(\mathfrak{A}).$$

All h_i are computable and h_1 and h_3 are even computable by DLOGTIME*-uniform* AC*-circuits of depth $O(1)$ and size $f(\phi, w)|\mathcal{S}||T|$.*

Since the size of ϕ and the tree depth or width of the input structure are parameters in our setting, we can use Fact 15 to prove Theorem 1 and Theorem 2:

Proof of Theorem 1. On input of a logical structure $\mathcal{S}$ and an MSO-formula ϕ, a para-$\mathrm{AC}^{0\uparrow}$-circuit can compute a tree decomposition (T, ι) of the Gaifman graph of $\mathcal{S}$ (the graph that uses the universe of $\mathcal{S}$ as vertex-set and that contains an edge between two elements if, and only if, the two elements stand in any relation) using Theorem 5. Given the tuple $(\mathcal{S}, (T, \iota), \phi)$, the circuit can then compute a labeled tree $(\mathfrak{T}, \lambda)$ and a multiset automaton $\mathfrak{A}$ using Fact 15. The depth of $\mathfrak{T}$ is bounded by the depth of (T, ι) and, hence, bounded by the parameter. Furthermore, we have $|\mathfrak{A}| \leq f(|\psi| + \mathrm{td}(\mathcal{S}))$ for a computable function f. Hence, a para-$\mathrm{AC}^{0\uparrow}$-circuit can now invoke Lemma 14 and output the result. ◀

Proof of Theorem 2. The proof is almost identical to the proof of Theorem 1. On input of a logical structure $\mathcal{S}$ and a MSO-formula ϕ, a para-$\mathrm{NC}^{2+\epsilon}$-circuit can compute a tree decomposition (T, ι) of the Gaifman graph of $\mathcal{S}$ using Theorem 7. At this point a problem arises, as the depth of (T, ι) is not bounded. This can be overcome as follows: Let the width of (T, ι) be w, then a FTC^0-circuit can compute a balanced tree decomposition (T', ι') of width at most $4w + 3$ [7]. Given this decomposition, we can proceed as in the proof of Theorem 1 and compute the labeled tree $(\mathfrak{T}, \lambda)$ and a multiset automaton $\mathfrak{A}$. Since $(\mathfrak{T}, \lambda)$ is balanced, it is binary and of logarithmic depth, and, therefore, Lemma 14 can be invoked by a para-$\mathrm{NC}^{1\uparrow}$-circuit which presents the result as output. ◀

6 Applications

Most graph problems studied in complexity theory can be described in monadic second order logic, including vertex cover, dominating set, independent set, or clique, and, thus, our algorithmic meta-theorems apply to them. For instance, we get corollaries like $p_{\mathrm{tw},k}$-DOMINATING-SET $\in$ para-NC$^{2+\epsilon}$ and $p_{\mathrm{td},k}$-DOMINATING-SET $\in$ para-AC$^{0\uparrow}$.

It is, however, worth to take a closer look, as we are naturally interested in more precise parameterizations than in the combined parameter $\mathrm{td}/\mathrm{tw} + k$. Although the tree width and depth are fairly sensible parameters, we are even more interested in the complexity of the problems without restrictions on *these* parameters (but, perhaps still with other, more natural parameters). Sometimes this is possible, as the tree width is parameterized *indirectly*. For the feedback vertex set problem (given an undirected graph $G = (V, E)$ and a parameter k, decide whether there exists a set $S \subseteq V$ with $|S| \leq k$ such that $G[V \setminus S]$ is acyclic) the existence of such a set implies that the tree width of G is at most $k + 1$: Since $G[V \setminus S]$ is a tree, adding S to each bag of a tree decomposition of $G[V \setminus S]$ yields a tree decomposition of G of width at most $k + 1$.

▶ **Corollary 16.** p_k-FEEDBACK-VERTEX-SET $\in$ para-NC$^{2+\epsilon}$.

The above corollary is currently the best result on the parallel parameterized complexity of the feedback vertex set problem. In other scenarios the opposite is possible, i. e., the tree width indirectly parameterizes the solution size. A well known example is the clique problem, as any graph with tree width at most w can not contain a clique bigger than $w + 1$.

▶ **Corollary 17.** p_{td}-CLIQUE $\in$ para-AC$^{0\uparrow}$, p_{tw}-CLIQUE $\in$ para-NC$^{2+\epsilon}$.

On the other hand, there are problems where we can not hope for such effects. For instance, the dominating-set problem is well known to be W[2]-complete and, hence, we can not hope to get rid of the parameter tree width. Since the tree width does not bound k in this case either, we do not get rid of this parameter as well. Nevertheless, our meta-theorems at least improve the known upper bound for the dominating-set problem with combined parameter. In contrast, for the vertex cover problem we do need both parameters as well and obtain $p_{\mathrm{tw},k}$-VERTEX-COVER $\in$ para-NC$^{2+\epsilon}$ ($p_{\mathrm{td},k}$-VERTEX-COVER $\in$ para-AC$^{0\uparrow}$), but one can show directly [1] that p_k-VERTEX-COVER $\in$ para-AC^0 holds. In other words, our algorithmic meta-theorems do not yield an optimal bound on the vertex cover problem, a "less generic" approach yields better bounds.

Reachability Problems. The charm of studying parameterized parallel complexity is that it is not only interesting to consider NP- or even PSPACE-hard problems, but also problems that lie within P. For instance, the classical reachability problem in directed graphs REACH $= \{ (G, s, t) \mid \text{in } G \text{ is a path from } s \text{ to } t \}$ is a natural NL-complete problem. If we consider graphs with parameterized tree depth, the complexity of the problem can be lowered by Theorem 1.

▶ **Corollary 18.** p_{td}-REACH $\in$ para-AC$^{0\uparrow}$.

From the point of view of parallel complexity, we are especially interested in P-complete problems, since it is believed that such problems are inherent sequential. A natural P-complete version of the reachability problem is the alternating reachability problem [11], which is based on the following definition of *alternating paths:* Given a directed graph $G = (V, E)$ and a partition $V = V_\exists \cup V_\forall$, an *alternating path* from s to t is a set S of paths in G, all of

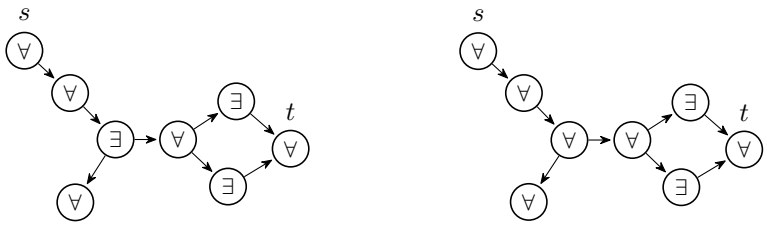


Figure 1 Examples of input graphs for the alternating reachability problem. In left graph there is an alternating path from s to t and the alternating distance is 5, in the right one there is not.

which end at t, such that (1) exactly one of them starts at s; (2) when a path in S starts at some $v \in V_\exists \setminus \{t\}$, then there is for some w with $(v, w) \in E$ a path in S starting at w; and (3) when a path in S starts at some $v \in V_\forall \setminus \{t\}$, then for all w with $(v, w) \in E$ there is a path in S starting at w (and there is at least one such w). The *length* of an alternating path is the maximum length of any path in the set S. The *alternating distance* between two vertices is the minimum distance of any alternating path between them.

▶ **Problem 19** (p_{tw}-AREACH, p_{td}-AREACH).
Instance: *A directed graph $G = (V, E)$, a partition $V = V_\exists \cup V_\forall$, and two vertices $s, t \in V$.*
Parameter: *Tree width or tree depth of G.*
Question: *Is there an alternating path from s to t in G?*

▶ **Problem 20** ($p_{\text{tw},d}$-ADISTANCE, $p_{\text{td},d}$-ADISTANCE).
Instance: *A directed graph $G = (V, E)$, a partition $V = V_\exists \cup V_\forall$, two vertices $s, t \in V$, a distance d.*
Parameter: *Tree width or tree depth of G as well as d.*
Question: *Is the alternating distance from s to t in G at most d?*

It is a standard exercise to describe the alternating reachability and distance problems using a monadic second order formula and, thus, our algorithmic meta-theorems yield the following:

▶ **Corollary 21.** p_{tw}-AREACH $\in$ para-NC$^{2+\epsilon}$, $p_{\text{tw},d}$-ADISTANCE $\in$ para-NC$^{2+\epsilon}$.

▶ **Corollary 22.** p_{td}-AREACH $\in$ para-AC$^{0\uparrow}$, $p_{\text{td},d}$-ADISTANCE $\in$ para-AC$^{0\uparrow}$.

It turns out that, as for the vertex cover problem, for the alternating distance problem we can do better, but also, that the classes we study are the "right" classes for these problems:

▶ **Theorem 23.** p_d-ADISTANCE *is complete for* para-AC$^{0\uparrow}$ *under* para-AC^0*-reduction.*

7 Conclusion

Algorithmic meta-theorems play a key role in modern complexity theory. We have seen that this powerful tool can also be applied to the study of parameterized parallel algorithms. Indeed, the results state that MSO-definable problems on graphs with parameterized tree width do not only allow linear time dynamic programs, but that these problems also allow fast parallel algorithms as well. The theorems show that problems definable in monadic second order logic can be solved in parallel time $f(k)$ or $f(k) \cdot \log n$ if a tree decomposition

of parameterized depth or width is given. In the first case, we have seen that such a decomposition can be computed in the same time. However, in the second case it turns out that the bottleneck is the computation of such a decomposition, since we were only able to show that this can be done in time $f(k) + \log^{2+\epsilon} n$. A reasonable research goal is therefore to seek an algorithm between para-L and para-$\text{NC}^{2+\epsilon}$ that computes a tree decomposition of a given graph with parameterized tree width. A first step would be to reduce the circuit depth to para-$\text{NC}^{2\uparrow}$.

References

1. M. Bannach, C. Stockhusen, and T. Tantau. Fast parallel fixed-parameter algorithms via color coding. In *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*, pages 224–235. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.IPEC.2015.224.
2. H. Bodlaender and T. Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM Journal on Computing*, 27(6):1725–1746, 1998. doi:10.1137/S0097539795289859.
3. Hans L. Bodlaender. NC-algorithms for graphs with small treewidth. In *Graph-Theoretic Concepts in Computer Science: International Workshop*, WG'88, pages 1–10. Springer Berlin Heidelberg, 1989. doi:10.1007/3-540-50728-0_32.
4. H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, STOC'93, pages 226–234. ACM, New York, USA, 1993. doi:10.1145/167088.167161.
5. B. Courcelle. Graph rewriting: An algebraic and logic approach. In *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, pages 193–242. Elsevier, Amsterdam, Netherlands and MIT Press, Cambridge, Massachusetts, 1990. doi:10.1016/B978-0-444-88074-1.50010-X.
6. M. Elberfeld, A. Jakoby, and T. Tantau. Logspace Versions of the Theorems of Bodleander and Courcelle. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science, October 23–26, 2010, Las Vegas, USA*, FOCS'10, pages 143–152. IEEE Computer Society, Los Alamitos, California, 2010. doi:10.1109/FOCS.2010.21.
7. M. Elberfeld, A. Jakoby, and T. Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. In *Proceedings of the Twenty-Ninth International Symposium on Theoretical Aspects of Computer Science, February 29 – March 3, 2012, Prais, France*, STACS'12, pages 66–77. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.66.
8. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, Heidelberg, Germany, 2006. doi:10.1007/3-540-29953-X.
9. A. Goldberg, S. Plotkin, and G. Shannon. Parallel symmetry-breaking in sparse graphs. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC'87, pages 315–324. ACM, New York, USA, 1987. doi:10.1145/28395.28429.
10. M. Grohe and S. Kreutzer. Methods for algorithmic meta theorems. In *Model Theoretic Methods in Finite Combinatorics*, pages 181–206. AMS, Contemporary Mathematics Series, 2011. doi:10.1090/conm/558/11051.
11. Neil Immerman. Languages which capture complexity classes. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC'83, pages 347–354. ACM New York, NY, 1983. doi:10.1145/800061.808765.
12. Stephan Kreutzer. Algorithmic meta-theorems. *CoRR*, abs/0902.3616, 2009. URL: http://arxiv.org/abs/0902.3616.

13 Jens Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *Journal of Algorithms*, 20:20–44, 1996. doi:10.1006/jagm.1996.0002.

14 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity*. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-27875-4.

15 Egon Wanke. Bounded tree-width and logcfl. *Graph-Theoretic Concepts in Computer Science*, 790:33–44, 2005. doi:10.1006/jagm.1994.1022.

A Technical Appendix: Proofs

For the readers convenience, the claims of the proofs given in this appendix are repeated before the proofs.

Claim of Lemma 4. *There is a* DLOGTIME*-uniform family of* AC*-circuits of depth* $f(k) + \log^* |V|$ *and size* $f(k) \cdot |V|^c$ *that, on input of an undirected graph* $G = (V, E)$ *and an integer* k*, outputs either that the maximum degree of* G *exceeds* k *or a maximal independent set* I *of* G*.*

Proof. As the circuit may have depth $f(k)$, it can count the degree of each vertex and can directly reject if any degree exceeds k [1]. Otherwise, the circuit implements the algorithm from Goldberg, Plotkin, and Shannon to compute a maximal independent set in degree-bounded graphs [9]. The circuit interprets G as directed graph $\vec{G}$ by considering each edge $\{u, v\}$ as two directed edges (u, v) and (v, u). The edge set of this graph is partitioned into k sets $E_1, \dots, E_k$ such that each of the graphs $\vec{G}_i = (V, E_i)$ has only vertices of out-degree at most 1. This partition can be computed in depth $f(k)$ as the circuit has essentially to count up to k.

The circuit now performs the following operations on all $\vec{G}_i$ in parallel: First, in constant depth, an initial coloring of $\vec{G}_i$ is computed by assigning each vertex v_i the color $i \in \mathbb{N}$, which needs at most $\log |V|$ bits. This coloring can be improved to a coloring with $\log |V|$ colors in constant depth: Replace the color c of each vertex v by $2k + b$, where k is the position of the lowest bit on which c differs from the color of the unique successor of v, and where b is the value of this bit. Computing this improvement consecutively $\log^* |V|$ times yields a coloring with 6 colors [9].

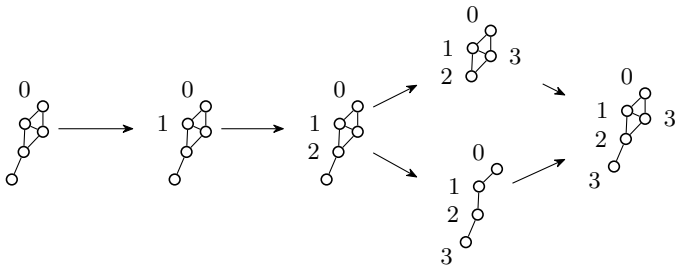
Given the colorings of the k graphs $\vec{G}_i$, the circuit can compute a 6^k coloring of G by assigning to each vertex the k-tuple of colors that this vertex has in the different $\vec{G}_i$. Finally, the circuit initializes a set $I = \emptyset$, iterates over the colors and, in parallel, adds all vertices of the current color that do not have a neighbor in I to I. As each step can be performed in a constant number of AC-layers, the set I can be computed in $f(k)$ AC-layers. The circuit outputs I, as the final value of I is a maximal independent-set. The total depth of the circuit is $f(k) + \log^* |V|$. ◀

Claim of Lemma 6. *There is a* DLOGTIME*-uniform family of* AC*-circuits of depth* $f(k)$ *and size* $f(k) \cdot |G|^c$ *that, on input of an undirected graph* $G = (V, E)$*, a vertex* $s \in V$*, and an integer* k*, either correctly detects that the longest path in* G *is longer than* 2^k*, or that output a depth-first and a breadth-first search labeling starting at* s*.*

Proof. We first handle the breadth-first search labeling, which yields a natural parallel algorithm. Our circuit starts by assigning color 0 to s. The circuit is build up of layers, where layer $i + 1$ assigns color $i + 1$ to each vertex that is not colored yet and that has at least one vertex of color i as neighbor. The algorithm stops if all vertices are colored, or at the very last after 2^k layers. In the later case, the circuit can report that the length of

the longest path exceeds 2^k. After a run of the algorithm, each vertex that has obtained a color is in the same connected component as s and, furthermore, the colors constitute a breadth-first search labeling starting at s.

Computing a depth-first search labeling turns out to be more complicated, since an AC-circuit of the desired depth cannot simply follow a path of the search tree and "backtrack" once it reaches a leaf, as the depth of the circuit would not be bounded by the longest path in this case. Instead, we have to compute the vertices which have more than one child in the depth-first search tree in advance. Once we know these vertices, we can perform a fork and compute the depth-first search labeling for all of there children in parallel. Since we have seen how the circuit can compute a breadth-first search labeling, we can assume that we have access to a subcircuit that computes the connected components of G. In order to compute the depth-first search labeling, the circuit first computes these connected components and checks if the the longest path in all these components is bounded by 2^k. Afterwards, the following algorithm, which we call a *phase*, is executed in parallel on all connected components with color $c = 0$ as argument. Each phase does nothing if all vertices are colored, this is the end of the recursion. If $c = 0$, an arbitrary vertex is selected and colored with c, otherwise an arbitrary vertex that is not colored, but that has a neighbor of color $c - 1$, is selected and colored with c. At the end of a phase the vertices of G are partitioned in colored vertices C and the uncolored vertices $V \setminus C$. The circuit computes the connected components of $G[V \setminus C]$, which we denote by $V_1, \ldots, V_\ell \subseteq V \setminus C$. Afterwards, a new phase is started recursively on each graph $G[V_i \cup C]$. If all phases are completed, the coloring constitutes a depth-first search labeling starting at s.



Since this algorithm never performs backtracking, the number of consecutive phases is bounded by the length of the longest path, which is bounded by 2^k. For each phase, a circuit of depth $f(k)$ is sufficient, since the most expensive part is clearly the computation of the connected components. Thus, a depth first-search labeling can be computed by an AC-circuit of depth $f(k)$. ◀

Claim of Lemma 8. *There is a* DLOGTIME*-uniform family of* NC*-circuits of depth* $f(k) + \log^{1+\epsilon} |V|$ *and size* $f(k) \cdot |V|^c$ *that, on input of a graph* $G = (V, E)$ *and* $k \in \mathbb{N}$*, outputs a set* I *of* $1/g(k) \cdot |V|$ *pairs of vertices that can be contracted in parallel, or that concludes* $\mathrm{tw}(G) > k$.

Proof. We call two vertices u, v *reduction partners* if we have either $\{u, v\} \in E$, or if they are twins, i. e., $N(u) = N(v)$. Let us call a vertex v *d-small* if $\delta(v) \leq d$.

Let $d = 2^{k+4}(54k + 54)$ and $c = 1/\big(8(27k + 27)^2\big)$. If $\mathrm{tw}(G) \leq k$, then there are at least $c|V|/2$ distinct pairs $\{u, v\}$ of d-small vertices that are reduction partners [2]. Since a circuit of the desired size can check all pairs of vertices in parallel, it can compute in the desired

depth a set S of reduction partners. Furthermore, the circuit can check whether $|S| \leq c|V|/2$ holds, and can reject otherwise.

We cannot contract all pairs in S simultaneously, as pairs may share a vertex, may be adjacent, or may have a common neighbor. Since all these properties are first-order definable, a circuit of the desired size and depth can easily check for each pair of reduction partners if they are in conflict. By doing so, the circuit can compute a *conflict graph* C whose node set is S and whose edges indicate conflicts. As the degree of each vertex appearing in a pair in S is bounded by d, the degree of C is bounded by $g(k)$ for a computable function g.

Since each maximal independent set I in a graph of maximum degree Δ has size at least $|V|/(\Delta + 1)$, it is sufficient to use the reduction partners that constitute a maximal independent set in C. The circuit can compute such a set using Lemma 4. ◀

Claim of Lemma 9. *There is a* DLOGTIME*-uniform family of* NC*-circuits of depth* $f(k) \cdot \log |V|$ *and size* $f(k) \cdot |V|^c$ *that, on input of a graph* $G = (V, E)$*, a set of pairs of vertices* I*, a graph* $G' = (V', E')$ *that is obtained from* G *by contracting the pairs in* I*, and a tree decomposition* (T', ι') *of* G' *of width* k*, outputs a balanced and nice tree decomposition* (T, ι, η) *of* G *of width at most* $8k + 3$ *and depth* $(16k + 6) \cdot \log |V| + 1$*.*

Proof. Let (T', ι') be the given tree decomposition. An AC^0-circuit can compute (T, ι) by adding for each pair $\{u, v\} \in I$ the vertex v to every bag that contains u. This can be done in parallel for all vertices and all bags. Since the number of vertices in each bag is at most doubled, (T, ι) has width at most $2k$.

This decomposition can be transformed into a balanced one of width at most $8k + 3$ by a TC^0-circuit [7]. The last thing we have to do is to transform this decomposition into a nice decomposition (T, ι, η). In order to do so, the circuit first adds an empty bag to each leaf, which is labeled as *leaf node*. Then, each node n with two children x and y is replaced by nodes n, n_l, and n_r such that n_l, n_r are the children of n, x is a child of n_l, and y a child of n_r. The node n is labeled as *join node*. This operation doubles the depth of the decomposition. Finally, for every node x with child y, the circuit computes a chain of *forget nodes* from x to a new node z with $\iota(x) \cap \iota(y) = \iota(z)$, and a chain of *introduce nodes* from z to y. This will increase the depth of the decomposition at most by a factor of $8k + 3$.

Since making a balanced tree decomposition nice will result in a balanced decomposition again, the above algorithm produced a nice, balanced tree decomposition of width at most $2k$ and depth at most $(16k + 6) \log |V| + 1$. ◀

Claim of Lemma 10. *There is a* DLOGTIME*-uniform family of* NC*-circuits of depth* $f(k) \cdot \log |V|$ *and size* $f(k) \cdot |V|^c$ *that, on input of a graph* $G = (V, E)$*, an integer* k*, and of a balanced and nice tree decomposition* (T, ι, η) *of* G *of width at most* $\ell \leq f(k)$*, outputs either* $\mathrm{tw}(G) > k$ *or a width-*k *tree decomposition of* G*.*

Proof. The original algorithm by Bodlaender and Hagerup [2] computes a path labeled tree representation of a tree decomposition of width k of G, or correctly detects $\mathrm{tw}(G) > k$. This algorithm "bubbles up" the nice tree decomposition and spends $f(k)$ time on every node. Since the depth of the tree is $f(k) \log |V|$, the desired circuit can implement this algorithm without modification.

After the execution of the above algorithm, the circuit may either reject if the algorithm reports that the tree width exceeds k, or obtains a path labeled tree representation. Recall that this implicit representation is a labeled binary tree T where exactly two nodes are labeled with each vertex of G. The idea is that the unique path between these two nodes

defines the bags in which the vertex (used as label) lies. Since the "real" tree decomposition we try to extract from this implicit representation uses the same tree, the rest of the lemma boils down to the following algorithmic task: Given a tree $T = (V, E)$ and three nodes $s, x, t \in V$, decide whether or not x lies on the unique path between s and t. This property is clearly expressible in MSO and, since T is a tree (of tree width 1), decidable in NC^1 [7]. ◀

Claim of Lemma 14. *Let $S_{k,d}$ be the set of labeled trees $(\mathfrak{T}, \lambda)$ of maximal depth d and maximal degree k. There is a* DLOGTIME*-uniform family of circuits over the standard base (only* AND*-,* OR*-, and* NOT*-gates) with fan-in k, depth $f(|\mathfrak{A}|) \cdot d$ and size $f(|\mathfrak{A}|) \cdot |\mathfrak{T}|^c$ that, on input of a labeled tree $(\mathfrak{T}, \lambda) \in S_{k,d}$ and a multiset automata $\mathfrak{A} = (\Sigma, Q, Q_a, \Delta, m)$, decides whether or not $(\mathfrak{T}, \lambda) \in \mathrm{L}(\mathfrak{A})$ holds.*

Proof. Since both, the depth and the size of the circuit, depend on the size of $\mathfrak{A}$ by an arbitrary computable function f, we can assume that $\mathfrak{A}$ is deterministic, since if this is not the case we can compute an equivalent deterministic automaton using Fact 13. The circuit has d layers, each of which consists of circuits of depth $f(|\mathfrak{A}|)$. The i-th layer will assign states to the nodes of the $(d-i)$-th layer of $\mathfrak{T}$. The first layer simply assigns states to the leafs of $\mathfrak{T}$. Layer i then has access to the assigned states of layer $i-1$. In order to compute the state $q(n)$ for a node n the circuit computes the multiset $M = \{\, q(c) \mid c \text{ is a child of } n \,\}$ using the result of the last layer. Now the circuit has to cap M to compute $M|_m$. In order to do so, the circuit has to count up to m. Since we have $m \leq |\mathfrak{A}|$, the value m is bounded by the parameter and, therefore, a para-AC^0 layer is sufficient for this task [1]. Once $M|_m$ is computed, the circuit can compute $q(n)$ by a lookup of $(\lambda(n), M|_m)$ in the description of δ. The circuit outputs 1 if, and only if, after the evaluation of the d layers the root r of $\mathfrak{T}$ is assigned with $q(r) \in Q_a$.

Clearly, the depth of the circuit is bounded by $f(|\mathfrak{A}|) \cdot d$. To see that the fan-in of the circuit does only depend on the maximal degree of $\mathfrak{T}$, observe the following: The subcircuit of a layer computing $q(n)$ for a node n has size bounded by $f(|\mathfrak{A}|)$ and, hence, can be replaced by a circuit of fan-in two without violating the depth bound of $f(|\mathfrak{A}|)$. The bigger fan-in is only needed to transmit the multiset $M = \{\, q(c) \mid c \text{ is a child of } n \,\}$ to the subcircuit computing $q(n)$, but since we have $|M| \leq k$ the claim follows. ◀

Claim of Theorem 23. *p_d-*ADISTANCE *is complete for* para-$\mathrm{AC}^{0\uparrow}$ *under* para-AC^0*-reduction.*

Proof. For containment consider a circuit that performs a backward breadth-first search starting at t, similar to Lemma 6. The circuit handles the graph in d layers, computing in layer i the vertices that have alternating distance i to t. In the first layer, vertex t is colored. In layer i, all vertices $x \in V_\exists$ that have one colored neighbor, and all $y \in V_\forall$ that have only colored neighbors (and at least one) are colored. There is an alternating path of distance at most d from s to t if, and only if, s is colored after d layers. The correctness of the circuit follows by a simple induction: in layer 1 we color exactly the vertices with alternating distance 1, and it is easy to see that coloring a vertex in layer i is only possible if it has a neighbor (or all its neighbors) with alternating distance $i-1$.

For completeness let us reduce any problem $L \in$ para-$\mathrm{AC}^{0\uparrow}$ to p_d-ADISTANCE. As L is in para-$\mathrm{AC}^{0\uparrow}$, there is a fixed family of circuits deciding L. Let C be such a circuit. We may assume that C is monotone since one can always replace a non-monotone circuit by a monotone one (using the standard argument used in showing that the circuit value problem reduces to its monotone version).

We translate the monotone circuit C into an alternating graph as follows: The vertices of the graph are the gates, and the wires are edges directed from the unique output gate towards the input bits. For each input bit there is a vertex as well. We label the output gate as s, add a new vertex t, and we add edges from all input bits that are set to 1 towards t. We partition the vertices such that $V_\exists$ is the set of OR -gates joined by t and the input bits; and such that $V_\forall$ is the set of AND -gates. The constructed graph with s and t, and with d as distance, constitutes an instance of p_d-ADISTANCE. An alternating path from s to t corresponds to wires that are set to true during the evaluation of the circuit and, hence, such a path can only exist if the circuit evaluates to true. Since, furthermore, the depth of the circuit is bounded by d, such a path has length at most d as well. ◀

Finding Secluded Places of Special Interest in Graphs*†

René van Bevern[‡1], Till Fluschnik[§2], George B. Mertzios[3], Hendrik Molter[¶4], Manuel Sorge[5], and Ondřej Suchý[‖6]

1 **Novosibirsk State University, Novosibirsk, Russian Federation; and Sobolev Institute of Mathematics, Siberian Branch of the Russian Academy of Sciences, Novosibirsk, Russian Federation**
rvb@nsu.ru

2 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
till.fluschnik@tu-berlin.de

3 **School of Engineering and Computing Sciences, Durham University, UK**
george.mertzios@durham.ac.uk

4 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
h.molter@tu-berlin.de

5 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
manuel.sorge@tu-berlin.de

6 **Faculty of Information Technology, Czech Technical University in Prague, Czech Republic**
ondrej.suchy@fit.cvut.cz

Abstract

Finding a vertex subset in a graph that satisfies a certain property is one of the most-studied topics in algorithmic graph theory. The focus herein is often on minimizing or maximizing the size of the solution, that is, the size of the desired vertex set. In several applications, however, we also want to limit the "exposure" of the solution to the rest of the graph. This is the case, for example, when the solution represents persons that ought to deal with sensitive information or a segregated community. In this work, we thus explore the (parameterized) complexity of finding such *secluded* vertex subsets for a wide variety of properties that they shall fulfill. More precisely, we study the constraint that the (open or closed) neighborhood of the solution shall be bounded by a parameter and the influence of this constraint on the complexity of minimizing separators, feedback vertex sets, $\mathcal{F}$-free vertex deletion sets, dominating sets, and the maximization of independent sets.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Neighborhood, Feedback Vertex Set, Vertex Deletion, Separator, Dominating Set

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.5

* A full version of the paper is available at https://arxiv.org/abs/1606.09000.

† This research was initiated at the annual research retreat of the algorithms and complexity group of TU Berlin, held in Krölpa, Thuringia, Germany, from April 3rd till April 9th, 2016.

‡ Results in Section 4 were obtained under support of the Russian Science Foundation, grant 16-11-10041.

§ Supported by the DFG, project DAMM (NI 369/13).

¶ Supported by the DFG, project DAPA (NI 369/12).

‖ Supported by grant 14-13017P of the Czech Science Foundation.

1 Introduction

In many optimization problems on graphs, one searches for a minimum or maximum cardinality subset of vertices and edges satisfying certain properties, like a minimum s-t path, a maximum independent set, or a minimum dominating set. In several applications, however, it is important to also limit the *exposure* of the solution [5, 17]. For instance, we may want to find a way to send sensitive information that we want to protect from a vertex s to a vertex t in a network. If we assume that the information is exposed to all vertices on the way and all of their neighbors, limiting the exposure means to find an s-t path with a small closed neighborhood [5]. Another example is the search for segragated communities in social networks [17]. Herein, we search for dense subgraphs which are exposed to few neighbors in the rest of the graph. In addition to being a natural constraint in these applications, restricting the exposure of the solution may also yield more efficient algorithms [14, 16, 17, 18].

In accordance with previous work, we call a solution *secluded* if it has a small exposure [5]. Secluded paths and Steiner trees have been studied before [5, 11]. Our aim in this paper is to study the constraint of being secluded on the complexity of diverse vertex-subset optimization problems.

Inspired by Chechik et al. [5], we first measure the exposure of a solution S by the size of the closed neighborhood $N_G[S] = S \cup \bigcup_{v \in S} N_G(v)$ of S in the input graph G. Given a predicate $\Pi(G, S)$ that determines whether S is a solution for input graph G, we hence study the following problem.

Secluded Π
Input: A graph $G = (V, E)$ and an integer k.
Question: Is there a subset $S \subseteq V$ of vertices such that S satisfies $\Pi(G, S)$ and $|N_G[S]| \leq k$?

It makes sense to also control the size of the solution and its neighborhood in the graph directly. For example, when sending sensitive information from s to t as above, we may simultaneously aim to optimize latency, that is, minimize the number of vertices in the communication path and limit the exposure. Hence, our second measure of exposure of the solution is the size of the open neighborhood $N_G(S) = N_G[S] \setminus S$. This leads to the following problem formulation.

Small (Large) Secluded Π
Input: A graph $G = (V, E)$ and two integers k, ℓ.
Question: Is there a subset $S \subseteq V$ of vertices of G such that S satisfies $\Pi(G, S)$, $|S| \leq k$, and $|N_G(S)| \leq \ell$ (resp. $|S| \geq k$, and $|N_G(S)| \leq \ell$)?

We study both problems in the framework of parameterized complexity. As a parameter for Secluded Π we use the size k of the closed neighborhood and as parameters for Small Secluded Π we use the size k of the solution as well as the size ℓ of the open neighborhood.

The predicates $\Pi(G, S)$ that we study are s-t Separator, Feedback Vertex Set (FVS), $\mathcal{F}$-free Vertex Deletion ($\mathcal{F}$-FVD) (for an arbitrary finite family $\mathcal{F}$ of graphs), encompassing Cluster Vertex Deletion, for example, and Independent Set (IS). Perhaps surprisingly, we find that Secluded s-t-Separator is polynomial-time solvable, whereas Small Secluded s-t Separator becomes NP-complete. The remaining problems are NP-complete. For them, roughly speaking, we prove that fixed-parameter tractability results for Π parameterized by the solution size carry over to Secluded Π parameterized by k. For Small Secluded Π parameterized by ℓ, however, we mostly obtain W[1]-hardness. On the positive side, for Small Secluded $\mathcal{F}$-FVD we prove fixed-parameter tractability when parameterized by $k + \ell$.

Table 1 Overview of our results. PK stands for polynomial kernel. The results marked by an asterisk follow by a straightforward reduction from the non-secluded variant.

Problem	Complexity		Parameterized Complexity / Kernelization			
	Secluded	Small Secl.	Secluded	Small Secluded		
			k	k	ℓ	$k+\ell$
s-t-separator	P	NP-c.	P	W[1]-h.	W[1]-h.	?/noPK
$\mathcal{F}$-free VD	NP-c.	NP-c.*	FPT/PK	?	?	FPT/?
FVS	NP-c.	NP-c.*	FPT/PK	?	W[1]-h.	? / ?
q-DS, $2p \leq q$	NP-c.	NP-c.*	W[2]-h.	$\rightarrow$	$\rightarrow$	W[2]-h.
q-DS, $2p > q$	NP-c.	NP-c.*	FPT/noPK	?	?	FPT/noPK
Large Secluded IS	NP-c.*			$\rightarrow$	$\rightarrow$	W[1]-h.

We also study, for two integers $p < q$, the p-secluded version of q-Dominating Set (q-DS): a vertex set S is a *q-dominating set* if every vertex of $V \setminus S$ has distance at most q to some vertex in S. Herein, by p-secluded we mean that we upper bound the size of the distance-p-neighborhood of the solution S. Interestingly, this problem admits a complexity dichotomy: Whenever $2p > q$, (Small) p-Secluded q-Dominating Set is fixed-parameter tractable with respect to k (with respect to $k + \ell$), but it is W[2]-hard otherwise.

We also study polynomial-size problem kernels for our secluded problems. Here we observe that the polynomial-size problem kernels for Feedback Vertex Set and $\mathcal{F}$-free Vertex Deletion carry over to their Secluded variants, but otherwise we obtain mostly absence of polynomial-size problem kernels unless the polynomial hierarchy collapses.

A summary of our results is given in Table 1.

Related work. Secluded Path and Secluded Steiner Tree were introduced and proved NP-complete by Chechik et al. [5]. They obtained approximation algorithms for both problems with approximation factors related to the maximum degree. They also showed that Secluded Path is fixed-parameter tractable with respect to the maximum vertex degree of the input graph, whereas vertex weights lead to NP-hardness for maximum degree four.

Fomin et al. [11] studied the parameterized complexity of Secluded Path and Secluded Steiner Tree, showing that both are fixed-parameter tractable even in the vertex-weighted setting. Furthermore, they showed that Secluded Steiner Tree is fixed-parameter tractable with respect to $r+p$, where $r = k-s$, k is the desired size of the closed neighborhood of the solution, s is the size of an optimum Steiner tree, and p is the number of terminals. On the other hand this problem is co-W[1]-hard when parameterized by r only.

The small secluded concept can be found in the context of cut problems in graph [21, 12]. Fomin et al. [12] introduced the Cutting at Most k Vertices problem, which asks, given a graph $G = (V, E)$ and two integers $k \geq 1$ and $\ell \geq 0$, whether there is a non-empty set $S \subseteq V$ such that $|S| \leq k$, and $|N_G(X)| \leq \ell$, thus resembling our small secluded concept. Both works [12, 21] study the parameterized complexity of related cut problems in graphs.

The concept of isolation can be found in the context of cuts [7] and was thoroughly explored for finding dense subgraphs [14, 16, 17, 18]. Herein, chiefly the constraint that the vertices in the solution shall have maximum/minimum/average outdegree bounded by a parameter was considered [14, 17, 18], leading to various parameterized tractability and hardness results. Also the overall number of edges outgoing the solution has been studied recently [16].

Preliminary observations. Concerning classical computational complexity, the Small (Large) Secluded variant of a problem is at least as hard as the nonsecluded problem, by a simple reduction in which we set $\ell = n$, where n denotes the number of vertices in the graph. Since this reduction is a parameterized reduction with respect to k, parameterized hardness results for this parameter transfer, too. Furthermore, observe that hardness also transfers from Secluded Π to Small Secluded Π for all problems Π, since Secluded Π allows for a parameterized Turing reduction to Small Secluded Π: try out all k' and ℓ' with $k = k' + \ell'$. Additionally, many tractability results (in particular polynomial time solvability and fixed-parameter tractability) transfer from Small Secluded Π parameterized by $(k+\ell)$ to Secluded Π parameterized by k.

▶ **Observation 1.1.** Secluded Π *parameterized by* k *is parameterized Turing reducible to* Small Secluded Π *parameterized by* $(k + \ell)$ *for all predicates* Π.

Therefore, for the Small (Large) Secluded variants of the problems the interesting cases are those where the base problem is tractable (deciding whether input graph G contains a vertex set S of size k that satisfies $\Pi(G, S)$) or where ℓ is a parameter.

Notation. We use standard notation from parameterized complexity and graph theory. All graphs in this paper are undirected. We denote $d_G(u, v)$ the *distance* between vertices u and v in G, that is, the number of edges of a shortest u-v path in G. For a set V' of vertices and a vertex $v \in V$ we let the distance of v from V' be $d_G(v, V') := \min\{d_G(u, v) \mid u \in V'\}$. We use $N^d_G[V'] = \{v \mid d_G(v, V') \leq d\}$ and $N^d_G(V') = N^d_G[V'] \setminus V'$ for any $d \geq 0$ (hence $N^0_G(V') = \emptyset$). We omit the index if the graph is clear from context and also use $N[V']$ for $N^1[V']$ and $N(V')$ for $N^1(V')$. If $V' = \{v\}$, then we write $N^d[v]$ in place of $N^d[\{v\}]$.

Organization. We dedicate one section to each studied problem. We study s-t-Separator in Section 2, q-Dominating Set in Section 3, $\mathcal{F}$-free Vertex Deletion in Section 4, Feedback Vertex Set in Section 5, and Independent Set in Section 6. Section 7 summarizes results and gives directions for future research. We remark that some proofs and proof details (marked with $(\star)$) are deferred to a full version of this paper.

2 s-t-Separator

In this section, we show that Secluded s-t-Separator is in P, while Small Secluded s-t-Separator is NP-hard and $W[1]$-hard with respect to the size of the open neighborhood, or the size of the solution. Moreover, we also exclude the existence of polynomial-size kernels for the latter problem with respect to the sum of the bounds.

2.1 Secluded s-t-Separator

In this subsection we show that the following problem can be solved in polynomial time.

Secluded s-t-Separator
Input: A graph $G = (V, E)$, two distinct vertices $s, t \in V$, and an integer k.
Question: Is there an s-t separator $S \subseteq V \setminus \{s, t\}$ such that $|N_G[S]| \leq k$?

▶ **Theorem 2.1.** Secluded s-t-Separator *can be solved in polynomial time.*

Proof. We reduce the problem to the problem of finding an ordinary s-t separator in an auxiliary graph. Let $(G = (V, E), s, t, k)$ be the input instance and G'' be a graph obtained

from G by adding two vertices s' and t' and making s' only adjacent to s and t' only adjacent to t'. Now let $G' = (V', E')$ be the third power of G'', that is, $V' = V(G') = V(G'') = V \cup \{s', t'\}$ and $\{u, v\} \in E'$ if and only if $d_{G''}(u, v) \leq 3$.

We claim that there is an s-t-separator S in G with $|N[S]| \leq k$ if and only if there is an s'-t'-separator S' in G' with $|S'| \leq k$. The theorem then follows as we can construct G' and find the minimum s'-t'-separator in G' in polynomial time using standard methods, for example, based on network flows.

"$\Rightarrow$": Let S be an s-t-separator in G with $|N[S]| \leq k$. Observe that S then also constitutes an s'-t'-separator in G'' as every path in G'' from s' must go through s and every path to t' must go through t. We claim that $S' = N[S]$ is an s'-t'-separator in G'. Suppose for contradiction that there is an s'-t' path $P = p_0, p_1, \ldots, p_q$ in $G' - S'$. Let A' be the set of vertices of the connected component of $G'' - S$ containing s' and let a be the last index such that $p_a \in A'$ (note that $p_0 = s' \in A'$ and $p_q = t' \notin A'$ by definition). It follows that $p_{a+1} \notin A'$ and, since $\{p_a, p_{a+1}\} \in E'$, there is a p_a-p_{a+1} path P' in G'' of length at most 3. As we have $p_a \in A'$ and $p_{a+1} \in V \setminus (A' \cup S')$ and $G[A']$ is a connected component of $G'' - S$, there must be a vertex x of S on P'. Since neither p_a nor p_{a+1} is in $S' = N[S]$, it follows that $d_G(p_a, x) \geq 2$ and $d_G(p_{a+1}, x) \geq 2$. This contradicts P' having length at most 3.

"$\Leftarrow$": Let S' be an s'-t'-separator in G' of size at most k. Let A' be the vertex set of the connected component of $G' - S'$ containing s'. Consider the set $S = \{v \in S' \mid d_{G''}(v, A') = 2\}$. We claim that S is an s-t-separator in G and, moreover, that $N[S] \subseteq S'$ and, hence, $|N[S]| \leq k$. As to the second part, we have $S \subseteq S'$ by definition. Suppose for contradiction that there was a vertex $u \in N(S) \setminus S'$ that is a neighbor of $v \in S$. Then, since $d_{G''}(v, A') = 2$, we have $d_{G''}(u, A') \leq 3$, u has a neighbor in A' in G', and, thus u is in A'. This implies that $d_{G''}(v, A') = 1$, a contradiction. Hence, $|N[S]| \leq k$.

It remains to show that S is an s-t-separator in G. For this, we prove that S is an s'-t'-separator in G''. Since it contains neither s nor t, it follows that it must be also an s-t-separator in G. Assume for contradiction that there is an s'-t' path in $G'' - S$. This implies that $d_{(G''-S)}(t', A')$ is well defined (and finite). Let $q := d_{(G''-S)}(t', A')$ and P be the corresponding shortest path in $G'' - S$. Let us denote $P = p_0, \ldots, p_q$ with $p_q = t'$ and $p_0 \in A'$. If $d_{G''}(t', A') \leq 3$, then t' has a neighbor in A' in G', and therefore it is in A' contradicting our assumption that S' is an s'-t'-separator in G'. As $t' = p_q$, we have $q > 3$. Since $d_{G''}(p_0, A') = 0$, $d_{G''}(p_q, A') > 3$, and $d_{G''}(p_{i+1}, A') \leq d_{G''}(p_i, A') + 1$ for every $i \in \{0, \ldots q-1\}$, there is an a such that $d_{G''}(p_a, A') = 2$. If p_a is not in S', then p_a is in A', contradicting our assumptions on P and q as $a \geq 2$. Therefore we have $d_{G''}(p_a, A') = 2$ and p_a is in S'. It follows that p_a is in S, a contradiction. ◀

2.2 Small Secluded s-t-Separator

In this subsection we prove hardness results for the following problem.

Small Secluded s-t-Separator
Input: A graph $G = (V, E)$, two distinct vertices $s, t \in V$, and two integers k, ℓ.
Question: Is there an s-t separator $S \subseteq V \setminus \{s, t\}$ such that $|S| \leq k$ and $|N_G(S)| \leq \ell$?

We show that, in contrast to Secluded s-t-Separator, the above problem is NP-hard. Moreover, at the same time, we show parameterized hardness with respect to k and with respect to ℓ.

▶ **Theorem 2.2.** Small Secluded s-t-Separator *is NP-hard and $W[1]$-hard when parameterized by k or by ℓ.*

In the proof of the theorem, we reduce from the Cutting at Most k Vertices with Terminal [12] problem, which asks, given a graph $G = (V, E)$, a vertex $s \in V$, and two integers $k \geq 1$, $\ell \geq 0$, whether there is a set $S \subseteq V$ such that $s \in S$, $|S| \leq k$, and $|N_G(X)| \leq \ell$. Fomin et al. [12] proved that Cutting at Most k Vertices with Terminal is NP-hard and W[1]-hard when parameterized by k or by ℓ.

Proof. We give a polynomial-parameter transformation from Cutting at Most k Vertices with Terminal to Small Secluded s-t-Separator.

Construction. Let $\mathcal{I} := (G = (V, E), s, k, \ell)$ be an instance of Cutting at Most k Vertices with Terminal. We construct an instance $\mathcal{I}' := (G', s', t', k', \ell')$ of Small Secluded s-t-Separator equivalent to $\mathcal{I}$ as follows. To obtain G' from G we add to G two vertices s' and t' and two edges $\{s', s\}$ and $\{s, t'\}$. Note that $G = G' - \{s', t'\}$. We set $k' = k$ and $\ell' = \ell + 2$. Hence, we ask for an s'-t' separator $S \subseteq V(G')$ in G' of size at most k' and $|N_{G'}(S)| \leq \ell'$. Clearly, the construction can be carried out in polynomial time.

Correctness. We show that $\mathcal{I}$ is a yes-instance of Cutting at Most k Vertices with Terminal if and only if $\mathcal{I}'$ is a yes-instance of Small Secluded s-t-Separator.

"$\Rightarrow$": Let $\mathcal{I}$ be a yes-instance and let $S \subseteq V(G)$ be a solution to $\mathcal{I}$, that is, $s \in S$, $|S| \leq k$, and $|N_G(S)| \leq \ell$. We claim that S is also a solution to $\mathcal{I}'$. Since $s \in S$ and s' and t' are both only adjacent to s, S separates s' from t' in G'. Moreover, $|S| \leq k = k'$ and, as $N_{G'}(S) = N_G(S) \cup \{s', t'\}$, we have $|N_{G'}(S)| \leq \ell + 2 = \ell'$. Hence, S' is a solution to $\mathcal{I}'$, and $\mathcal{I}'$ is a yes-instance.

"$\Leftarrow$": Let $\mathcal{I}'$ be a yes-instance and let $S' \subseteq V(G')$ be an s'-t' separator in G' with $|S'| \leq k'$ and $|N_{G'}(S')| \leq \ell'$. We claim that S' is also a solution to $\mathcal{I}$. Note that $|S'| \leq k' = k$. Since S' is an s'-t' separator in G' and s' and t' are both adjacent to s, it follows that $s \in S'$ and $s', t' \in N_{G'}(S')$. Thus, we have $s \in S'$ and $|N_G(S')| = |N_{G'-\{s',t'\}}(S')| = |N_{G'}(S')| - 2 \leq \ell' - 2 = \ell$. Hence, S' is a solution to $\mathcal{I}$ and $\mathcal{I}$ is a yes-instance.

Note that, in the reduction, k' and ℓ' only depend on k and ℓ, respectively. Since Cutting at Most k Vertices with Terminal parameterized by k or by ℓ is W[1]-hard [12], it follows that Small Secluded s-t-Separator parameterized by k or by ℓ is W[1]-hard. ◀

It would be interesting to know whether Small Secluded s-t-Separator is FPT when parameterized by $k + \ell$. We conjecture that this is the case. However, under standard assumptions, the problem does not admit a polynomial-size kernel with respect to this parameter:

▶ **Theorem 2.3** (⋆). *Unless NP $\subseteq$ coNP/poly,* Small Secluded s-t-Separator *parameterized by $k + \ell$ does not admit a polynomial kernel.*

3 q-Dominating Set

In this section, for two constants $p, q \in \mathbb{N}$ with $0 \leq p < q$, we study the following problems:

p-Secluded q-Dominating Set
Input: A graph $G = (V, E)$ and an integer k.
Question: Is there a set $S \subseteq V$ such that $V = N_G^q[S]$ and $|N_G^p[S]| \leq k$?

Small p-Secluded q-Dominating Set
Input: A graph $G = (V, E)$ and two integers k, ℓ.
Question: Is there a set $S \subseteq V$ such that $V = N_G^q[S]$, $|S| \leq k$, and $|N_G^p(S)| \leq \ell$?

For $p = 0$, the size restrictions in both cases boil down to $|S| \leq k$. This is the well-known case of q-Dominating Set (also known as q-Center) which is NP-hard and W[2]-hard with respect to k (see Lokshtanov et al. [20], for example). Therefore, for the rest of the section we focus on the case $p > 0$. Additionally, by a simple reduction from q-Dominating Set, letting $\ell = |V(G)|$, we arrive at the following observation.

▶ **Observation 3.1.** *For any $0 < p < q$,* Small p-Secluded q-Dominating Set *is* W[2]*-hard with respect to k.*

We now go on to show NP-hardness and W[2]-hardness with respect to k for p-Secluded q-Dominating Set. We reduce from the following problem:

Set Cover
Input: A finite universe U, a family $F \subseteq 2^U$, and an integer k.
Question: Is there a subset $X \subseteq F$ such that $|X| \leq k$ and $\bigcup_{x \in X} x = U$?

We write $\bigcup X$ short for $\bigcup_{x \in X} x$. It is known that Set Cover is NP-complete, W[2]-hard with respect to k, and admits no polynomial kernel with respect to $|F|$, unless NP $\subseteq$ coNP/poly [6].

▶ **Theorem 3.2.** *For any $0 < p < q$, p-Secluded q-Dominating Set is NP-hard. Moreover, it does not admit a polynomial kernel with respect to k, unless NP $\subseteq$ coNP/poly.*

Proof. We give a polynomial-parameter transformation from Set Cover parameterized by $|F|$. Let (U, F, k) be an instance of Set Cover. Without loss of generality we assume that $0 \leq k < |F|$.

Construction. Let $k' = p + 1 + |F| \cdot p + k$. We construct the graph G of a p-Secluded q-Dominating Set instance (G, k') as follows. We start the construction by taking two vertices s and r and three vertex sets $V_U = \{u \mid u \in U\}$, $V_F = \{v_A \mid A \in F\}$, and $V'_F = \{v'_A \mid A \in F\}$. We connect vertex r with vertex s by a path of length exactly q. For each $A \in F$ we connect vertices v_A and r by an edge and vertices v_A and v'_A by a path $t^A_0, t^A_1, \ldots, t^A_p$ of length exactly p, where $t^A_0 = v_A$ and $t^A_p = v'_A$. All introduced paths are internally disjoint and the internal vertices are all new. We connect a vertex $v'_A \in V'_F$ with a vertex $u \in V_U$ by an edge if and only if $u \in A$. Furthermore, we introduce a clique C_U of size k' and make all its vertices adjacent to each vertex in $V'_F \cup V_U$.

If $q - p \geq 2$, then for each $u \in U$, we create a path $b^u_0, b^u_1, \ldots, b^u_{q-p-2}$ of length exactly $q - p - 2$ such that $b^u_0 = u$ and the other vertices are new. Furthermore, in this case, for each $h \in \{0, \ldots, q - p - 2\}$ we introduce a clique C^u_h of size k' and make all its vertices adjacent to vertex b^u_h. If $q - p = 1$ we do not introduce any new vertices.

One can show that the original instance of Set Cover is a yes-instance if and only if the constructed instance of p-Secluded q-Dominating Set is. We defer the proof of the equivalence to a full version of the paper. ◀

In the following, we observe that the parameterized complexity of both problems varies for different choices of p and q.

▶ **Theorem 3.3** (⋆)**.** *For any $0 < p \leq \frac{1}{2}q$, p-Secluded q-Dominating Set is* W[2]*-hard with respect to k.*

For Small p-Secluded q-Dominating Set, we remark that we can adapt the reduction for Theorem 3.2: instead of restricting the size of the closed neighborhood of the q-dominating set to at most $p + 1 + |F| \cdot p + k$, we restrict the size of the q-dominating set to at most $k + 1$ and the size of its open neighborhood to at most $p + |F| \cdot p$. Analogously, we can adapt the reduction for Theorem 3.3. This yields the following hardness results.

▶ **Corollary 3.4.** *For any $0 < p < q$,* Small p-Secluded q-Dominating Set *is NP-hard. Moreover, it does not admit a polynomial kernel with respect to $(k+\ell)$ unless NP $\subseteq$ coNP/poly. For any $0 < p \leq \frac{1}{2}q$,* Small p-Secluded q-Dominating Set *is* W[2]*-hard with respect to $(k+\ell)$.*

Now we look at the remaining choices for p and q, that is all p, q with $p > \frac{1}{2}q$. In these cases we can show fixed-parameter tractability.

▶ **Theorem 3.5.** *For any $p > \frac{1}{2}q$,* Small p-Secluded q-Dominating Set *can be solved in $O(mk^{k+2}(k+\ell)^{qk})$ time and, hence, it is fixed-parameter tractable with respect to $k+\ell$.*

Proof. Consider a solution S for an instance (G, k, ℓ) of Small p-Secluded q-Dominating Set. If $x \in S$, then $|N^p[x]| \leq k+\ell$, since $|S| \leq k$ and $|N^p(S)| \leq \ell$. Moreover, $|N^p[x]| \leq k+\ell$ implies $|N[v]| \leq k+\ell$ for every $v \in N^{p-1}[x]$. It follows that, if $|N^p[y]| \leq k+\ell$ and $y \notin S$, then for each $x \in N^q[y] \cap S$ every vertex on every x-y path of length at most $2p-1 \geq q$ has degree at most $k+\ell-1$, since each such vertex has distance at most $p-1$ to x or y.

If $k+\ell = 1$, then either G has at most one vertex or (G, k, ℓ) is a no-instance. Hence, in the following, we assume $k+\ell \geq 2$. We call vertices u and v *linked*, if there is a path of length at most q between u and v in G such that the degree of every vertex on the path is at most $k+\ell-1$. Let $B[u] = \{v \mid u \text{ and } v \text{ are linked}\}$. One can show $|B[v]| \leq (k+\ell)^q$ for any v (we defer the proof to a full version of the paper).

Let $Y = \{y \mid |N^p[y]| \leq k+\ell\}$. Obviously, we have $S \subseteq Y$, since $|N^p[S]| \leq k+\ell$. If $y \in Y \setminus S$, then there is $x \in S$ such that x and y are linked. It follows that $y \in B[x]$ and, thus, $Y \subseteq \bigcup_{x \in S} B[x]$. Hence, $|Y| \leq k \cdot (k+\ell)^q \leq (k+\ell)^{q+1}$.

This suggests the following algorithm for Small p-Secluded q-Dominating Set: Find the set Y. If $|Y| > k \cdot (k+\ell)^q$, then answer "no". Otherwise, for each $k' \leq k$ and each size-k' subset S' of Y, check whether S' is a p-secluded q-dominating set in G. If any such set is found, return it. Otherwise, answer "no". Since $S \subseteq Y$, this check is exhaustive.

As to the running time, the set Y can be determined in $O(n(k+\ell))$ time by running a BFS from each vertex and stopping it after it discovers $k+\ell$ vertices or all vertices in distance at most p, whichever occurs earlier. Then, there are $k \cdot \binom{k \cdot (k+\ell)^q}{k} \leq k^{k+1}(k+\ell)^{qk}$ candidate subsets of Y. For each such set S' we can check whether it is a p-secluded q-dominating set in G by running a BFS from each vertex of S' and marking the vertices which are in distance at most p and at most q, respectively. This takes $O(mk)$ time. Hence, in total, the algorithm runs in $O(mk^{k+2}(k+\ell)^{qk})$ time. ◀

By Observation 1.1, the previous result transfers to p-Secluded q-Dominating Set parameterized by k.

▶ **Corollary 3.6.** *For any $p > \frac{1}{2}q$, p-*Secluded q-Dominating Set *is fixed-parameter tractable with respect to k.*

4 $\mathcal{F}$-free Vertex Deletion

In this section, we study the $\mathcal{F}$-free Vertex Deletion ($\mathcal{F}$-FVD) problem for families $\mathcal{F}$ of graphs with at most a constant number c of vertices, that is, the problem of destroying all induced subgraphs isomorphic to graphs in $\mathcal{F}$ by at most k vertex deletions. The problem can, in particular, model various graph clustering tasks [3, 15], where the secluded variants can be naturally interpreted as removing a small set of outliers that are weakly connected to the clusters.

4.1 Secluded $\mathcal{F}$-free Vertex Deletion

In this section, we prove a polynomial-size problem kernel for Secluded $\mathcal{F}$-free Vertex Deletion, where $\mathcal{F}$ is a family of graphs with at most a constant number c of vertices:

Secluded $\mathcal{F}$-free Vertex Deletion
Input: A graph $G = (V, E)$ and an integer k.
Question: Is there a set $S \subseteq V$ such that $G - S$ is $\mathcal{F}$-free and $|N_G[S]| \leq k$?

Henceforth, we call a set $S \subseteq V$ such that $G - S$ is $\mathcal{F}$-free an *$\mathcal{F}$-free vertex deletion set*.

Note that Secluded $\mathcal{F}$-free Vertex Deletion can be polynomial-time solvable for some families $\mathcal{F}$ for which $\mathcal{F}$-FVD is NP-hard: Vertex Cover (where $\mathcal{F}$ contains only the graph consisting of a single edge) is NP-hard, yet any vertex cover S satisfies $N[S] = V$. Therefore, an instance to Secluded Vertex Cover is a yes-instance if and only if $k \geq n$. In general, however, one can show that Secluded $\mathcal{F}$-Free Vertex Deletion is NP-complete for every family $\mathcal{F}$ that includes only graphs of minimum vertex degree two (Theorem 4.1). We mention in passing that, from this peculiar difference of the complexity of Vertex Cover and Secluded Vertex Cover, it would be interesting to find properties of $\mathcal{F}$ which govern whether Secluded $\mathcal{F}$-free Vertex Deletion is NP-hard or polynomial-time solvable along the lines of the well-known dichotomy results [9, 19].

▶ **Theorem 4.1** ($\star$). *For each family $\mathcal{F}$ containing only graphs of minimum vertex degree two,* Secluded $\mathcal{F}$-Free Vertex Deletion *is NP-complete.*

It is easy to see that Secluded $\mathcal{F}$-free Vertex Deletion is fixed-parameter tractable. More specifically, it is solvable in $c^k \cdot \text{poly}(n)$ time: simply enumerate all inclusion-minimal $\mathcal{F}$-free vertex deletion sets S of size at most k using the standard search tree algorithm described by Cai [4] and check $|N[S]| \leq k$ for each of them. This works because, for any $\mathcal{F}$-free vertex deletion set S with $|N[S]| \leq k$, we can assume that S is an inclusion-minimal $\mathcal{F}$-free vertex deletion set since $|N[S']| \leq |N[S]|$ for every $S' \subsetneq S$.

We complement this observation of fixed-parameter tractability by the following kernelization result.

▶ **Theorem 4.2.** Secluded $\mathcal{F}$-free Vertex Deletion *has a problem kernel comprising $O(k^{c+1})$ vertices, where c is the maximum number of vertices in any graph of $\mathcal{F}$.*

Our proof of Theorem 4.2 exploits *expressive* kernelization algorithms for d-Hitting Set [1, 2, 8], which preserve inclusion-minimal solutions and that return subgraphs of the input hypergraph as kernels: Herein, given a hypergraph $H = (U, \mathcal{C})$ with $|C| \leq d$ for each $C \in \mathcal{C}$, and an integer k, d-Hitting Set asks whether there is a *hitting set* $S \subseteq U$ with $|S| \leq k$, that is, $C \cap S \neq \emptyset$ for each $C \in \mathcal{C}$. Our kernelization for Secluded $\mathcal{F}$-free Vertex Deletion is based on transforming the input instance (G, k) to a d-Hitting Set instance (H, k), computing an expressive d-Hitting Set problem kernel (H', k), and outputting a Secluded $\mathcal{F}$-free Vertex Deletion instance (G', k), where G' is the graph induced by the vertices remaining in H' together with at most $k + 1$ additional neighbors for each vertex in G.

▶ **Definition 4.3.** Let $(G = (V, E), k)$ be an instance of Secluded $\mathcal{F}$-free Vertex Deletion. For a vertex $v \in V$, let $N_j(v) \subseteq N_G(v)$ be j arbitrary neighbors of v, or $N_j(v) := N_G(v)$ if v has degree less than j. For a subset $S \subseteq V$, let $N_j(S) := \bigcup_{v \in S} N_j(v)$. Moreover, let

- $c := \max_{F \in \mathcal{F}} |V(F)|$ be the maximum number of vertices in any graph in $\mathcal{F}$,
- $H = (U, \mathcal{C})$ be the hypergraph with $U := V$ and $\mathcal{C} := \{S \subseteq V \mid G[S] \in \mathcal{F}\}$,

- $H' = (U', \mathcal{C}')$ be a subgraph of H with $|U'| \in O(k^c)$ such that each set $S \subseteq U$ with $|S| \leq k$ is an inclusion-minimal hitting set for H if and only if it is for H', and
- $G' = (V', E')$ be the subgraph of G induced by $U' \cup N_{k+1}(U')$.

To prove Theorem 4.2, we show that (G', k) is a problem kernel for the input instance (G, k). The subgraph H' exists and is computable in linear time from H [2, 8]. Moreover, for constant c, one can compute H from G and G' from H' in polynomial time. It is obvious that the number of vertices of G' is $O(k^{c+1})$. Hence, it remains to show that (G', k) is a yes-instance if and only if (G, k) is. This is achieved by the following two lemmas.

▶ **Lemma 4.4.** *For any $S \subseteq U'$ with $|N_{G'}[S]| \leq k$, it holds that $N_G[S] = N_{G'}[S]$.*

Proof. Since $S \subseteq U' \subseteq V' \cap V$ and since G' is a subgraph of G, it is clear that $N_G[S] \supseteq N_{G'}[S]$. For the opposite direction, observe that each $v \in S$ has degree at most k in G'. Thus, v has degree at most k in G since, otherwise, $k+1$ of its neighbors would be in G' by construction. Thus, $N_{G'}(v) \supseteq N_{k+1}(v) = N_G(v)$ for all $v \in S$ and, thus, $N_{G'}[S] \supseteq N_G[S]$. ◀

▶ **Lemma 4.5 (⋆).** *Graph G allows for an $\mathcal{F}$-free vertex deletion set S with $|N_G[S]| \leq k$ if and only if G' allows for an $\mathcal{F}$-free vertex deletion set S with $|N_{G'}[S]| \leq k$.*

4.2 Small Secluded $\mathcal{F}$-free Vertex Deletion

In this subsection, we present a fixed-parameter algorithm for the following problem parameterized by $\ell + k$.

Small Secluded $\mathcal{F}$-free Vertex Deletion
Input: A graph $G = (V, E)$ and two integers k, ℓ.
Question: Is there a subset $S \subseteq V$ such that $G - S$ is $\mathcal{F}$-free, $|S| \leq k$, and $|N_G(S)| \leq \ell$?

As before, we call a set $S \subseteq V$ such that $G - S$ is $\mathcal{F}$-free an *$\mathcal{F}$-free vertex deletion set*.

In the previous section, we discussed a simple search tree algorithm for Secluded $\mathcal{F}$-free Vertex Deletion that was based on the fact that we could assume that our solution is an inclusion-minimal $\mathcal{F}$-free vertex deletion set. However, an $\mathcal{F}$-free vertex deletion set S with $|S| \leq k$ and $|N_G(S)| \leq \ell$ is not necessarily inclusion-minimal: some vertices may have been added to S just in order to shrink its open neighborhood. However, the following simple lemma limits the number of possible candidate vertices that can be used to enlarge S in order to shrink $N(S)$, which we will use in a branching algorithm.

▶ **Lemma 4.6.** *Let S be an $\mathcal{F}$-free vertex deletion set and $S' \supseteq S$ such that $|S'| \leq k$ and $|N_G(S')| \leq \ell$, then $|N_G(S)| \leq \ell + k$.*

Proof. $|N_G(S)| = |N_G[S] \setminus S| \leq |N_G[S'] \setminus S| \leq |N_G[S']| \leq |N_G(S') \cup S'| \leq \ell + k$. ◀

▶ **Theorem 4.7.** Small Secluded $\mathcal{F}$-free Vertex Deletion *can be solved in* $\max\{c, k+\ell\}^k \cdot \text{poly}(n)$*-time, where c is the maximum number of vertices in any graph of $\mathcal{F}$.*

Proof. First, enumerate all inclusion-minimal $\mathcal{F}$-free vertex deletion sets S with $|S| \leq k$. This is possible in $c^k \cdot \text{poly}(n)$ time using the generic search tree algorithm described by Cai [4]. For each $k' \leq k$, this search tree algorithm generates at most $c^{k'}$ sets of size k'. For each enumerated set S of k' elements, do the following:

1. If $|N_G(S)| \leq \ell$, then output S as our solution.
2. If $|N_G(S)| > \ell + k$, then S cannot be part of a solution S' with $N_G(S') \leq \ell$ by Lemma 4.6, we proceed with the next set.

3. Otherwise, initiate a recursive branching: recursively branch into at most $\ell+k$ possibilities of adding a vertex from $N_G(S)$ to S as long as $|S| \leq k$.

The recursive branching initiated at step 3 stops at depth $k - k'$ since, after adding $k - k'$ vertices to S, one obtains a set of size k. Hence, the total running time of our algorithm is

$$\text{poly}(n) \cdot \sum_{k'=1}^{k} c^{k'}(\ell+k)^{k-k'} = \text{poly}(n) \cdot \sum_{k'=1}^{k} \max\{c, \ell+k\}^k = \text{poly}(n) \cdot \max\{c, \ell+k\}^k. \quad \blacktriangleleft$$

Given Theorem 4.7, a natural question is whether the problem allows for a polynomial kernel.

5 Feedback Vertex Set

In this section, we study secluded versions of the Feedback Vertex Set (FVS) problem, which asks, given a graph G and an integer k, whether there is a set $W \subseteq V(G)$, $|W| \leq k$, such that $G - W$ is cycle-free.

5.1 Secluded Feedback Vertex Set

We show in this subsection that the problem below is NP-hard and admits a polynomial kernel.

Secluded Feedback Vertex Set (SFVS)
Input: A graph $G = (V, E)$ and an integer k.
Question: Is there a set $S \subseteq V$ such that $G - S$ is cycle-free and $|N_G[S]| \leq k$?

▶ **Theorem 5.1** ($\star$). Secluded Feedback Vertex Set *is NP-hard.*

The proof is by a reduction from the FVS problem and works by attaching to each vertex in the original graph a large set of new degree-one neighbors. On the positive side, SFVS remains fixed-parameter tractable with respect to k:

▶ **Theorem 5.2.** Secluded Feedback Vertex Set *admits a kernel with $O(k^5)$ vertices.*

In the remainder of this section, we describe the data reduction rules that yield the polynomial-size problem kernel. The running time and correctness proofs, as well as the kernel-size bound of Theorem 5.2 is deferred to a full version of this paper. The reduction rules are inspired by the kernelization algorithm for the Tree Deletion Set problem given by Giannopoulou et al. [13].

We start by introducing the following notation. A *2-core* [22] of a graph G is a maximum subgraph H of G such that, for each $v \in V(H)$, we have $\deg_H(v) \geq 2$. Note that a 2-core H of a given graph G is unique and can be found in polynomial time [22]. If H is a 2-core of G, then we use $\deg_{H|0}(v)$ to denote $\deg_H(v)$ if $v \in V(H)$ and $\deg_{H|0}(v) = 0$ if $v \notin V(H)$.

▶ **Observation 5.3.** *Let G be a graph, H its 2-core, and C a connected component of $G - V(H)$. Then $|N(C) \cap V(H)| \leq 1$ and $|N(H) \cap V(C)| \leq 1$.*

Proof. We only show the first statement. The second statement follows analogously. Towards a contradiction, assume that $|N(C) \cap V(H)| \geq 2$. Then, there are vertices $x, y \in V(H)$ with $x \neq y$ such that x and y have neighbors $a, b \in V(C)$. If $a = b$, then $G' = G[V(H) \cup \{a\}]$ is a subgraph of G such that $\deg_{G'}(v) \geq 2$ for every $v \in V(G')$, contradicting the choice of H as the 2-core of G. If $a \neq b$, then, since C is connected, there is a path P_C in C connecting a and b. Thus, $G' = G[V(H) \cup V(P_C)]$ is a subgraph of G such that $\deg_{G'}(v) \geq 2$ for every $v \in V(G')$, again contradicting the choice of H as the 2-core of G. $\blacktriangleleft$

Note that only the vertices in the 2-core are involved in cycles of G. However, the vertices outside the 2-core can influence the size of the closed neighborhood of the feedback vertex set. Next, we apply the following reduction rules to our input instance with G given its 2-core H.

▶ **Reduction Rule 1.** *If* $\deg_{H|0}(v) = 0$ *for every* $v \in N[u]$*, then delete* u*.*

Note that, if Reduction Rule 1 has been exhaustively applied, then $\deg_{H|0}(v) = 0$ implies that v has exactly one neighbor, which is in the 2-core of the graph.

▶ **Reduction Rule 2.** *If* $v_0, v_1, \ldots, v_\ell, v_{\ell+1}$ *is a path in the input graph such that* $\ell \geq 3$*,* $\deg_{H|0}(v_i) = 2$ *for every* $i \in \{1, \ldots, \ell\}$*,* $\deg_{H|0}(v_0) \geq 2$*, and* $\deg_{H|0}(v_{\ell+1}) \geq 2$*, then let* $r = \min\{\deg_G(v_i) \mid i \in \{1, \ldots, \ell\}\} - 2$ *and remove vertices* $v_1, \ldots, v_\ell$ *and their neighbors not in the 2-core. Then introduce two new vertices* u_1 *and* u_2 *with edges* $\{v_0, u_1\}$*,* $\{u_1, u_2\}$*, and* $\{u_2, v_{l+1}\}$ *and* $2r$ *further new vertices and connect* u_1 *with* r *of them and* u_2 *with another* r *of them.*

For $x \in V(G)$, we denote by $\text{petal}(x)$ the maximum number of cycles only intersecting in x.

▶ **Reduction Rule 3.** *If there is a vertex* $x \in V(G)$ *such that* $\text{petal}(x) \geq \lceil \frac{k}{2} \rceil$*, then output that* (G, k) *is a no-instance of SFVS.*

▶ **Reduction Rule 4.** *If* $v \in V(G)$ *is a vertex such that* $\deg_G(v) > k$*, but* $\deg_{H|0}(v) < \deg_G(v)$*, then remove one of its neighbors not in the 2-core.*

▶ **Reduction Rule 5.** *Let* x, y *be two vertices of* G*. If there are at least* k *internally vertex disjoint paths of length at least 2 between* x *and* y *in* G*, then output that* (G, k) *is a no-instance of SFVS.*

Note that Reduction Rules 1, 2, 4, and 5 can be applied trivially in polynomial time. Reduction Rule 3 can be applied exhaustively in polynomial time due to Thomassé [23].

After applying the reduction rules above exhaustively, one can show that the resulting instance is either a no-instance, or the number of vertices is $O(k^5)$.

5.2 Small Secluded Feedback Vertex Set

In contrast to restricting the closed neighborhood of a feedback vertex set, restricting the open neighborhood by a parameter yields a W[1]-hard problem.

Small Secluded Feedback Vertex Set
Input: A graph $G = (V, E)$ and two integers k, ℓ.
Question: Is there a set $S \subseteq V$ such that $G - S$ is cycle-free, $|S| \leq k$, and $|N_G(S)| \leq \ell$?

▶ **Theorem 5.4.** Small Secluded Feedback Vertex Set *is W[1]-hard with respect to* ℓ*.*

Proof. We provide a parameterized reduction from Multicolored Independent Set (MIS): given a k-partite graph $G = (V, E)$ and its partite sets $V_1 \cup \ldots \cup V_k = V$, the question is whether there is an independent set I of size k such that $I \cap V_i \neq \emptyset$ for each $i \in \{1, \ldots, k\}$. MIS is W[1]-hard when parameterized by the size k of the independent set [10].

Let $G = (V, E)$ with partite sets $V_1 \cup V_2 \cup \ldots \cup V_k = V$ be an instance of MIS. We can assume that for each $i \in \{1, \ldots, k\}$ we have $|V_i| \geq 2$ and there is no edge $\{v, w\} \in E$ with $v, w \in V_i$. We create an instance (G', k', ℓ) of Small Secluded Feedback Vertex Set (SSFVS) with $k' = |V| - k$ and $\ell = k + 1$ as follows.

Construction: Refer to Figure 1 for an sketch of the following construction. Initially, let $G' := G$. For each $i \in \{1, \ldots, k\}$ turn V_i into a clique, that is, add the edge sets

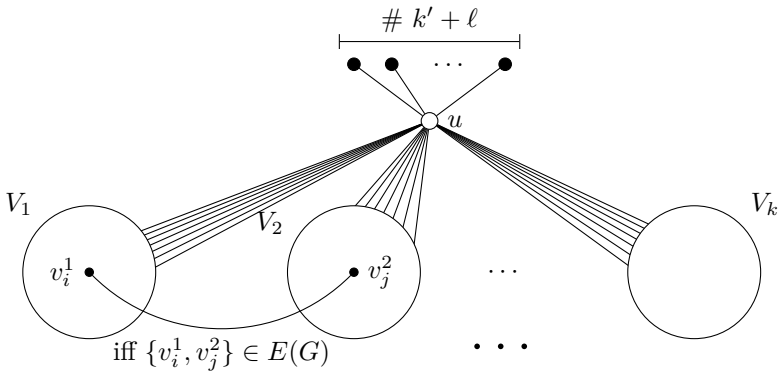


Figure 1 Sketch of the construction in proof of Theorem 5.4. The circles refer to cliques with vertex set V_i, $i \in \{1, \ldots, k\}$.

$\{\{a, b\} \mid a, b \in V_i, a \neq b\}$. Next, add to G' a vertex u and a set L of $k' + \ell$ vertices. Finally, connect each vertex in $V \cup L$ to u by an edge.

Correctness: We show that (G, k) is a yes-instance of MIS if and only if (G', k', ℓ) is a yes-instance of SSFVS.

"⇒": Let (G, k) be a yes-instance of MIS and let $I \subseteq V$ with $|I| = k$ be a multicolored independent set in G. We delete all vertices in $S := V(G') \setminus (I \cup L \cup \{u\})$ from G'. Observe that $|S| = |V| - k = k'$. Moreover, $N_{G'}(S) = k + 1 = \ell$. Since there is no edge between any two vertices in I, $G - S$ forms a star with center u and $k' + \ell + 1 + k$ vertices. Since every star is acyclic, (G', k', ℓ) is a yes-instance of SSFVS.

"⇐": Let (G', k', ℓ) be a yes-instance of SSFVS and let $S \subseteq V(G')$ be a solution. Observe that $G'[V_i \cup \{u\}]$ forms a clique of size $|V_i| + 1$ for each $i \in \{1, \ldots, k\}$. Since the budget does not allow for deleting the vertex u (i.e. $u \notin S$), all but one vertex in each V_i must be deleted. Since $k' = |V| - k$ and $|V_i| \geq 2$ for all $i \in \{1, \ldots, k\}$, S contains exactly $|V_i| - 1$ vertices of V_i for each $i \in \{1, \ldots, k\}$. Hence, $|S| = |V| - k$ and $N_{G'}(S) = k + 1 = \ell$. Let $F := V \setminus S$ denote the set of vertices in V not contained in S. Recall that $|F| = k$ and $|F \cap V_i| = 1$ for all $i \in \{1, \ldots, k\}$. Next, suppose there is an edge between two vertices $v, w \in F$. Since $u \notin S$ and u is incident to all vertices in V, the vertices u, v, w form a triangle in G'. This contradicts the fact that S is a solution for (G', k', ℓ), that is, that $G' - S$ is acyclic. It follows that $E(G'[F]) = \emptyset$, that is, no two vertices in F are connected by an edge. Together with $|F| = k$ and $|F \cap V_i| = 1$ for all $i \in \{1, \ldots, k\}$, it follows that F forms a multicolored independent set in G. Thus, (G, k) is a yes-instance of MIS. ◀

6 Independent Set

For Independent Set, it makes little sense to bound the size of the closed neighborhood from above, as in this case the empty set always constitutes a solution. One might ask for an independent set with closed neighborhood as large as possible. However, for any inclusion-wise maximal independent set S, one has $N[S] = V$. Hence, this question is also trivial. Therefore, in this section we only consider the following problem.

Large Secluded Independent Set (LSIS)
Input: A graph $G = (V, E)$ and two integers k, ℓ.
Question: Is there an independent set $S \subseteq V$ such that $|S| \geq k$ and $|N_G(S)| \leq \ell$?

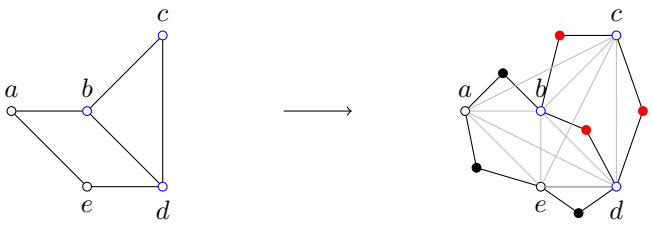


Figure 2 Example of the construction in Theorem 6.1. The left-hand side shows the original graph, the right-hand side the graph constructed by the reduction, where the newly introduced edges between each pair of vertices of the original graph are drawn in grey. The vertices introduced for each edge of the original graph are filled red and black, the corresponding new edges are drawn in black. Note that the blue vertices of the original graph form a clique and that the vertices corresponding to the edges of said clique (filled red) form an independent set in the new graph.

The case $\ell = |V|$ equals Independent Set and, thus, LSIS is $W[1]$-hard with respect to k. We show that LSIS is also W[1]-hard when parameterized by $k + \ell$.

▶ **Theorem 6.1.** Large Secluded Independent Set *is* $W[1]$*-hard with respect to* $k + \ell$.

Proof. We provide a polynomial-parameter transformation from Clique parameterized by the solution size k.

Construction. Let (G, k) be an instance of Clique and assume without loss of generality that $k < |V(G)| - 1$ (otherwise, solve the instance in polynomial time). We construct an equivalent instance (G', k', ℓ') of Large Secluded Independent Set as follows (see Figure 2 for an example).

Initially, let G' be an empty graph. Add all vertices of G to G'. Denote the vertex set by V. If two vertices of G are adjacent, we add a vertex to G', that is, G' additionally to V contains the vertex set $X := \{x_{uv} \mid \{u, v\} \in E\}$. Next, connect x_{uv} to u and v, that is, add the edge set $E' = \{\{u, x_{uv}\}, \{v, x_{uv}\} \mid \{u, v\} \in E\}$. Finally, connect any two vertices in V by an edge. Graph G' consists of the vertex set $V \cup X$ and of the edge set $E' \cup \binom{V}{2}$. Observe that X forms an independent set in G'. Set $k' := \binom{k}{2}$ and $\ell' := k$. We claim that (G, k) is a yes-instance of Clique if and only if (G', k', ℓ') is a yes-instance of Large Secluded Independent Set.

"⇒": Let $C \subseteq V(G)$ be a clique of size $k = |C|$ in G. We claim that $X' := \{x_{u,v} \mid u, v \in C\}$ forms an independent set of size $\binom{k}{2}$ with $|N(X')| = k = \ell'$ in G'. Since $X' \subseteq X$, X' forms an independent set. Moreover, since C is a clique of size k, there are $\binom{k}{2}$ edges in $G[C]$, and thus $|X'| = \binom{k}{2}$. By construction, each vertex in X is only adjacent to vertices in C. Hence, $|N(X')| = |C| = k$. Therefore, X' witnesses that (G', k', ℓ') is a yes-instance of Large Secluded Independent Set.

"⇐": Let $U \subseteq V(G')$ form an independent set of size k' with open neighborhood of size upper-bounded by ℓ'. Suppose that $v \in V$ is contained in U (observe that U contains at most one vertex of V, as otherwise it would not be independent). Then $|N(U)| \geq |V| - 1 > k = \ell'$, which contradicts the choice of U. It follows that $U \cap V = \emptyset$, and hence $U \subseteq X$. By construction, for each $x_{uv} \in U$, the vertices u, v are contained in $N(U)$. Since each vertex in U corresponds to an edge in G, we have $\binom{k}{2}$ edges incident with at most k vertices. The only graph that fulfills this property is the complete graph on k vertices. Hence, G contains a clique of size k, and thus (G, k) is a yes-instance of Clique(k). ◀

We remark that the proof above is similar to the W[1]-hardness proof for Cutting ℓ Vertices [21].

7 Summary and Future Work

In this paper, we studied the problem of finding sets of vertices in a graph that fulfill certain properties and have a small neighborhood. We presented computational complexity results for secluded and small secluded variants of s-t-Separator, q-Dominating Set, Feedback Vertex Set, $\mathcal{F}$-free Vertex Deletion, and for the large secluded variant of Independent Set. In the case of s-t-Separator, we leave as an open question the parameterized complexity of Small Secluded s-t-Separator with respect to $k + \ell$. We conjecture that it is fixed-parameter tractable. Concerning Secluded $\mathcal{F}$-free Vertex Deletion, we would like to point out that it is an interesting question which families $\mathcal{F}$ exactly yield NP-hardness as opposed to polynomial-time solvability.

A natural way to generalize our results would be to consider vertex-weighted graphs and directed graphs. This generalization was already investigated by Chechik et al. [5] for Secluded Path and Secluded Steiner Tree. Furthermore, we would like to mention that replacing the bound on the open neighborhood in the case of small secludedness by a bound on the outgoing edges of a solution would be an interesting modification of the problem. The variation follows the idea of the concept of *isolation* as used, e.g., in [14, 16, 17, 18]. As the number of outgoing edges is at least as large as the open neighborhood, this might offer new possibilities for fixed-parameter algorithms.

Acknowledgment. We would like to thank the anonymous referees of IPEC for comments that helped to improve the paper and for pointing us to the work of Fomin et al. [12]. The second author thanks Nikolay Karpov (St. Petersburg Department of the Steklov Institute of Mathematics of the RAS) for discussion on secluded problems.

References

1 R. van Bevern. *Fixed-Parameter Linear-Time Algorithms for NP-hard Graph and Hypergraph Problems Arising in Industrial Applications*. PhD thesis, TU Berlin, 2014.

2 R. van Bevern. Towards optimal and expressive kernelization for d-Hitting Set. *Algorithmica*, 70(1):129–147, 2014.

3 R. van Bevern, H. Moser, and R. Niedermeier. Approximation and tidying – a problem kernel for s-Plex Cluster Vertex Deletion. *Algorithmica*, 62(3-4):930–950, 2012.

4 L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inform. Process. Lett.*, 58(4):171–176, 1996.

5 S. Chechik, M. Johnson, M. Parter, and D. Peleg. Secluded connectivity problems. In *Proc. 23rd ESA*, volume 8125 of *LNCS*, pages 301–312. Springer, 2013.

6 M. Dom, D. Lokshtanov, and S. Saurabh. Kernelization lower bounds through colors and IDs. *ACM TALG*, 11(2):13, 2014.

7 R. Downey, V. Estivill-Castro, M. Fellows, E. Prieto, and F. 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.

8 S. Fafianie and S. Kratsch. A shortcut to (sun)flowers: Kernels in logarithmic space or linear time. In *Proc. 40th MFCS*, volume 9235 of *LNCS*, pages 299–310. Springer, 2015.

9 M. Fellows, J. Guo, H. Moser, and R. Niedermeier. A complexity dichotomy for finding disjoint solutions of vertex deletion problems. *ACM ToCT*, 2(2):5, 2011.

10 M. Fellows, D. Hermelin, F. Rosamond, and S. Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009.

11 F. Fomin, P. Golovach, N. Karpov, and A. Kulikov. Parameterized complexity of secluded connectivity problems. In *Proc. 35th FSTTCS*, volume 45 of *LIPIcs*, pages 408–419. Schloss Dagstuhl, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.408.

12 F. Fomin, P. Golovach, and J. Korhonen. On the parameterized complexity of cutting a few vertices from a graph. In *Proc. 38th MFCS*, volume 8087 of *LNCS*, pages 421–432. Springer, 2013. doi:10.1007/978-3-642-40313-2_38.

13 A. Giannopoulou, D. Lokshtanov, S. Saurabh, and O. Suchý. Tree deletion set has a polynomial kernel (but no $\mathrm{OPT}^{\mathrm{O}(1)}$ approximation). In *Proc. 34th FSTTCS*, volume 29 of *LIPIcs*, pages 85–96. Schloss Dagstuhl, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.85.

14 F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Isolation concepts for clique enumeration: Comparison and computational experiments. *Theor. Comput. Sci.*, 410(52):5384–5397, 2009.

15 F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theor. Comput. Syst.*, 47(1):196–217, 2010.

16 F. Hüffner, C. Komusiewicz, and M. Sorge. Finding highly connected subgraphs. In *Proc. 41st SOFSEM*, volume 8939 of *LNCS*, pages 254–265. Springer, 2015.

17 H. Ito, K. Iwama, and T. Osumi. Linear-time enumeration of isolated cliques. In *Proc. 13th ESA*, volume 3669 of *LNCS*, pages 119–130. Springer, 2005.

18 C. Komusiewicz, F. Hüffner, H. Moser, and R. Niedermeier. Isolation concepts for efficiently enumerating dense subgraphs. *Theor. Comput. Sci.*, 410(38-40):3640–3654, 2009.

19 J. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980.

20 D. Lokshtanov, N. Misra, G. Philip, M. Ramanujan, and S. Saurabh. Hardness of r-dominating set on graphs of diameter $(r+1)$. In *Proc. 8th IPEC*, volume 8246 of *LNCS*, pages 255–267. Springer, 2013.

21 D. Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006. doi:10.1016/j.tcs.2005.10.007.

22 S. Seidman. Network structure and minimum degree. *Soc. Networks*, 5(3):269–287, 1983.

23 S. Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM TALG*, 6(2), 2010. doi:10.1145/1721837.1721848.

The Parameterized Complexity of Dependency Detection in Relational Databases

Thomas Bläsius[1], Tobias Friedrich[2], and Martin Schirneck[3]

1 **Hasso Plattner Institute, Potsdam, Germany**
`thomas.blaesius@hpi.de`
2 **Hasso Plattner Institute, Potsdam, Germany**
`tobias.friedrich@hpi.de`
3 **Hasso Plattner Institute, Potsdam, Germany**
`martin.schirneck@hpi.de`

Abstract

We study the parameterized complexity of classical problems that arise in the profiling of relational data. Namely, we characterize the complexity of detecting *unique column combinations* (candidate keys), *functional dependencies*, and *inclusion dependencies* with the solution size as parameter. While the discovery of uniques and functional dependencies, respectively, turns out to be $W[2]$-complete, the detection of inclusion dependencies is one of the first natural problems proven to be complete for the class $W[3]$. As a side effect, our reductions give insights into the complexity of enumerating all minimal unique column combinations or functional dependencies.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases parameterized complexity, unique column combination, functional dependency, inclusion dependency, profiling relational data

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.6

1 Introduction

Data profiling is the process of gathering metadata from a given database, which in turn facilitates various tasks such as data cleansing, normalization and integration as well as query optimization. A common problem in data profiling is the detection of different types of dependencies between pieces of data, most notably unique column combinations, functional dependencies, and inclusion dependencies. Due to their practical relevance, these three problems have received much attention, which lead to numerous detection as well as enumeration algorithms, see e.g. the survey by Abedjan, Golab and Naumann [1]. Despite the fact that these algorithms perform well in practice, there are usually no theoretical performance guarantees. This is not very surprising as all three problems are known to be intractable: finding a minimum unique column combination is NP-complete [3] and cannot be approximated within a factor of $1/4 \log n$ (under reasonable complexity assumptions) [2], finding a minimum functional dependency is also NP-complete [7] and finding a maximum inclusion dependency is NP-complete even for restricted cases [14].

One approach to overcome these difficulties is to exploit properties that are usually observed in realistic data to design algorithms that guarantee a polynomial run time in case these features are present in the problem instance. Consider for example the histograms in Figure 1, showing the size distribution of minimal unique column combinations, minimal functional dependencies, and maximal inclusion dependencies in the MusicBrainz database [18]. Usually the majority of functional dependencies (as well as unique column

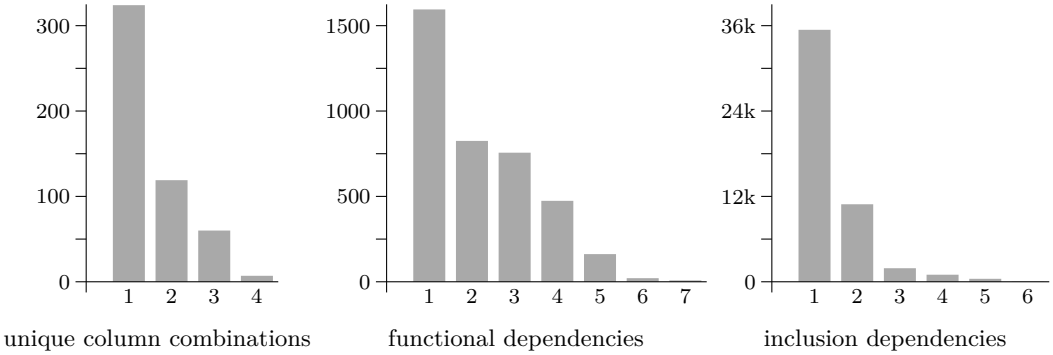


Figure 1 The number of minimal unique column combinations, minimal functional dependencies and maximal inclusion dependencies for given solution sizes in the MusicBrainz database.

combinations and inclusion dependencies) are rather small. Beside surrogate keys, giving rise to multiple functional dependencies of size 1, natural causalities also lead to small functional dependencies. For example, the name of an event together with the year in which it starts determines the year in which it ends, implying a functional dependency of size 2. Note that the starting year alone is not enough to infer this information. The name of the action, however, seems to indicate whether the event ends in the starting year or the following one.

Although the size k of the minimum functional dependency can in principle be (almost) as large as the total number of attributes, it appears to be a reasonable assumption that k is significantly smaller. It is thus very natural to ask whether the problem of finding a minimum functional dependency is *fixed-parameter tractable* (FPT) with respect to k, i.e., whether it can be solved in time $O(f(k) \cdot p(n))$, where p is a polynomial in the input size n, while f is an arbitrary function in the parameter k, but not in n. Note that the running time of an FPT-algorithm in general can still be superpolynomial. However, when assuming the parameter k to be bounded by a constant, one obtains a polynomial running time, in $O(p(n))$, whose order of growth does not depend on k. Hence, one can think of parameterized complexity as being a more fine-grained approach to complexity theory.

Parameterized complexity has been a great success in the design and analysis of algorithms [11, 5]. Nevertheless, its techniques have rarely been employed in the context of database theory so far. A notable exception is the complexity of database queries. Papadimitriou and Yannakakis [17] considered this problem for different query languages using the size of the query or alternatively the number of variables as the parameter. They showed that presumably none of the variants admits an FPT-algorithm, as the resulting problems are at least $W[1]$-hard (some are actually $W[t]$-hard for any positive integer t, $W[\mathrm{SAT}]$-hard or even $W[\mathrm{P}]$-hard). For further results on the parameterized complexity of database queries see the survey by Grohe [13]. Besides queries, we are not aware of any algorithmic database problems that have been considered through the lens of parameterized complexity.

Our Contribution and Outline. We show that detecting minimum unique column combinations and minimum functional dependencies are both $W[2]$-complete problems. Also, we prove that finding maximum inclusion dependencies is $W[3]$-complete. Thereby we completely settle the parameterized complexity of these problems with the solution size as parameter. We would like to point out that the completeness for the class $W[3]$ of a well-studied problem like the discovery of inclusion dependencies is quite surprising as natural problems are rarely $W[3]$-complete. In fact, besides a result by Chen and Zhang [4] related to supply chain management, we are not aware of *any* natural $W[t]$-complete problem for $t > 2$.

In Section 2 we give basic definitions and formal problem statements. In Section 3, we examine the detection of minimum unique column combinations as well as minimum functional dependencies. For the latter, we actually consider two variants, one for which the right hand side of the functional dependency is fixed and one in which it is variable. We show that all three problems are $W[2]$-complete. As a byproduct, our reductions (involving the problem Hitting Set) have certain implications on the computational hardness of enumerating all unique column combinations or functional dependencies of a given relation. See the end of Section 3 for more details. In Section 4 we show that finding minimum inclusion dependencies for a pair of relations is $W[3]$-complete. We also show that the problem remains $W[3]$-complete if both relations are defined over the same schema together with a fixed mapping between the columns of the tables. In Section 5, we conclude this paper by discussing alternative parameter choices as well as possible future research in general.

2 Notation and Problems

2.1 Parameterized Complexity

For an instance I of a decision problem and a parameter $k \in \mathbb{N}^+$, the pair (I, k) is an instance of the corresponding *parameterized problem*. The running time of an algorithm is then considered not only in terms of the input size $|I|$ but also in terms of k. A parameterized problem is *fixed-parameter tractable*, i.e., it belongs to the complexity class FPT, if a given instance can be solved in time $O(f(k) \cdot p(|I|))$, where p is a polynomial while f is an arbitrary computable function. We then also say that the algorithm runs in FPT-*time*.

Let P and P' be two parameterized problems. A *parameterized reduction* from P to P' is an algorithm running in FPT-time that maps an instance (I, k) of P to an equivalent instance (I', k') of P' such that the parameter k' depends only on the value of k (and not on $|I|$). Note that an (hypothetical) FPT-algorithm for P' would also yield an FPT-algorithm for P via this reduction. Hence, considering their parameterized complexity, P is at most as hard as P', which we denote by $P \leq_{\text{FPT}} P'$. If conversely $P' \leq_{\text{FPT}} P$ also holds, we say that P and P' are FPT-*equivalent*.

The parameterized reduction leads to a hierarchy of complexity classes, the so-called W-*hierarchy*, by specifying a *complete* problem for each class. To define the desired family of problems, we employ Boolean formulas in propositional logic. Let φ be such a formula. A satisfying truth assignment for φ has *Hamming weight* k if exactly k variables are set to TRUE in this assignment; we also call the set of these k variables a *solution* for φ. The formula φ is t-*normalized* if it can be written as a conjunction of disjunctions of conjunctions of disjunctions (and so on) of literals with $t-1$ alternations between conjunction and disjunction. Observe that a Boolean formula is 2-normalized if it is in conjunctive normal form (CNF) and 3-normalized if it is a conjunction of subformulas in disjunctive normal form (DNF).

The problem Weighted t-normalized Satisfiability is to decide for a given t-normalized formula φ and a positive integer k whether φ has a weight k satisfying assignment; here k serves as the parameter. For any $t \geq 1$, a parameterized problem P is said to be in the complexity class $W[t]$ in case $P \leq_{\text{FPT}}$ Weighted t-normalized Satisfiability.[1]

The classes $\text{FPT} \subseteq W[1] \subseteq W[2] \subseteq \dots$ form an ascending hierarchy and all inclusion are assumed to be proper, which is however still unproven [11]. The higher a problem ranks in the W-hierarchy the lower we consider the chances of finding an FPT-algorithm to solve it.

[1] We tacitly avoid the classical definition of $W[t]$ via weft-t-depth-d-families of decision circuits. This is justified by the Normalization Theorem by [9, 10].

2.2 Dependencies in Relational Databases

If not explicitly stated otherwise, notation regarding relational databases follows the survey by Abedjan et al. [1]. We let R and S be *relational schemata*, i.e., sets of *columns*; each column is associated with a set of admissible values. Symbols X, Y refer to sets of columns and symbols A, B refer to a single column, an *attribute*. We denote with r_i, r_j tuples whose entries, respectively, are indexed by some schema R and, for any subset $X \subseteq R$ of columns, we let $r_i[X]$ denote the sub-tuple of r_i consisting only of the entries indexed by X. In particular, $r_i[A]$ denotes the *value* of attribute A in r_i. A set r of such tuples is an *instance* of schema R if, for any $r_i \in r$ and $A \in R$, value $r_i[A]$ is admissible for attribute A. Instances of schematas are called *relations* or *relational databases* (over the corresponding schema). With $r[X]$ we denote the collection of all sub-tuples $r_i[X]$ for $r_i \in r$.

Let r be an instance of schema R. A collection $X \subseteq R$ of columns is called a *unique column combination* or *unique* if, for any two distinct tuples $r_i \neq r_j$ in r, we have $r_i[X] \neq r_j[X]$. So the combination of values for X fully identifies a tuple of relation r. Otherwise, X is called a *non-unique*. The *size* of a unique X is the cardinality $|X|$. Clearly, any superset of a unique is unique and any subset of a non-unique is again non-unique. The problem Unique is to decide for a given relational database r and a positive integer k whether r has a unique column combination of size at most k. Unique is known to be NP-complete [3].

A *functional dependency* (FD) over a schema R is an expression of the form $X \to A$ for some set $X \subseteq R$ of columns and an attribute $A \in R$. The set X is called the *left-hand side* (LHS) of the dependency and attribute A the *right-hand side* (RHS). A functional dependency $X \to A$ *holds* in an instance r (of schema R) if any pair of tuples that agree on X also agree on A, i.e., if $r_i[X] = r_j[X]$ implies $r_i[A] = r_j[A]$ for any two tuples $r_i, r_j \in r$. Otherwise, the FD is said to *fail* in r. A functional dependency is *non-trivial* if $A \notin X$ ($X \to A$ evidently holds if $A \in X$). The *size* of an FD is the cardinality of its LHS. The problem FD is to decide for a given relational database r and a positive integer k whether there is a non-trivial functional dependency of size at most k that holds in r. The problem $\text{FD}_{\textit{fixed}}$ is to decide for a given attribute $A \in R$, whether there is such a functional dependency with right-hand side A. The restricted variant $\text{FD}_{\textit{fixed}}$ is known to be NP-complete [7].

At last we define inclusions between columns among different relations. Let r be an instance of schema R and s be an instance of S. For some $X \subseteq R$, let $\sigma\colon X \to S$ be an injective map. Then the pair (X, σ) is an *inclusion dependency* (IND) if, for each tuple $r_i \in r$, there exists a tuple $s_j \in s$ such that $r_i[A] = s_j[\sigma(A)]$ for every $A \in X$, i.e., $r[X] \subseteq s[\sigma(X)]$. If the map σ is given in the input, we simply say that X is the inclusion dependency. The *size* of an inclusion dependency is $|X|$. The problem IND is to decide for two relations r and s (over schemata R and S, respectively) and a positive integer k whether there is an inclusion dependency (X, σ) of size at least k. In case of $R = S$ and σ being the identity mapping over R, the problem $\text{IND}_{\textit{fixed}}$ is to decide whether there is an inclusion dependency X of size at least k. Detecting an inclusion dependency in a relational database is NP-complete [14].

Note that the decision problems defined above do not depend upon asking for a solution of size at most/at least k as opposed to one of size exactly k. A functional dependency stays valid when adding arbitrary additional columns to the LHS. The same holds for unique column combinations. Conversely, if a pair of relations admits an inclusion dependency of size at least k, one can obtain one comprising exactly k columns by removing dispensable attributes. In this paper, we consider all the decision problems to be parameterized by the size of the respective solution. Thus, Unique, FD, $\text{FD}_{\textit{fixed}}$, IND and $\text{IND}_{\textit{fixed}}$ refer to the corresponding parameterized problems with parameter k.

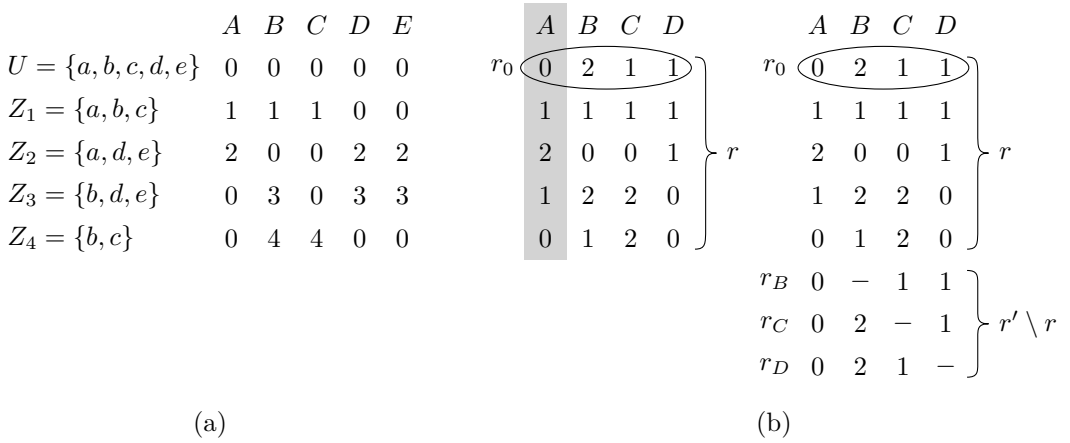


Figure 2 (a) An instance of Hitting Set and its equivalent instance of Unique. (b) An instance r of problem FD$_{fixed}$ with fixed RHS A and the resulting instance r' of problem FD. Note that the functional dependency $AB \to D$ holds in r but not in r'.

3 Unique Column Combinations and Functional Dependencies

It is a well-known phenomenon that throughout the fields of database design and data profiling theoreticians as well as practitioners are frequently confronted with the task of finding an inclusion-minimal collection of items that has a non-empty intersection with each member of a prescribed family of sets [1, 6, 16]. Thus, they aim to solve instances of the so-called Hitting Set problem. In this section we show that this encounter is somewhat inevitable in the sense that detecting uniques or functional dependencies is both *exactly* as hard as finding a hitting set in terms of parameterized complexity.

Hitting Set is formally defined as follows. For a finite system of subsets $\mathcal{Z} \subseteq \mathcal{P}(U)$ of some finite ground set U, a set $H \subseteq U$ is called *hitting set* iff for all $Z \in \mathcal{Z}$, $H \cap Z \neq \emptyset$. The problem Hitting Set is to decide for a positive integer k whether there is a hitting set H with $|H| \leq k$. Hitting Set is NP-complete [15] and $W[2]$-complete with respect to k [11]. Hence, we can utilize it to show the $W[2]$-completeness of the dependency problems at hand.

More precisely, in this section we establish a (seemingly ascending) chain of problems via parameterized reductions. This chain consists of problems Hitting Set, Unique, FD$_{fixed}$, FD and Weighted 2-normalized Satisfiability in that order. As the first and last problem are both $W[2]$-complete and thus FPT-equivalent, this in fact proves equivalence (and hence $W[2]$-completeness) for the other problems as well. Due to space constraints, we only sketch key ideas for the first three reductions (Hitting Set to Unique, Unique to FD$_{fixed}$ and FD$_{fixed}$ to FD).

▶ **Lemma 1.** Hitting Set $\leq_{\text{FPT}}$ Unique $\leq_{\text{FPT}}$ FD$_{fixed}$ $\leq_{\text{FPT}}$ FD.

Proof (sketch). The first reduction regarding Hitting Set and Unique is a straightforward translation of the sets to hit into tuples of a relational database, cf. Figure 2.(a). For the second reduction, the main idea is to add an extra column serving as a tuple ID and to subsequently show that a column combination is unique just in case it is the LHS of a functional dependency pointing to this ID. The last reduction is established by adding to a given relation copies of an already present tuple, "ruling out" a different entry in each copy, see Figure 2.(b). This then invalidates functional dependency with unwanted RHS. ◀

In the next lemma we prove that every instance of FD can be expressed by an equivalent

Boolean CNF-formula, establishing the reduction from FD to Weighted 2-normalized Satisfiability. This is the main result of this section.

▶ **Lemma 2.** FD $\leq_{\text{FPT}}$ Weighted 2-normalized Satisfiability.

Proof. Given a relation r (over some schema R), we derive a propositional formula that has a satisfying truth assignment of weight $k+1$ if and only if there is a non-trivial functional dependency of size k that holds in r. The formula will be in CNF and hence 2-normalized. We use two types of variables distinguished by their semantic purpose, namely, the elements of $\text{Var}_R = \{x_A \mid A \in R\}$ and $\text{Var}'_R = \{x'_A \mid A \in R\}$. A variable from Var_R being set to TRUE denotes that the corresponding attribute appears on the LHS of the FD; for Var'_R, this denotes that the attribute is the RHS. Consequently, we want to ensure that any satisfying assignment chooses exactly one variable from Var'_R while the corresponding variable in Var_R is not chosen. We achieve this as follows. First, define some clause

$$c_R = \bigvee_{x'_A \in \text{Var}'_R} x'_A.$$

Then, for any two distinct attributes $A \neq B$, the clause

$$c_{A,B} = \neg x'_A \vee \neg x'_B.$$

Finally, we set, for every $A \in R$,

$$c_A = \neg x'_A \vee \neg x_A.$$

We note that the clauses $c_{A,B}$ and c_A are only added to smooth the analysis, the correctness of the reduction does not depend upon their presence. Clause c_R however is essential. Now, for any possible RHS $A \in R$ and any two tuples $r_i, r_j \in r$ with $r_i[A] \neq r_j[A]$, let

$$c_{A,r_i,r_j} = \neg x'_A \vee \bigvee_{\substack{B \in R \setminus A \\ r_i[B] \neq r_j[B]}} x_B.$$

That is, clause c_{A,r_i,r_j} contains the negative literal of the variable from Var'_R corresponding to the RHS and the positive literal of any variable in Var_R that corresponds to *another* attribute on which r_i and r_j disagree. Intuitively speaking, clause c_{A,r_i,r_j} states that if A is the RHS of a non-trivial FD holding in r, then the LHS has to contain at least one of the attributes $B \neq A$ such that $r_i[B] \neq r_j[B]$. We assemble the following Boolean formulas:

$$\varphi_{\text{RHS}} = c_R \wedge \bigwedge_{\substack{A,B \in R \\ A \neq B}} c_{A,B} \wedge \bigwedge_{A \in R} c_A;$$

as well as, for each possible RHS $A \in R$,

$$\varphi_A = \bigwedge_{\substack{r_i, r_j \in r \\ r_i[A] \neq r_j[A]}} c_{A,r_i,r_j}$$

and their conjunction

$$\varphi_{\text{LHS}} = \bigwedge_{A \in R} \varphi_A.$$

At last, we take $\varphi_{\text{FD}} = \varphi_{\text{LHS}} \wedge \varphi_{\text{RHS}}$ as the result of the reduction. In total, φ_{FD} has at most $|R|^2 + |R|\,|r|^2$ clauses with at most $|R|$ literals each and a representation of φ_{FD} is computable in time polynomial in the input size.

Regarding the correctness of this reduction, recall that we claimed φ_{FD} to have a weight $k+1$ satisfying assignment just in case there is a non-trivial functional dependency of size k in r. First assume we are given a satisfying assignment for subformula φ_{RHS}. This ensures that exactly one variable $x'_A \in \text{Var}'_R$ is set to TRUE, which uniquely determines an attribute A. The variables in Var_R that are set to TRUE determine a set X, i.e., $B \in X$ iff x_B is set to TRUE. Note that $A \notin X$ is enforced by clause c_A. Thus, a satisfying assignment for φ_{RHS} defines a non-trivial functional dependency $X \to A$. We show that this FD holds in r if and only if the assignment additionally fulfills subformula φ_{LHS}.

Suppose that $X \to A$ holds in r. Then all variables in Var'_R are set to FALSE, except x'_A, automatically satisfying clauses c_{B,r_i,r_j} for all attributes $B \neq A$. It remains to show that c_{A,r_i,r_j} is satisfied for every pair of tuples $r_i, r_j \in r$ with $r_i[A] \neq r_j[A]$. Since $X \to A$ holds, X includes, for every such pair, an attribute B such that $r_i[B] \neq r_j[B]$. Clause c_{A,r_i,r_j} in turn comprises the literal x_B, which is satisfied by above assignment. Conversely, assume $X \to A$ fails in r. Then there is a pair of tuples $r_i, r_j \in r$ such that $r_i[A] \neq r_j[A]$ but $r_i[X] = r_j[X]$. The clause c_{A,r_i,r_j} does not contain any variables x_B such that $B \in X$. As a result, all literals in c_{A,r_i,r_j} for variables from Var_R evaluate to FALSE. Literal $\neg x'_A$, however, is FALSE as well as A is the RHS. ◀

The reductions in Lemma 1 and 2, and the fact that HITTING SET and WEIGHTED 2-NORMALIZED SATISFIABILITY are both $W[2]$-complete yield the following theorem.

▶ **Theorem 3.** *The problems* UNIQUE, FD_{fixed} *and* FD *are* $W[2]$*-complete.*

We point out that our reductions have some further implications beyond the scope of parameterized complexity. Our reduction from FD_{fixed} to FD, is actually a polynomial-time reduction and thus proves that FD is NP-hard. As the problem is also trivially contained in NP, it is in fact NP-complete. This is not very surprising, but has only been proven for the restricted case FD_{fixed} before. More importantly, the parameterized reduction from HITTING SET to UNIQUE, also runs in polynomial time and establishes a one-to-one correspondence between inclusion-wise minimal hitting sets and unique column combinations. Thus, *enumerating* all uniques is at least as hard as enumerating all hitting sets. Using the techniques presented in this section, it is not hard to extend this observation to the task of enumerating all FDs with a fixed RHS or all FDs in general. Finding all hitting sets is also known by the name TRANSVERSAL ENUMERATION for hypergraphs and is notoriously difficult [12]. Up until now, there is no output-polynomial algorithm known for this problem. Hence, it is quite astonishing that the enumeration problems arising in data profiling can be solved in reasonable time on practical data sets [1]. Regarding the opposite direction, we would like to mention that there is a straight-forward polynomial reduction from FD_{fixed} to HITTING SET that additionally shows that enumerating FDs with fixed RHS or (and thus uniques) is also *at most* as hard as enumerating hitting sets. It is still unknown, however, whether this holds for arbitrary functional dependencies as well.

4 Inclusion Dependencies

In this section, we identify the detection of inclusion dependencies as one of the first natural problems to be complete for the parameterized class $W[3]$. More precisely, we show that both IND and IND_{fixed} are FPT-equivalent to a $W[3]$-complete parameterized version of

the satisfiability problem for certain Boolean formulas. Recall that a propositional formula is 3-normalized if it is a conjunction of disjunctions of conjunctions of literals. A formula is *antimonotone* if it only contains negative literals as, e.g., in the following expression

$$((\neg a \wedge \neg b) \vee (\neg c \wedge \neg d)) \wedge ((\neg a \wedge \neg c) \vee (\neg b \wedge \neg d)).$$

It is antimonotone, 3-normalized and admits a satisfying assignments of Hamming weight 0 and 1, but none of larger weight. The problem Weighted Antimonotone 3-normalized Satisfiability (WA3NS) is to decide for a 3-normalized antimonotone formula φ and a positive integer k whether φ has a weight k satisfying assignment. Although it seems to be a unreasonably restricted case of Weighted 3-normalized Satisfiability, WA3NS is $W[3]$-complete in its own right when parameterized by k [8, 9]. As all literals in WA3NS are negative, every subset of a solution is a solution. Thus, asking for a solution of size exactly k is again equivalent to asking for a solution of size at least k.

4.1 IND is in W[3]

In this subsection, we show that both variants of the IND problem are members of the parameterized class $W[3]$. As a first step, we reduce the special case $\text{IND}_{\textit{fixed}}$ to the (seemingly) more general problem IND.

▶ **Lemma 4.** $\text{IND}_{\textit{fixed}} \leq_{\text{FPT}} \text{IND}$.

Proof. Let r and s be two relations over the same schema R forming an instance of problem $\text{IND}_{\textit{fixed}}$. We introduce a new tuple $t^- = (-_A)_{A \in R}$, where each "$-_A$" is a unique symbol not used anywhere in the relations r or s. Note that (r, s) has an inclusion dependency of size k with the identity as the fixed mapping if and only if $(r \cup \{t^-\}, s \cup \{t^-\})$ has one. It is easy to see that $(r \cup \{t^-\}, s \cup \{t^-\})$ interpreted as an instance of IND (now without prescribed mapping) has an inclusion dependency (X, σ) if and only if σ is the identity and X is an inclusion dependency for (r, s), which implies the claim. ◀

To reduce IND to WA3NS we construct from the two relations an antimonotone formula which has a weight k satisfying assignment if and only if the relations have an inclusion dependency of the same size. For this, we use a correspondence between *pairs* of attributes of the relational schemata and Boolean variables.

▶ **Theorem 5.** $\text{IND} \leq_{\text{FPT}} \text{WA3NS}$.

Proof. Let $R = \{A_1, \ldots, A_{|R|}\}$ and $S = \{B_1, \ldots, B_{|S|}\}$ be two schemata. We introduce a Boolean variable $x_{m,n}$ for each pair of attributes $A_m \in R$ and $B_n \in S$. We let Var_P denote the set of variables corresponding to a collection $P \subseteq R \times S$ of such pairs. Consider subsets $X \subseteq R$ and $Y \subseteq S$ together with a bijection $\sigma \colon X \to Y$. From this we can construct a truth assignment by setting variable $x_{m,n}$ to TRUE iff $A_m \in X$ and $\sigma(A_m) = B_n$ (implying $B_n \in Y$). The resulting assignment has weight $|X|$ and the collection of all possible configurations (X, σ) is uniquely described by $\text{Var}_{R \times S}$ and the truth assignments obtained this way. Moreover, these assignments all satisfy the following Boolean formula.

$$\varphi_{\text{map}} = \left(\bigwedge_{m=1}^{|R|} \bigwedge_{n=1}^{|S|-1} \bigwedge_{n'>n}^{|S|} (\neg x_{m,n} \vee \neg x_{m,n'}) \right) \quad \wedge \quad \left(\bigwedge_{n=1}^{|S|} \bigwedge_{m=1}^{|R|-1} \bigwedge_{m'>m}^{|R|} (\neg x_{m,n} \vee \neg x_{m',n}) \right).$$

The first half of φ_{map} states that, for every pair of variables $x_{m,n}$ and $x_{m,n'}$ such that $n \neq n'$, at most one of them is set to TRUE; the second half is satisfied if the same holds for all pairs $x_{m,n}$

and $x_{m',n}$ such that $m \neq m'$. Conversely, given a satisfying assignment for φ_{map}, we obtain sets $X = \{A_m \mid \exists\, 1 \leq n \leq |S| \colon x_{m,n} = \text{TRUE}\}$ and $Y = \{B_n \mid \exists\, 1 \leq m \leq |R| \colon x_{m,n} = \text{TRUE}\}$ as well as a bijection $\sigma \colon X \to Y$ by setting $\sigma(A_m) = B_n$ iff $x_{m,n}$ is TRUE. So φ_{map} is *exactly* fulfilled by the assignments described above.

We now formalize the requirement that (X, σ), for some set $X \subseteq R$, actually is an inclusion dependency. First, assume that the relations r and s consist of a single tuple r_i and s_j, respectively. We say a pair (A_m, B_n) is *forbidden* for r_i and s_j if $r_i[A_m] \neq s_j[B_n]$. Let $F_{i,j}$ be the set of all forbidden pairs. Then (X, σ) is an inclusion dependency if $x_{m,n}$ is set to FALSE for all pairs $(A_m, B_n) \in F_{i,j}$. In terms of Boolean formulas, this is represented as

$$\varphi_{i,j} = \bigwedge_{x \in \text{Var}_{F_{i,j}}} \neg x.$$

It follows that (X, σ) is an inclusion dependency if and only if the corresponding variable assignment satisfies both φ_{map} and $\varphi_{i,j}$.

Now suppose s has multiple tuples (while r is still considered to have only one). (X, σ) is an inclusion dependency for (r, s) just in case it is an inclusion dependency for *at least one* instance $(r, \{s_j\})$, $1 \leq j \leq |s|$. If also r has more than one tuple, then (X, σ) is an inclusion dependency for (r, s) if it is one in *each* instance $(\{r_i\}, s)$, $1 \leq i \leq |r|$. Thus, we obtain an inclusion dependency if and only if φ_{map} and the formula

$$\varphi = \bigwedge_{i=1}^{|r|} \bigvee_{j=1}^{|s|} \varphi_{i,j}$$

are simultaneously satisfied by the assignment corresponding to (X, σ).

The formula $\varphi \wedge \varphi_{\text{map}}$ is antimonotone and 3-normalized and a representation can be computed in polynomial time. Moreover, by the above observation that any solution for the sub-formula φ_{map} that corresponds to (X, σ) has size $|X|$, the reduction preserves the parameter. ◀

▶ **Corollary 6.** $\text{IND}_{\textit{fixed}}$ *and* IND *are both in class* $W[3]$.

4.2 IND is W[3]-hard

In the remainder of this section, we argue that the existence of weight k satisfying assignments for 3-normalized antimonotone formulas can be decided by solving $\text{IND}_{\textit{fixed}}$ instances. As a result, we prove that both variants of problem IND are hard for the class $W[3]$. For this reduction we make use of indicator functions, which we will define in a moment. First, we would like to point out that in the following we interpret propositional formulas φ over n variables as functions $f_\varphi \colon \{0,1\}^n \to \{0,1\}$ in the obvious way. For an instance (r, s) of $\text{IND}_{\textit{fixed}}$, we encode any subset $X \subseteq R$ using its characteristic vector (of length $|R|$). Then we define the *indicator function* $f_{(r,s)} \colon \{0,1\}^{|R|} \to \{0,1\}$, where $f_{(r,s)}(X) = 1$ iff X is an inclusion dependency. We claim that for any antimonotone and 3-normalized formula φ, there is an instance (r, s) of $\text{IND}_{\textit{fixed}}$ computable in FPT-time such that $f_\varphi = f_{(r,s)}$. This clearly gives an FPT-reduction from WA3NS to $\text{IND}_{\textit{fixed}}$. The remaining lemmas are dedicated to proving this claim.

Recall that an antimonotone, 3-normalized formula is a conjunction of antimonotone sub-formulas in DNF. Thus, the top level connective is a conjunction. Next we show how to model this connective in terms of relational databases.

▶ **Lemma 7.** *Let $(r^{(1)}, s^{(1)})$ and $(r^{(2)}, s^{(2)})$ be two instances for the problem* $\text{IND}_{\textit{fixed}}$ *(all relations are over the same schema R) with indicator functions $f^{(1)}$ and $f^{(2)}$, respectively. Then there exists an instance (r, s) (over R), having size $|r| = |r^{(1)}| + |r^{(2)}|$ and $|s| = |s^{(1)}| + |s^{(2)}|$, with indicator function $f_{(r,s)} = f^{(1)} \wedge f^{(2)}$.*

Proof. W.l.o.g. assume that the values appearing in $r^{(1)}$ and $s^{(1)}$ are disjoint from those in $r^{(2)}$ and $s^{(2)}$. We straightforwardly construct instance (r, s) by defining $r = r^{(1)} \cup r^{(2)}$ and $s = s^{(1)} \cup s^{(2)}$. Obviously, the construction matches the requirements on both the computability and size of the resulting instance. It remains to show that $f_{(r,s)} = f^{(1)} \wedge f^{(2)}$.

Equivalently, we show that X is an inclusion dependency in (r, s) if and only if it is one in both sub-instances $(r^{(1)}, s^{(1)})$ and $(r^{(2)}, s^{(2)})$. First, suppose the condition holds. Then, for every tuple $r_i^{(1)} \in r^{(1)}$, there exists a tuple $s_j^{(1)} \in s^{(1)}$ with $r_i^{(1)}[X] = s_j^{(1)}[X]$, and the same holds for $r^{(2)}$ and $s^{(2)}$. As all said tuples are also present in (r, s), X is an inclusion dependency there as well. Conversely, suppose X is not an inclusion dependency in, say, $(r^{(1)}, s^{(1)})$. Then $r^{(1)}$ has a tuple $r_i^{(1)}$ such that $r_i^{(1)}$ disagrees on X with *every* $s_j^{(1)} \in s^{(1)}$. By construction, $r_i^{(1)}$ is also in r. Moreover, all tuples in s belong either to $s^{(1)}$ or have completely disjoint values. This results in $r_i^{(1)}[X] \neq s_j[X]$ for every $s_j \in s$ as desired. ◀

One could hope that there is a similar method treating disjunctions. However, we believe that there is none that is both computable in FPT-time and compatible with a complementing method for conjunctions (e.g. the one shown above). The reasoning is as follows: Negative literals are easily expressible as instances of $\text{IND}_{\textit{fixed}}$ using pairs of single-tuple relations. Together with FPT-time procedures of constructing conjunctions as well as disjunctions one could encode antimonotone Boolean formulas of *arbitrary logical depth*. According to the Antimonotone Collaps Theorem by [11] this would render $\text{IND}_{\textit{fixed}}$ to be hard for *all* classes $W[t]$. As a consequence of Theorem 5 (in conjunction with Lemma 4) the W-hierarchy would collapse to the level $W[3]$. That being said, there is a method specifically tailored to antimonotone formulas in DNF.

▶ **Lemma 8.** *Let φ be an antimonotone formula in DNF. Then there is an instance (r, s) for problem* $\text{IND}_{\textit{fixed}}$ *of size polynomial in $|\varphi|$ such that $f_\varphi = f_{(r,s)}$.*

Proof. Let $x_1, \ldots, x_n$ be the variables of formula φ. Define schema $R = \{A_1, \ldots, A_n\}$ by identifying each variable x_i with attribute A_i. To build the instance (r, s) over the schema R, we first describe the relation r and then construct s accordingly.

As φ is a DNF formula, it is the disjunction of conjunctive clauses $c_1, \ldots, c_m$. For each clause c_j, we create the tuple r_j by setting $r_j[A_i] = j$ if x_i occurs in c_j and $r_j[A_i] = 0$ otherwise. For example, the clause $c_1 = (\neg x_1 \wedge \neg x_2 \wedge \neg x_3)$ in Figure 3 leads to the tuple $(1, 1, 1, 0, 0, 0)$. Relation s is obtained by first creating m copies of r. In the j-th copy we then set the value for attribute A_i to the special symbol "$-$" whenever x_i occurs in the conjunctive clause c_j; see Figure 3 again. Note that $|R|$ equals the number of variables of φ and $|r|$ is linear while $|s|$ is quadratic in the number of conjunctive clauses in φ. In total, the size of the instance (r, s) is polynomial in $|\varphi|$. It is left to show that $f_\varphi = f_{(r,s)}$.

First, suppose $f_\varphi(X) = 1$ for some binary vector X of length $|R|$ or, equivalently, a subset $X \subseteq R$. We show that $f_{(r,s)}(X) = 1$. Necessarily, we have $f_{c_j}(X) = 1$ for at least one conjunctive clause c_j and since the clause contains only negative literals, all of its variables evaluate to 0. This is equivalent to X not containing any attributes corresponding to variables occurring in c_j. In the j-th copy of r in s the values were changed to "$-$" for exactly those attributes. Thus, restriction $s[X]$ comprises an exact copy of $r[X]$, resulting in $f_{(r,s)}(X) = 1$ by definition.

$\varphi = c_1 \vee c_2 \vee c_3$

$c_1 = (\neg x_1 \wedge \neg x_2 \wedge \neg x_3)$

$c_2 = (\neg x_2 \wedge \neg x_4 \wedge \neg x_5)$

$c_3 = (\neg x_1 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_6)$

A_1	A_2	A_3	A_4	A_5	A_6
1	1	1	0	0	0
0	2	0	2	2	0
3	0	3	3	0	3

A_1	A_2	A_3	A_4	A_5	A_6
–	–	–	0	0	0
–	–	–	2	2	0
–	–	–	3	0	3
1	–	1	–	–	0
0	–	0	–	–	0
3	–	3	–	–	3
–	1	–	–	0	–
–	2	–	–	2	–
–	0	–	–	0	–

Figure 3 Illustration of Lemma 8. Formula φ is on the left (with the three conjunctive clauses c_1, c_2, c_3), relation r in the center and relation s on the right.

For the opposite direction, suppose $f_\varphi(X) = 0$. Then, for each conjunctive clause, at least one variable evaluates to 1 and, consequently, for each tuple in s, the value of at least one attribute in X was replaced by "$-$". As r does not contain the special symbol "$-$" at all, X is not an inclusion dependency, i.e., $f_{(r,s)}(X) = 0$. ◀

Lemma 8 in combination with Lemma 7 implies that, given an antimonotone 3-normalized formula φ, we can build an instance (r, s) of problem IND_{fixed} in FPT-time (even polynomial) such that $f_\varphi = f_{(r,s)}$. Using the findings of Section 4.1, this proves the desired theorem.

▶ **Theorem 9.** IND_{fixed} *and* IND *are* $W[3]$*-complete.*

5 Conclusion

We have determined the complexity of various dependency problems when parameterized by the solution size. Our results imply that these problems do *not* admit FPT algorithms unless the W-hierarchy at least partially collapses. This is unfortunate, the choice of parameter appears to be very natural in the sense that the requirement of a small solution size is regularly met in practice (Figure 1). Notwithstanding our results, one can still obtain FPT algorithms by using *other* parameters. As an example, to solve the problem UNIQUE for a relation r over the schema R (and similar considerations hold for FD and IND), one can consider all subsets of R and check for each whether it is a unique column combination. This takes polynomial time for each of the $2^{|R|}$ subsets. Thus, this leads to an FPT-algorithm with $|R|$ as parameter. This is of course not very satisfying, as assuming $|R|$ to be small is a much stronger assumption than assuming the solution size to be small.

Similarly, one could consider the maximum number d of attributes on which two tuples in a relation r disagree. Then any pair of tuples yields up to d candidate attributes such that at least one of these attributes must be contained in every unique column combination. Thus, one can check whether there is a solution of size k to the problem UNIQUE by using a bounded search tree of height k with nodes of degree at most d. This gives an FPT algorithm with respect to the parameter $d + k$. However, assuming that any two pairs in a relation differ only on a few columns seems to be an unrealistic assumption for most data sets.

We leave it as an open problem for future research to figure out properties of realistic instances that can explain and hopefully even improve the running times of practical methods

for dependency detection in relational databases. For example, by designing a multivariate algorithm with more than one parameter.

Acknowledgement. We would like to thank Sebastian Kruse, Felix Naumann and Thorsten Papenbrock for the interesting discussions that initiated our research on this topic, for their continued support and of course for providing us with the data presented in the introduction. We also want to thank Christoph Keßler for proof-reading an early draft of this paper and Sharon Nemeth for revising the final version.

References

1 Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: a survey. *The VLDB Journal*, 24:557–581, 2015.

2 Tatsuya Akutsu and Feng Bao. Approximating minimum keys and optimal substructure screens. In *Proceedings of the 2nd Annual International Conference on Computing and Combinatorics (COCOON)*, pages 290–299, 1996. doi:10.1007/3-540-61332-3_163.

3 C. Beeri, M. Dowd, R. Fagin, and R. Statman. On the structure of armstrong relations for functional dependencies. *Journal of the ACM*, 31(1):30–46, January 1984. doi:10.1145/2422.322414.

4 Jianer Chen and Fenghui Zhang. On product covering in 3-tier supply chain models: Natural complete problems for $W[3]$ and $W[4]$. *Theoretical Computer Science*, 363(3):278–288, 2006. doi:10.1016/j.tcs.2006.07.016.

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 C.J. Date. *An Introduction to Database Systems*. Addison-Wesley Longman Publishing, Boston, MA, USA, 8th edition, 2003.

7 Scott Davies and Stuart Russell. NP-completeness of searches for smallest possible feature sets. Technical report, AAAI, 1994.

8 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium*, 87:161–178, 1992.

9 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

10 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for $W[1]$. *Theoretical Computer Science*, 141(1&2):109–131, 1995.

11 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.

12 Thomas Eiter and Georg Gottlob. Hypergraph transversal computation and related problems in logic and AI. In *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA)*, pages 549–564, 2002. doi:10.1007/3-540-45757-7_53.

13 Martin Grohe. The parameterized complexity of database queries. In *Proceedings of the 20th Symposium on Principles of Database Systems (PODS)*, pages 82–92, 2001. doi:10.1145/375551.375564.

14 Martti Kantola, Heikki Mannila, Kari-Jouko Räihä, and Harri Siirtola. Discovering functional and inclusion dependencies in relational databases. *International Journal of Intelligent Systems*, 7(7):591–607, 1992. doi:10.1002/int.4550070703.

15 R. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations*, pages 85–103, 1972. URL: http://www.cs.berkeley.edu/~luca/cs172/karp.pdf.

16 D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

17 Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999. doi:10.1006/jcss.1999.1626.

18 Aaron Swartz. MusicBrainz: a semantic web service. *IEEE Intelligent Systems*, 17(1):76–77, Jan 2002. See http://www.musicbrainz.org. doi:10.1109/5254.988466.

A Faster Parameterized Algorithm for Pseudoforest Deletion*

Hans L. Bodlaender[1], Hirotaka Ono[2], and Yota Otachi[3]

1 **Department of Information and Computing Sciences, Utrecht University; and Department of Mathematics and Computer Science, University of Technology Eindhoven, The Netherlands**
h.l.bodlaender@uu.nl

2 **Department of Economic Engineering, Kyushu University, Fukuoka, Japan**
hirotaka@econ.kyushu-u.ac.jp

3 **School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa, Japan**
otachi@jaist.ac.jp

Abstract

A pseudoforest is a graph where each connected component contains at most one cycle, or alternatively, a graph that can be turned into a forest by removing at most one edge from each connected component. In this paper, we show that the following problem can be solved in $O(3^k nk^{O(1)})$ time: given a graph G and an integer k, can we delete at most k vertices from G such that we obtain a pseudoforest? The result improves upon an earlier result by Philip et al. [MFCS 2015] who gave a (nonlinear) $7.56^k n^{O(1)}$-time algorithm both in the exponential factor depending on k as well as in the polynomial factor depending on n.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases pseudoforest deletion, graph class, width parameter, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.7

1 Introduction

In this paper, we consider the Pseudoforest Deletion problem. A pseudoforest is an undirected graph that is obtained from a forest by adding at most one edge to each connected component. In the Pseudoforest Deletion problem, we are given a graph $G = (V, E)$ and an integer k, and ask if there is a set of at most k vertices in G, that, when deleted from G, turns G into a pseudoforest. The Pseudoforest Deletion problem is closely related to the well known Feedback vertex set problem, where we want to delete at most k vertices from a graph so that the graph becomes a forest.

The Pseudoforest Deletion problem was first studied by Philip et al. [12], together with the generalization where each connected component is a tree plus at most ℓ edges. They showed that for each ℓ, the problem to delete at most k vertices such that we obtain such an *ℓ-pseudoforest* has a kernel with $f(\ell)k^2$ vertices. For the Pseudoforest Deletion problem,

* This research was partially supported by the Networks project, funded by the Dutch Ministry of Education, Culture and Science through NWO and by MEXT/JSPS KAKENHI grant numbers 24106004, 24220003, 25730003, 26540005. The third author was partially supported by FY 2015 Researcher Exchange Program between JSPS and NSERC.

i.e., the case that $\ell = 1$, they give a deterministic algorithm with running time $7.56^k n^{O(1)}$.[1] In this paper, we improve upon the latter result, both with respect to the exponential factor in k, as well as in the polynomial factor in n, which is, in our case, linear.

It is easy to see that the Pseudoforest deletion problem belongs to the class of problems studied by Fomin et al. [9], and thus, by these results, the problem has a constant factor polynomial time approximation algorithm, a polynomial kernel (improved to quadratic by the results of Philip et al. [12]), and a randomized algorithm that runs in time $O(c^k n)$ for some constant c. The randomized algorithm is a generalization of an algorithm by Becker et al. [3] for the Feedback Vertex Set problem and a related problem called the Loop Cutset problem. Fomin et al. [9] also give deterministic algorithms running in time $O(2^{O(k)} n \log^2 n)$ and $O(nm)$ time constant factor approximation algorithms for a large class of problems that includes Pseudoforest deletion. If one looks closely at the randomized algorithm by Becker et al. [3] and the generalization by Fomin et al. [9], it follows that one can solve the Pseudoforest deletion problem with a randomized algorithm in $O(4^k nk^{O(1)})$ time.

Our improvement on these two algorithms is based upon the combination of a few different insights and techniques, in particular:

- Positive instances, i.e., graphs that can be turned into a pseudoforest by deleting at most k vertices have treewidth at most $k + 2$.
- The notion of *pseudoforest* has the following *local characterization*: a graph is a pseudoforest if and only if it has an edge orientation such that each vertex has outdegree at most one.
- The local characterization allows us to solve the problem with dynamic programming on a tree decomposition in time that is linear in the number of vertices and single exponential in the treewidth, without the need to use advanced techniques like the cut and count method [8] or the rank based approach [6].
- With help of *convolutions* [15] (see also [4]), the running time of the dynamic programming algorithm is reduced to $O(3^t nt^{O(1)})$ on tree decompositions of width t.
- What remains is the need to find an initial tree decomposition to run the dynamic programming algorithm on. For this, we use a modification of the $O(f(t)n)$ algorithm for Treewidth by Bodlaender [5]. The modification includes the use of *iterative compression* inside one of the subroutines.

It is interesting to contrast our result with the currently best known parameterized algorithms for Feedback Vertex Set: for the Pseudoforest Deletion problem we have a deterministic $O(3^k nk^{O(1)})$ algorithm, while Feedback Vertex Set can be solved in $O(3^k n^{O(1)})$ time with a randomized algorithm [8] and $O(3.63^k n^{O(1)})$ time with a deterministic algorithm [10]; in both cases, the running time is not linear in n.

This paper is organized as follows. In Section 2, we give some preliminary definitions. Section 3 contains a number of graph theoretic observations; in many cases these are not hard to observe, from existing literature or folklore. Section 4 discusses how the Pseudoforest Deletion problem can be solved when a tree decomposition of bounded width is available. This method is used as a subroutine in the main algorithm, that is given in Section 5. The paper ends with some conclusions in Section 6.

[1] They did not specify the exact dependency in n, which is at least quadratic.

2 Preliminaries

When not specified otherwise, a graph $G = (V, E)$ is considered to be undirected, but possibly with selfloops and parallel edges. Allowing selfloops and parallel edges makes the description of the main algorithm easier. An *orientation* of a graph $G = (V, E)$ is a directed graph obtained by giving each edge in G a direction. For a graph $G = (V, E)$ and vertex set $W \subseteq V$, the *subgraph of G induced by W* is denoted by $G[W] = (W, \{e \in E \mid$ both endpoints of e belong to $W\})$.

A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ with T a tree, and $\{X_i \mid i \in I\}$ a collection of subsets (called *bags*) of V, such that

1. $\bigcup_{i \in I} X_i = V$;
2. for all $\{v, w\} \in E$, there is an $i \in I$ with $\{v, w\} \subseteq X_i$
3. for all $v \in V$, the set of nodes $\{i \in I \mid v \in X_i\}$ forms a connected subtree of T.

The *width* of a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G.

For the definition above, if there are parallel edges or selfloops, we can just ignore them, i.e., a tree decomposition of a graph with parallel edges and selfloops is a tree decomposition of the associated simple graph (obtained by keeping only one of each set of parallel edges and removing all selfloops).

In this paper, we also use the related notion of *nice* tree decomposition. In the literature, there are a few variants of this notion that differ in details. In this case, we use the variant with *edge introduce nodes* and leaf bags of size one.

A *nice tree decomposition* is a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ where T is a *rooted* tree, and nodes are of one of the following five different types. With each bag/node in the tree decomposition, we also associate a subgraph of G; the subgraph associated with node i is denoted $G_i = (V_i, E_i)$. We give each type together with how the corresponding subgraph is formed.

- **Leaf** nodes i. i is a leaf of T; $|X_i| = 1$, and $G_i = (\{v_i\}, \emptyset)$ is the graph consisting of the vertex v_i and no edges.
- **Introduce vertex** nodes i. i has one child, say j. There is a vertex v with $X_i = X_j \cup \{v\}$, $v \notin V_j$, and $G_i = (V_j \cup \{v_i\}, E_j)$, i.e., G_i is obtained from G_j by adding v_i as isolated vertex.
- **Introduce edge** nodes i. i has one child, say j. There are two vertices $v, w \in X_i$, $X_i = X_j$, and $G_i = (V_j, E_j \cup \{v, w\})$. I.e., G_i is obtained from G_j by adding an edge between two vertices in $X_i = X_j$. If we have parallel edges, we have one introduce edge node for each parallel edge. E.g., if there are two edges from v to w, we have two edge introduce nodes for the pair v, w; typically, one of these can be the parent of the other in the tree. A selfloop with endpoint v is handled in the same way, i.e., there is an introduce edge node i with $v \in X_i$, and G_i is obtained by adding the selfloop to G_j.
- **Forget** nodes i. i has one child, say j. There is a vertex v with $X_i = X_j - \{v\}$. G_i and G_j are the same graph.
- **Join** nodes i. i has two children, say j_1 and j_2. $X_i = X_{j_1} = X_{j_2}$, $V_{j_1} \cap V_{j_2} = X_i$ and $E_{j_1} \cap E_{j_2} = \emptyset$. $G_i = (V_{j_1} \cup V_{j_2}, E_{j_1} \cup E_{j_2})$. I.e., G_i is obtained by taking the union of G_{j_1} and G_{j_2}, where the vertices in X_i are the intersection of these two graphs.

If r is the root of T, then $G_r = G$.

Restricting a function f to a sub-domain Z is denoted $f|_Z$. With $f + v \to i$ we denote the new function, obtained by adding v to the domain of f, mapping v to i. $f^{v \to i}$ denotes the function, obtaining by changing f by mapping v to i.

A *pseudotree* is a connected graph that is either a tree or obtained by adding one edge to a tree. Note that, as we allow selfloops and parallel edges, this edge may be a selfloop or a parallel edge. A graph is a *pseudoforest*, if each connected component is a pseudotree.

A *pseudoforest deletion set* in a graph $G = (V, E)$ is a set of vertices $W \subseteq V$ such that $G[V - W]$ is a pseudoforest.

A *p-contraction* of an edge $\{v, w\}$ is the operation that identifies v and w, removes the edge $\{v, w\}$, but keeps parallel edges, e.g., if there are edges $\{v, x\}$ and $\{w, x\}$ before the contraction, then x has two parallel edges to the newly formed vertex; if there is an edge parallel to the contracted edge, then this turns into a selfloop. Note that the number of edges of a graph drops by exactly one when doing a p-contraction.

The *c-improved graph* of a graph $G = (V, E)$ is the graph, obtained by adding an edge between each pair of vertices that have at least c common neighbors of degree at most $c + 1$. (We do not take the closure of this operation.)

A vertex v is *simplicial* in a graph $G = (V, E)$ if the neighborhood of v is a clique.

3 Graph theoretic observations

In this section, we give some graph theoretic results that are either folklore or easy to see. The following lemma is a trivial observation.

▶ **Lemma 3.1.** *Let $G = (V, E)$ be a graph. The following statements are equivalent.*

1. *G is a pseudoforest.*
2. *G has an orientation such that each vertex has outdegree at most 1.*

While Lemma 3.1 is an easy observation, it is a key point to our result: being a pseudoforest seems to be a global property, it actually can be expressed by a local property: having an orientation with outdegree at most one allows a dynamic programming algorithm on tree decompositions with three states per vertex, i.e., with tables of size bounded by 3^t, t being the width of the tree decomposition.

▶ **Lemma 3.2.** *Let $G = (V, E)$ be a graph.*

1. *Suppose that there are four or more parallel edges from v to w. Let G' be the graph, obtained from G by removing one parallel edge from v to w. The minimum size of a pseudoforest deletion set in G equals the minimum size of the pseudoforest deletion set of G'.*
2. *Suppose that there are three or more self loops with v as endpoint. Let G'' be the graph, obtained from G by removing one selfloop with v as endpoint. The minimum size of a pseudoforest deletion set in G equals the minimum size of the pseudoforest deletion set of G''.*

Proof. The result follows by observing that if there are three or more parallel edges from v to w then any pseudoforest deletion set must contain v or w, and that if there are two or more selfloops with v as endpoint, then any pseudoforest deletion set contains v. ◀

The following lemma, used in our algorithm is the main reason why we use p-contractions and graphs with parallel edges and self-loops: we do not have such a result when we would use simple graphs and the usual notion of contraction.

▶ **Lemma 3.3.** *Let G' be obtained from G by a p-contraction of the edge $\{v, w\}$. Let x be the vertex resulting from the contraction of $\{v, w\}$. Suppose W is a pseudoforest deletion set in G'.*

1. *If $x \notin W$, then W is a pseudoforest deletion set in G.*
2. *If $x \in W$, then $W - \{x\} \cup \{v, w\}$ is a pseudoforest deletion set in G.*

Proof. The result follows by observing that when we contract an edge from a pseudoforest, we again obtain a pseudoforest. ◀

The following result is folklore. While the folklore result deals with simple graphs, we can build a nice tree decomposition for a graph with parallel edges and selfloops by building a tree decomposition for the underlying simple graph, and then adding the selfloops and parallel edges in the obvious way.

▶ **Lemma 3.4.** *Suppose $G = (V, E)$ is given with a tree decomposition of width k with r bags. Then one can construct a nice tree decomposition of G with $O(kr + |E|)$ bags in $O(k^2 r + |E|k)$ time.*

The following result is a trivial consequence of treewidth folklore. As the construction in the proof is used in the algorithm, we give the constructive proof here.

▶ **Lemma 3.5.** *Let $G = (V, E)$ be a graph.*
1. *If G is a pseudoforest, the treewidth of G is at most 2.*
2. *If there is a set $W \subseteq V$ such that $G[V - W]$ is a pseudoforest, the treewidth of G is at most $2 + |W|$. The corresponding tree decomposition can be computed in $O(n \cdot |W|)$ time.*

Proof.
1. The treewidth of a tree is 1; the treewidth of a tree plus one edge is 2: add one endpoint of the new edge to all bags. The treewidth of a graph equals the maximum treewidth of a connected component, hence the treewidth of a pseudoforest is 2.
2. Take a tree decomposition of width 2 of $G[V - W]$, and add W to all bags. ◀

One ingredient of our algorithm is an approach, first used by Bodlaender [5] to obtain an algorithm for Treewidth that uses $O(f(k)n)$ time, see Theorem 3.6 below. Perković and Reed [11] showed that the result can be improved with respect to factors polynomial in k; for our purposes, the form below suffices.

▶ **Theorem 3.6** (Bodlaender [5])**.** *Let $G = (V, E)$ be a graph and t an integer. At least one of the following three statements is true.*
- *Any maximal matching of G has $\frac{1}{O(t^8)}n$ edges.*
- *The t-improved graph of G has at least $\frac{1}{O(t^2)}n$ simplicial vertices of degree at most t.*
- *The treewidth of G is at least $t + 1$.*

▶ **Lemma 3.7.** *Let G be a graph and let k be an integer. G has a set $X \subseteq V(G)$ of size at most k such that $G - X$ is a pseudoforest if and only if the $k + 3$-improved graph of G has a set X' of size at most k such that $G - X'$ is a pseudoforest.*

Proof. As a subgraph of a pseudoforest is a pseudoforest, the 'if'-direction is trivial.

Suppose G has a set X of size at most k such that $G - X$ is a pseudoforest. Consider two vertices v, w, with at least $k + 3$ common neighbors. We claim that $v \in X$ or $w \in X$. Suppose not. Vertices v and w have at least 3 common neighbors that do not belong to X. We now have five vertices with at least six edges between them, so for any orientation, at least one of these five vertices has outdegree two or more, contradiction. As $v \in X$ or $w \in X$, we can safely add the edge $\{v, w\}$, as $G - X$ remains a pseudoforest. ◀

4 Solving Pseudoforest Deletion on tree decompositions

In this section, we will prove the following result.

▶ **Theorem 4.1.** *Suppose $G = (V, E)$ is given with a tree decomposition of width at most t with $O(n)$ bags. One can find in $O(3^t nt^{O(1)})$ time a minimum size pseudoforest deletion set.*

For easier explanation of the algorithm, we will first derive an algorithm that uses $O(4^t nt^{O(1)})$ time and solves the decision problem, i.e., computes the *size* of the minimum pseudoforest deletion set. Then, with help of the *convolutions* technique for tree decompositions, introduced by van Rooij et al. [15], we obtain a decision problem with $O(3^t nt^{O(1)})$ running time. At the end, we discuss how we can compute within the same time bound also the corresponding minimum size pseudoforest deletion set.

An algorithm that runs in $O(4^t nt^{O(1)})$ time

We first transform the tree decomposition to a nice tree decomposition, which has $O(tn)$ bags.

Recall that we associate a subgraph of G, G_i with each node i in the nice tree decomposition. A *partial solution* for a node $i \in I$ is a pair (Y, Λ), with $Y \subseteq V_i$ a set of vertices and Λ an orientation of E_i such that each vertex in $V_i - Y$ has at most one outgoing arc in Λ. If r is the root of the nice tree decomposition, then a partial solution for r is called a *solution*. We say a solution (Y, Λ) *extends* partial solution (Y', Λ') for i if $Y' = Y \cap V_i$ and Λ' is the restriction of Λ to E_i.

The *characteristic* of a partial solution (Y, Λ) for i is the function $f : X_i \to \{X, 0, 1\}$, such that

- For all $v \in X_i$, $f(v) = X$ if and only if $v \in Y$.
- If $v \in X_i$ and $f(v) = 0$, then v has no outgoing arcs in Λ.
- If $v \in X_i$ and $f(v) = 1$, then v has exactly one outgoing arc in Λ.

The main ingredient of the algorithm is to compute for each node in i a table (function) T_i, in postorder, i.e., we compute the table for a node after the tables for its children are known. A table T_i maps each function $f : X_i \to \{0, 1, X\}$ to an nonnegative integer or to ∞, in the following way.

Suppose i is a bag in a nice tree decomposition, with corresponding set X_i and subgraph G_i. For a function $f : X_i \to \{0, 1, X\}$, $T_i(f)$ equals the minimum of $|Y|$ over all partial solutions (Y, Λ) at i with characteristic f. If no such partial solution exists, then $T_i(f) = \infty$.

The following claim trivially holds by Lemma 3.1, and shows how to obtain the answer to the decision version of the Pseudoforest Deletion problem given T_r for the root r of the tree decomposition.

▶ **Claim 4.2.** *Let r be the root of a nice tree decomposition of $G = (V, E)$. The minimum size of a pseudoforest deletion set in G equals the minimum of $T_r(f)$ over all $f : X_r \to \{0, 1, X\}$.*

We will now discuss for each of the types of nodes in a nice tree decomposition how to compute the table T_i, given the tables of the children of the node.

Leaf nodes. Let i be a leaf node, with $X_i = \{v\}$. Now, if $f(v) = 0$, then $T_i(f) = 0$; if $f(v) = 1$, then $T_i(f) = \infty$, and if $f(v) = X$, then $T_i(f) = 1$.

Introduce vertex nodes. Suppose i is an introduce vertex node i with child j with $X_i = X_j \cup \{v\}$.

As the degree of v in G_i is 0, for each f with $f(v) = 1$, we have $T_i(f) = \infty$, as there are no partial solutions with v having outdegree 1.

For a function f with $f(v) = 0$, we have $T_i(f) = T_i(f|_{X_i})$; and for functions f with $f(v) = X$, we have $T_i(f) = T_i(f|_{X_i}) + 1$ — we can just extend any partial solution for G_j by either not placing v in the pseudoforest deletion set, in which case v has outdegree 0; or placing v in the pseudoforest deletion set, in which case v is mapped to X and the size of the set is increased by one.

Introduce edge nodes. Consider an introduce edge node i with child j, where we introduce an edge with endpoints v and w. Note that we allow parallel edges and selfloops; the subroutine below is also correct in case the introduced edge is parallel to an existing edge or is a selfloop (i.e., $v = w$.)

For each $f : X_i \to \{0, 1, X\}$, we consider the two cases in which $\{v, w\}$ can be oriented. We then obtain the following cases; for brevity, we omit the isomorphic cases with the roles of v and w switched.

- If $f(v) = X$ and $f(w) = X$, then $T_i(f) = T_j(f)$.
- If $f(v) = X$ and $f(w) = 0$, then $T_i(f) = T_j(f)$. (We must orient the edge from v to w.)
- If $f(v) = X$ and $f(w) = 1$, then $T_i(f) = \min\{T_j(f), T_j(f^{w\to 0})\}$.
- If $f(v) = 1$ and $f(w) = 1$, then $T_i(f) = \min\{T_j(f^{v\to 0}), T_j(f^{w\to 0})\}$.
- If $f(v) = 1$ and $f(w) = 0$, then $T_i(f) = T_j(f^{v\to 0})$. (We must orient the edge from v to w, and thus v has outdegree 0 in the corresponding orientation of G_j.)
- If $f(v) = 0$ and $f(w) = 0$, then $T_i(f) = \infty$. (No orientation with both v and w having outdegree 0 is possible.)

Forget nodes. Let i be a forget node with child j with $X_j = X_i \cup \{v\}$. Then $T_i(f) = \min\{T_j(f + v \to 0), T_j(f + v \to 1), T_j(f + v \to X)\}$.

Join nodes. Suppose i is a join node with children j_1 and j_2. The following claim gives that we can compute T_i, given T_{j_1} and T_{j_2} in time $O(4^t t^{O(1)})$. As said, we later will improve the exponential factor to 3^t with help of convolutions.

▶ **Lemma 4.3.** *$T_i(f)$ is the minimum over all f_1 and f_2 of $T_{j_1}(f_1) + T_{j_2}(f_2) - \alpha$, where*

- *For all $v \in X_i$, $f(v) = X \Leftrightarrow f_1(v) = X \Leftrightarrow f_2(v) = X$.*
- *For all $v \in X_i$, $f(v) = 0 \Leftrightarrow f_1(v) = 0 \Leftrightarrow f_2(v) = 0$.*
- *For all $v \in X_i$, if $f(v) = 1$ then either $f_1(v) = 1$ and $f_2(v) = 0$, or $f_1(v) = 0$ and $f_2(v) = 1$.*
- *$\alpha = |\{v \in X_i \mid f(v) = X\}|$.*

Proof. The proof follows standard techniques for dynamic programming on tree decompositions. The number of elements in the vertex deletion set Z in G_i equals the number of elements in Z in G_{j_1} plus the number of elements in Z in G_{j_2}, minus the number of elements in Z in both — the latter number is α; we thus have to subtract α once to prevent counting vertices in $Z \cap X_i$ twice. ◀

The claim above shows that we can compute T_i given T_{j_1} and T_{j_2} in $O(4^t t^{O(1)})$ time: for each $v \in X_i$, there are four combinations to consider: $f_1(v) = f_2(v) = X$; $f_1(v) = f_2(v) = 0$; $f_1(v) = 1$ and $f_2(v) = 0$; $f_1(v) = 0$ and $f_2(v) = 1$. This gives $4^{|X_i|}$ combinations in total;

for each, look up the table entries in T_{j_1} and T_{j_2}, compute the value which arrives when we combine these entries. We initialize each value in T_i to ∞, and for each computed value, we set the value of the corresponding entry in T_i to the minimum of its current value and the just computed value.

Pseudoforest Deletion is finite integer index

We now discuss a small modification, that deletes some table entries which will never lead to an optimal solution. The modification shows that Pseudoforest Deletion is *finite integer index* (see [7]), and in fact, has the *de Fluiter property*, as defined by van Rooij [14, Chapter 11.2]. We do not give the formal definition of this property, but state the elements that are needed for our algorithm.

▶ **Lemma 4.4.** *Let i be a bag, and let f_X be the function, that maps each element of X_i to X.*

1. *For all $f : X_i \to \{0, 1, X\}$, $T_i(f_X) \leq T_i(f) + |X_i|$.*
2. *Let $f : X_i \to \{0, 1, X\}$. If $T_i(f) > T_i(f_X)$, then no partial solution at i with characteristic f will extend to an optimal solution.*

As a result, we have that we can ignore in our computations, all values for T_i that are larger than $T_i(f_X)$ without affecting the correctness of the algorithm. In the implementation, we just delete these entries from the tables or set there values to $T_i(f_X) + 1$. As a result, all values in a table T_i are in the range $T_i(f_X) - |X_i|, \ldots, T_i(f_X)$.

Using convolutions for Join Nodes

In order to speed up the dynamic programming algorithm, we use convolutions. The use of this technique in the setting of dynamic programming on tree decompositions was introduced by van Rooij et al. [15, 14].

Obtaining a constructive algorithm

As for many dynamic programming algorithms, constructing an optimal *solution* is done after computing its value, by traversing the tree top-down. We first select an entry from the root table T_r with minimum value, i.e., a function $f : X_r \to \{0, 1, X\}$ with $T_r(f) = \min_{f' : X_r \to \{0,1,X\}} T_r(f')$. We construct a solution corresponding to f by finding (a) 'corresponding' table entries in the child nodes, constructing partial solutions corresponding to these nodes, and placing the vertices in X_r with $f(v) = X$ in the pseudoforest deletion set. What are 'corresponding' table entries is different for the different types of nodes of a nice tree decompositions; e.g., for a forget node an entry corresponding to f is where the minimum in $\min\{T_j(f + v \to 0), T_j(f + v \to 1), T_j(f + v \to X)\}$ is attained. Obtaining these entries is trivial, except for join nodes.

For a join node i, we must solve the following problem: we are given an $f : X_i \to \{0, 1, X\}$, and must find f_1 and f_2 as in Lemma 4.3. It is easy to see, and for our purposes sufficient to notice that we can try all combinations f_1 and f_2, such that for all $v \in X_i$:

- If $f(v) = X$, then $f_1(v) = f_2(v) = X$.
- If $f(v) = 0$, then $f_1(v) = f_2(v) = 0$.
- If $f(v) = 1$, then ($f_1(v) = 1$ and $f_2(v) = 0$) or ($f_1(v) = 0$ and $f_2(v) = 1$).

These are at most 2^{t+1} different combinations to try; for each, we can see if these combine to f as in Lemma 4.3 in $O(t^{O(1)})$ time. With $O(n)$ nodes in the tree decomposition, the time to construct a solution after all tables T_i have been computed is bounded by $O(n2^t t^{O(1)})$.

(As a side remark, using self reduction (see [14, Chapter 12]) it is possible to avoid the factor exponential in t here and perform this step in $O(nt^{O(1)})$ time, but as the asymptotic running time is not dominated by this step, we prefer to give the simpler argument.)

Note that the algorithm remains correct when we run it on multigraphs with possible parallel edges and selfloops. This ends the proof of Theorem 4.1.

5 Main algorithm

In this section, we give the main algorithm and prove that it attains the $O(3^k nk^{O(1)})$ time bound. We first give the general outline of the algorithm (Section 5.1); then discuss two subroutines for two cases in Sections 5.2 and 5.3. Some implementation details and the time analysis will be discussed in Section 5.4.

5.1 Outline

We now give the overall outline of the algorithm. We have a recursive algorithm, that follows the cases of Theorem 3.6. In addition, we have a base case: if we have a graph $G = (V, E)$ with at most k vertices, we can just return V and are done. So suppose $|V| > k$. Let $t = k + 2$.

The algorithm first computes an arbitrary maximal matching M. If this matching M is large enough, i.e., has size $\frac{1}{O(t^8)}n = \frac{1}{O(k^8)}n$ as in the first case of Theorem 3.6, then we proceed with the subroutine discussed in Section 5.2. If this matching is not of this size, we compute the $k+3$-improved graph, and then find the set of simplicial vertices S of degree at most $k+3$. If set S is large enough, i.e., has size $\frac{1}{O(t^2)}n = \frac{1}{O(k^2)}n$ as in the second case of Theorem 3.6, then we proceed with the subroutine discussed in Section 5.3. If neither M nor S is large enough, we halt and reject: by Theorem 3.6, we know that G has treewidth at least $t + 1 = k + 3$, and hence G has no pseudoforest deletion set of size at most k; see Lemma 3.5.

We will discuss how each of the subroutines solves the problem when the corresponding case holds, and how this leads to an algorithm with the stated time bounds below.

5.2 Graphs with a large maximal matching

In this section, we suppose that we have a (maximal) matching M in $G = (V, E)$ with size $\frac{1}{O(k^8)}n$. We give a subroutine that either gives a pseudoforest deletion set of size at most k, or decides that G has no such set.

Let $G_M = (V_M, E_M)$ be the graph obtained by p-contracting all edges in M, i.e., we contract the edges but keep parallel edges and selfloops.

Now, recursively solve the problem on G_M. From Lemma 3.3, we have:

▶ **Lemma 5.1.** *Suppose G has a pseudoforest deletion set X. Let X_M be the set of vertices in G_M obtained from X by the p-contraction of edges. Then X_M is a pseudoforest deletion set of G.*

From Lemma 5.1, it follows that if our recursive call to G_M tells us that G_M has no pseudoforest deletion set of size at most k, then also G has no pseudoforest deletion set of size at most k, and thus we say 'no' and halt.

So, now assume that our recursive call gives us a pseudoforest deletion set S of G_M of size at most k. Let S' be the set of vertices that are contracted to S; i.e., if a vertex $v \in S$ is

the result of contracting an edge from x to y, then we have $x, y \in S'$; if a vertex $v \in S$ is not the result of a contraction, then we place v in S'.

▶ **Claim 5.2.** *S' is a pseudoforest deletion set of size at most $2k$.*

We thus build S', and now we apply *iterative improvement*. Number the vertices in S', i.e., write $S' = \{v_1, v_2, \ldots, v_r\}$; we have $r \leq 2k$. Write $V_i = (V - S') \cup \{v_1, v_2, \ldots, v_i\}$, and $G_i = G[V_i]$. Note that $\{v_1, \ldots, v_k\}$ is a pseudoforest deletion set of G_k. Set $W = \{v_1, \ldots, v_k\}$.

Now, for $i = k + 1$ to r, do iteratively the following steps.

- Set $S = \{v_i\} \cup W$.
- *An invariant of the algorithm is that S is a pseudoforest deletion set of size at most $k + 1$ of G_i.*
- Compute a tree decomposition of G_i of width at most $k + 3$:
  - $G_i - S$ is a pseudoforest, so we can build a tree decomposition of $G_i - S$ of width at most 2 in linear time.
  - Add S to all bags of this tree decomposition.
- Run the algorithm of Theorem 4.1 and solve the Pseudoforest Deletion problem on G_i with parameter k.
- If this algorithm returns that G_i has no pseudoforest deletion set of size at most k, then G has no pseudoforest deletion set of size at most k, and we say 'no' and halt.
- Otherwise, let W be the pseudoforest deletion set of G_i that was obtained.

When we are done, we either have decided that G has no pseudoforest deletion set of size k, or we obtained a pseudoforest deletion set W of $G_r = G$ of size at most k.

5.3 Improved graphs with many simplicial vertices

We now suppose that we have the $k+3$-improved graph G', and a set Z with $\frac{1}{O(t^2)}n = \frac{1}{O(k^2)}n$ simplicial vertices of degree at most $k + 3$.

We recursively run the algorithm on $G' - Z$. If $G' - Z$ has no pseudoforest deletion set of size at most k, then G' has none, and hence, by Lemma 3.7 G has no pseudoforest deletion set of size at most k; we can halt and answer 'no'.

Otherwise, we obtain a pseudoforest deletion set of size at most k of $G' - Z$. We can thus build a tree decomposition of width at most $k + 2$ of $G' - Z$, as in Lemma 3.5. Build a tree decomposition of width at most $k + 2$ of G', by adding a bag with vertex set $N[z]$ for all $z \in Z$; making this bag adjacent to a bag that contains the clique $N(z)$. This is identical to an operation from the algorithm in [5]. As G is a subgraph of G', we now have a tree decomposition of G of width at most $k + 2$, and thus can run the dynamic programming algorithm from Theorem 4.1 on this latter tree decomposition. Return the answer of this algorithm.

5.4 Implementation and time analysis

We discuss here some implementation details. Each of the steps except for the recursive calls and the call to the dynamic programming algorithm of Theorem 4.1 can be done in $O(nk^{O(1)})$ time: finding a maximal matching and contracting a maximal matching is trivially within this time bound; how to find the improved graph, the simplicial vertices of bounded degree, and how to transform a tree decomposition of the graph without these simplicial vertices to one with the simplicial vertices (the main steps from Sections 5.2 and 5.3) in $O(nk^{O(1)})$ time is shown in [5]; we can use the same procedures as in [5] here. We call the

$O(n3^k k^{O(1)})$ dynamic programming algorithm $O(k)$ times, and thus the time per recursive call is bounded by $O(n3^k k^{O(1)})$. One call of the procedure makes one recursive call on a graph where we lost a fraction of $\frac{1}{O(k^8)}$ of the vertices, and thus our running time satisfies the following recurrence:

$$T(n) = T\left(n - \frac{1}{O(k^8)}n\right) + O(n3^k k^{O(1)}).$$

This resolves to $T(n) = O(n3^k k^{O(1)})$, which shows our main result Theorem 4.1.

▶ **Theorem 5.3.** *The problem, given a graph G and integer k, to decide if G has a pseudoforest deletion set of size at most k, and if so, find one, can be solved in $O(n3^k k^{O(1)})$ time.*

6 Concluding remarks

In this paper, we gave a fast parameterized algorithm for the Pseudoforest Deletion problem, with a running time with the currently best known factor depending on the parameter k, and a factor, linear in the number of vertices.

It is an interesting open problem whether this is (up to factors, polynomial in k) optimal, assuming the (Strong) Exponential Time Hypothesis, or whether a result similar to the lower bound proofs by Cygan et al. [8] can show that there is no $O((3-\epsilon)^t n^{O(1)})$ algorithm for Pseudoforest Deletion on graphs given with a tree (or path) decomposition of width t; compare the similar result for Feedback Vertex Set in [8].

A generalization of the Pseudoforest Deletion problem is the ℓ-Pseudoforest Deletion problem; a graph is an ℓ-pseudoforest, if it can be obtained from a forest by adding at most ℓ edges to each tree. It seems that the problem is harder when $\ell > 1$, as there is no apparent 'local formulation', whereas for $\ell = 1$, we have the formulation from Lemma 3.1. Thus, we wonder whether there exist deterministic algorithms for ℓ-Pseudoforest Deletion that run in $O(c_\ell^k n)$ time for constant c_ℓ depending on ℓ. Philip et al. [13] show that for every ℓ, ℓ-Pseudoforest Deletion has a kernel with $O(k^2)$ vertices. Given the local nature of Pseudoforest Deletion, it is interesting to see if there exists a kernel for it with a linear number of vertices.

Our result also implies a 2-approximation algorithm for Feedback Vertex Set, see below. There exist polynomial-time[2] 2-approximation algorithms for this problem [2, 1]; our algorithm uses linear time at the cost of a factor, exponential in k. The result can possibly be used as a first step in an fpt algorithm for Feedback Vertex Set using iterative compression, aiming at an algorithm that is efficient both in the term depending on k as well as in the term depending on n.

▶ **Corollary 6.1.** *There is a 2-approximation algorithm for* Feedback Vertex Set *that runs in $O(n3^k k^{O(1)})$ time.*

Proof. Run the algorithm of Theorem 5.3. If G has no pseudoforest deletion set of size at most k, then G also has no feedback vertex set of size at most k. Otherwise, let X be a pseudoforest deletion set of size at most k. If $G - X$ contains more than k cycles, then $G - X$ has no feedback vertex set of size k; otherwise, choose a set Y with one vertex per cycle in $G - X$; $X \cup Y$ is a feedback vertex set in G of size at most $2k$. ◀

[2] $O(m + n \log n)$-time [2] and $O(\min\{n^2, m \log n\})$-time [1].

References

1. V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12:289–297, 1999.
2. A. Becker and D. Geiger. Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artificial Intelligence*, 83:167–188, 1996.
3. Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized algorithms for the loop cutset problem. *Journal of Artificial Intelligence Research*, 12:219–234, 2000.
4. Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: Fast subset convolution. In *Proceedings of the 39th Annual Symposium on Theory of Computing, STOC 2007*, pages 67–74, 2007.
5. Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.
6. Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.
7. Hans L. Bodlaender and Babette van Antwerpen-de Fluiter. Reduction algorithms for graphs of small treewidth. *Information and Computation*, 167:86–119, 2001.
8. 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 *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011*, pages 150–159, 2011.
9. Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar F-deletion: Approximation, kernelization and optimal FPT algorithms. In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science, FOCS 2012*, pages 470–479, 2012. doi:10.1109/FOCS.2012.62.
10. Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Inf. Process. Lett.*, 114(10):556–560, 2014.
11. Ljubomir Perković and Bruce Reed. An improved algorithm for finding tree decompositions of small width. *International Journal of Foundations of Computer Science*, 11:365–371, 2000.
12. Geevarghese Philip, Ashutosh Rai, and Saket Saurabh. Generalized pseudoforest deletion: Algorithms and uniform kernel. In *40th International Symposium on Mathematical Foundations of Computer Science 2015, MFCS 2015*, volume 9235 of *Lecture Notes in Computer Science*, pages 517–528. Springer Verlag, 2015.
13. Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Transactions on Algorithms*, 9(1):11, 2012.
14. Johan M. M. van Rooij. *Exact Exponential-Time Algorithms for Domination Problems in Graphs*. PhD thesis, Utrecht University, 2011. URL: dspace.library.uu.nl/bitstream/handle/1874/205442/rooij.pdf.
15. Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Proceedings of the 17th Annual European Symposium on Algorithms, ESA 2009*, pages 566–577. Springer Verlag, Lecture Notes in Computer Science, vol. 5757, 2009.

Optimal Dynamic Program for r-Domination Problems over Tree Decompositions*

Glencora Borradaile[1] and Hung Le[2]

1 **Department of Electrical Engineering and Computer Science, Oregon State University, USA**
glencora@eecs.orst.edu

2 **Department of Electrical Engineering and Computer Science, Oregon State University, USA**
lehu@onid.oregonstate.edu

Abstract

There has been recent progress in showing that the exponential dependence on treewidth in dynamic programming algorithms for solving NP-hard problems is optimal under the Strong Exponential Time Hypothesis (SETH). We extend this work to r-domination problems. In r-dominating set, one wishes to find a minimum subset S of vertices such that every vertex of G is within r hops of some vertex in S. In connected r-dominating set, one additionally requires that the set induces a connected subgraph of G. We give a $O((2r+1)^{\mathrm{tw}}n)$ time algorithm for r-dominating set and a randomized $O((2r+2)^{\mathrm{tw}}n^{O(1)})$ time algorithm for connected r-dominating set in n-vertex graphs of treewidth tw. We show that the running time dependence on r and tw is the best possible under SETH. This adds to earlier observations that a "+1" in the denominator is required for connectivity constraints.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases r-dominating set, Exponential Time Hypothesis, Dynamic Programming

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.8

1 Introduction

There has been recent progress in showing that the exponential dependence on treewidth in dynamic programming algorithms for solving NP-hard problems is optimal under the Strong Exponential Time Hypothesis (SETH) [12]. Lokshtanov, Marx and Saurabh showed that for a wide variety of problems with local constraints, such as maximum independent set, minimum dominating set and q-coloring, require $\Omega^*((2-\epsilon)^{\mathrm{tw}})$, $\Omega^*((3-\epsilon)^{\mathrm{tw}})$ and $\Omega^*((q-\epsilon)^{\mathrm{tw}})$ time in graphs of treewidth tw, where Ω^* hides polynomial dependence on the size of the graph [15]; these lower bounds met the best-known upper bounds for the same problems. For problems with connectivity constraints, such as connected dominating set, some thought that a dependence of $\mathrm{tw}^{\mathrm{tw}}$ would be required. Cygan et al. showed that this is not the case, giving tight upper and lower bounds for the dependence on treewidth for many problems, including connected dominating set [6]. They also observed that the base of the constant increased by one when adding a connectivity constraint. For example, vertex cover has tight upper and lower bounds of $O^*(2^{\mathrm{tw}})$ and $\Omega^*((2-\epsilon)^{\mathrm{tw}})$ while connected vertex cover has tight

* This material is based upon work supported by the National Science Foundation under Grant No. CCF-1252833.

Table 1 Old and new results on the domination problem

	Lower Bound	Reference	Upper Bound	Reference
DS	$\Omega^*((3-\epsilon)^{\mathrm{tw}})$	[15]	$O^*(3^{\mathrm{tw}})$	[19]
CDS	$\Omega^*((4-\epsilon)^{\mathrm{tw}})$	[6]	$O^*(4^{\mathrm{tw}})$	[6]
rDS	$\Omega^*(2r+1)^{(1-\epsilon)\mathrm{tw}})$	Theorem 3	$O^*((2r+1)^{\mathrm{tw}})$	Theorem 2
rCDS	$\Omega^*(2r+2)^{(1-\epsilon)\mathrm{tw}})$	Theorem 5	$O^*((2r+2)^{\mathrm{tw}})$	Theorem 4

upper and lower bounds of $O^*(3^{\mathrm{tw}})$ and $\Omega^*((3-\epsilon)^{\mathrm{tw}})$. Similarly, dominating set has tight upper and lower bounds of $O^*(3^{\mathrm{tw}})$ and $\Omega^*((3-\epsilon)^{\mathrm{tw}})$ while connected dominating set has tight upper and lower bounds of $O^*(4^{\mathrm{tw}})$ and $\Omega^*((4-\epsilon)^{\mathrm{tw}})$.

1.1 Generalization to r-domination

In this paper, we show that this pattern of dependence extends to domination problems over greater distances. The r-dominating set (rDS) problem is a natural extension of the dominating set (DS) problem, in which, given a graph G of n vertices, the goal is to find a minimum subset S of vertices such that every vertex of G is within r hops of some vertex in S. Likewise, the connected r-dominating set (rCDS) is the connected generalization of connected dominating set (CDS). We show that rDS can be solved in $O^*((2r+1)^{\mathrm{tw}})$ time and that rCDS can be solved in $O^*((2r+2)^{\mathrm{tw}})$ time. Further, we show that these upper bounds are tight, assuming SETH, even when r is non-constant. We note that our results generalize the previous results listed in Table 1 when $r = O(1)$ since we can reformulate our lower bound as $\Omega^*(2r+1-\epsilon')^{\mathrm{tw}})$ for rDS problem and as $\Omega^*(2r+2-\epsilon')^{\mathrm{tw}})$ for rCDS problem by setting $\epsilon = 1 - \frac{\ln(2r+1-\epsilon')}{\ln(2r+1)}$.

1.2 Notation

We denote the input graph with vertex set V and edge set E by $G = (V, E)$ and use n to denote the number of vertices. Edges of the graph are undirected and unweighted. For two vertices u, v, we denote the shortest distance and path between them by $d_G(u, v)$ and $P_G(u, v)$. Given a subset of vertices S and u a vertex of G, we define $d_G(u, S) = \min_{v \in S}\{d_G(u, v)\}$ and $P_G(u, S) = P_G(u, v)$ where v is a vertex in S such that $d(u, S) = d(u, v)$. We omit the subscript G when G is clear from context. $G[S]$ is the subgraph induced by S. A set of vertices D is an r-dominating set if for every vertex $v \in V$, there exists $u \in D$ such that $d_G(u, v) \leq r$.

▶ **Definition 1** (Tree decomposition). *A tree decomposition* of G is a tree T whose nodes are subsets X_i (so-called bags) of V satisfying the following conditions:

1. The union of all sets X_i is V.
2. For each edge $(u, v) \in E$, there is a bag X_i containing both u, v.
3. For a vertex $v \in V$, all the bags containing v make up a subtree of T.

The *width* of a tree decomposition T is $\max_{i \in T} |X_i| - 1$ and the treewidth of G is the minimum width among all possible tree decompositions of G. We will assume throughout that graph G has treewidth tw and that we are given a tree decomposition of G of width tw.

A *path decomposition* is a tree decomposition whose underlying structure is a path. The pathwidth of a path decomposition and a graph G is defined the same as treewidth.

1.2.1 Upper and lower bounds for r-dominating set

The algorithm we give is a generalization of the $O(3^{\mathrm{tw}}\mathrm{tw}^2 n)$-time algorithm for DS given by van Rooij, Bodlaender and Rossmanith [19].

▶ **Theorem 2.** *There is an $O((2r+1)^{\mathrm{tw}+1}n)$-time algorithm for rDS in graphs G with n vertices and treewidth* tw*.*

Demaine et al. [8] gave an algorithm with running time $O((2r+1)^{\frac{3}{2}\mathrm{bw}}n)$ for rDS in graphs of branchwidth bw; since branchwidth and treewidth are closely related by the inequality $\mathrm{bw} \leq \mathrm{tw}+1 \leq \lfloor \frac{3}{2}\mathrm{bw} \rfloor$ (for which there are tight examples) [16], our algorithm improves the exponential dependence. Our proof of the corresponding lower bound uses a high level constructionz of Lokshtanov, Marx and Saurabh for DS [15] to get around the case when $\log(2r+1)$ is not an integer. However, to handle a wide range of values r, the gadgets we require are non-trivial. We prove the following in Section 3.

▶ **Theorem 3.** *If r-dominating set can be solved in $(2r+1)^{(1-\epsilon)\mathrm{pw}}n^{O(1)}$ time in a graph with pathwidth* pw *and n vertices for every $\epsilon < 1$, then there is a $\delta < 1$ such that SAT instances of n_0 variables can be solved in $O^*(2^{\delta n_0})$.*

1.2.2 Upper and lower bounds for connected r-dominating set

As with the algorithms for connectivity problems with singly-exponential time dependence on treewidth as introduced by Cygan et al. [6], our algorithm for rCDS is a randomized Monte-Carlo algorithm. We note that there exists deterministic algorithms with singly-exponential time dependence on treewidth for connected domination problems [2, 11]. However, these algorithms have worse exponential time than the randomized algorithm that we present. As for rDS, we include the details of this upper bound in Appendix B:

▶ **Theorem 4.** *There is a $O^*((2r+2)^{\mathrm{tw}+1})$-time true-biased Monte-Carlo algorithm that decides rCDS for graphs of treewidth* tw*.*

The gadget construction from the lower bound of Cygan et al. [6] for $r = 1$ assigns truth values to vertices of the gadget, and hence, it is not immediately extended to $r \geq 2$. We design a new construction that assigns truth values to edges of the gadget. Furthermore, we employ the global construction in the proof of Theorem 3 to get around the case when $\log(2r+2)$ is not an integer. It turns out that our construction works for a wide range of values of r. The following theorem is proved in Section 5.

▶ **Theorem 5.** *If connected r-dominating set can be solved in $(2r+2)^{(1-\epsilon)\mathrm{pw}}n^{O(1)}$ time in a graph with pathwidth* pw *and n vertices for every $\epsilon < 1$, then there is a $\delta < 1$ such that SAT instances of n_0 variables can be solved in $O^*(2^{\delta n_0})$.*

1.3 Motivation

Algorithms for problems in graphs of bounded treewidth are useful as subroutines in many approximation algorithms for graphs having bounded local treewidth [10]; specifically, polynomial-time approximation schemes (PTASes) for many problems, including dominating set, TSP and Steiner tree, in planar graphs and graphs of bounded genus all reduce to the same problem in a graph of bounded treewidth whose width depends on the desired precision [4, 5, 13, 1]. For sufficiently small r, Baker's technique and Demaine and Hajiaghayi's bidimensionality framework imply PTASes for rDS and rCDS (respectively) [1, 7]. For larger

values of r, *approximate* r-domination can be achieved by the recent bi-criteria PTAS due to Eisenstat, Klein and Mathieu [9]; they guarantee a $(1+\epsilon)r$-dominating set of size at most $1+\epsilon$ times the optimal r-dominating set. It is an interesting open question of whether a true PTAS (without approximating the domination distance) can be achieved for rDS in planar graphs for arbitrary values of r. We also note that the bi-criteria PTAS of Eisenstat, Klein and Mathieu [9] is not an *efficient* PTAS, one which the degree of the polynomial in n (the size of the graph) does not depend on the desired precision, ϵ. Our new lower bounds suggest that, for large r, it may not be possible to design an efficient PTAS for rDS without also approximating the domination distance, since the $O^*(r^{\mathrm{tw}})$ run-time of the dynamic program becomes an $O^*(r^{1/\epsilon})$ run time for the PTAS.

2 Algorithm for r-dominating Set

In this section, we sketch the dynamic programming algorithm to find an optimal r-dominating set and leave the details in Appendix A. To simplify the dynamic program, we will use a *nice tree decomposition*[1]. Kloks shows how to make a tree decomposition *nice* in linear time with only a constant factor increase in space (Lemma 13.1.2 [14]).

We denote the size of the bag X_i by n_i. We use V_i to denote the set of vertices in descendant bags of X_i. The dynamic programming table A_i for a node X_i of the tree decomposition is indexed by bags of the tree decomposition and all possible *distance-labelings* of the vertices in that bag. For a vertex v in bag X_i, a positive distance label for v indicates that v is r-dominated at that distance in $G[V_i]$, and a negative distance label for v indicates that v should be r-dominated at that distance in G but not in $G[V_i]$.

For an r-dominating set D, we say that D *induces* the labeling $c : X_i \to [-r, r]$ for bag X_i such that:

$$c(u) = \begin{cases} d_G(u, D) & \text{if } d_{G[V_i]}(u, D \cap V_i) = d_G(u, D) \\ -d_G(u, D) & \text{otherwise} \end{cases}$$

If D induces the labeling c, the set $D \cap V_i$ is the partial solution associated with c. We limit ourselves to labelings that are locally valid as optimal r-dominating sets in G induce locally valid labelings at any bag of the tree decomposition; c is *locally valid* if $|c(u) - c(v)| \leq 1$ for any two adjacent vertices $u, v \in X_i$. If a labeling c is not locally valid, we define $A_i[c] = -\infty$.

We show how to populate A_i from the populated tables for the child/children of X_i. Over the course of the dynamic programming, we maintain the following correctness invariant at all bags of the of the tree T:

Correctness Invariant. For any locally valid labeling c of X_i, we will maintain:

$$A_i[c] = \min_{\substack{D \subseteq V \\ D \text{ induces } c}} |D \cap V_i| \,. \tag{1}$$

Intuitively, $A_i[c]$ is the minimum size of the partial solution associated with labeling c. From the root bag X_0, we can extract the minimum size of an r-dominating set from the root's table. This is the optimal answer by the correctness invariant and the definition of *induces*.

We define an ordering $\preceq$ on labels for single vertices: $\ell_1 \preceq \ell_2$ if $|\ell_1| = |\ell_2|$ and $\ell_1 \leq \ell_2$. We extend this ordering to labelings c, c' for a bag of vertices X_i by saying $c \preceq c'$ if $c(u) \preceq c'(u)$ for all $u \in X_i$.

[1] The definition of a nice tree decomposition is in Appendix A

▶ **Lemma 6** (Ordering Lemma). *Let D' and D be two r-dominating sets that induce two labelings c and c' of a bag X_i such that (i) $|D \cap X_i| = A[c]$, (ii) $|D' \cap X_i| = A[c']$, (iii) $c' \preceq c$ and (iv) D has minimum size among r-dominating sets that induce c. If $A[c'] > A[c]$, then $|D'| > |D|$.*

The Ordering Lemma tells us that if $c' \preceq c$ and $A[c'] > A[c]$, the r-dominating set that induces c' cannot be the minimum r-dominating set of the graph. Thus, we will maintain the following Ordering Invariant to reduce the number of cases we need to consider in populating the table of an introduce node and join node.

Ordering Invariant. If two labelings c and c' of X satisfy $c' \preceq c$, then $A_i[c'] \leq A_i[c]$.

We sketch some ideas to show how to update the dynamic programming tables here and leave the rest in Appendix A. There are four types of nodes in a nice tree decomposition: leaf nodes, forget nodes, introduce nodes and join nodes and we will handle each type of nodes separately. Leaf nodes and forget nodes can be handled straightforwardly and more care is needed when when handling introduce nodes and join nodes. For introduce nodes, the introduced vertices can r-dominate the vertices that are not r-dominated in the subgraph induced by the descendant bags. Thus, we may need to change negative distance labelings to positive distance labelings and that introduces some complication.

For join nodes, we need two intermediate tables, called the *indication table* and the *convolution table* that can be computed efficiently from children tables. Suppose we are given two dynamic programming tables of two children nodes X_j, X_k and we want to propagate the dynamic programming table of the parent node X_i. The *indication table* indexes the solution $A_j[c]$ and is indexed by labellings and numbers from 0 to n. We initialize the indication table N_j for X_j by (we likewise initialize N_k):

$$N_j[c_j][x] = \begin{cases} 1 & \text{if} A_j[c_j] = x \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The *convolution table* $\bar{N}_i$ for X_i (and likewise $\bar{N}_j$ and $\bar{N}_k$ for X_j and X_k) is also indexed by labellings and numbers from 0 to n. However, we use a different labeling scheme. To distinguish between the labeling schemes, we use the *bar*-labels $[-r, \ldots, -1, 0, \bar{1}, \ldots, \bar{r}]$ for the convolution table and $\bar{c}$ to represent a *bar*-labeling of the vertices in a bag. We define the convolution table in terms of the indication tables as:

$$\bar{N}_i[\bar{c}][x] = \sum_{c \,:\, \overline{|c(u)|} = \bar{c}(u)} N_i[c][x] \tag{3}$$

We show (in Appendix A) that convolution tables can be computed from indication tables *and vice versa* in time $O(nn_i(2r+1)^{n_i})$ and the convolution table of a join node can be efficiently computed from those of children nodes. This property gives us an efficient way to update the dynamic programming table of a join node.

3 Lower Bound for r-dominating Set

In this section we prove Theorem 3 by giving a reduction from a SAT instance of n_0 variables and m clauses to an instance of r-dominating set in a graph of pathwidth pw such that:

$$\text{pw} \leq \frac{n_0 p}{\lfloor p \log(2r+1) \rfloor} + O(rp) \text{ for any integer } p. \tag{4}$$

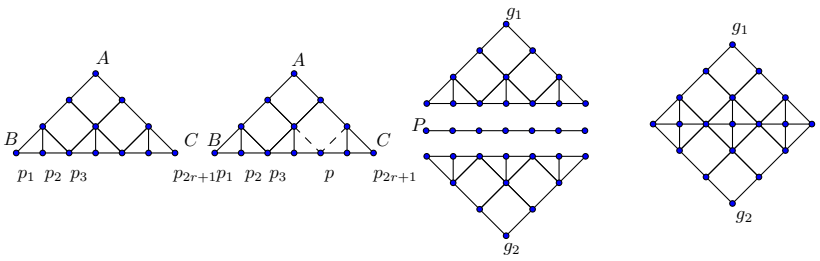


Figure 1 (a) An r-frame (r = 3). (b) An r-frame avoiding p (dashed edges to be deleted). (c) Two r-frames and a path of length $2r + 1$ (r=3). (d) Graph obtained from identifying the two r-frames and path in (c).

Therefore, an $O^*\left((2r+1)^{(1-\epsilon)\text{pw}}\right)$-time algorithm for r-dominating set would imply an algorithm for SAT of time $O^*\left((2r+1)^{(1-\epsilon)\text{pw}}\right)$ which is $O^*\left(2^{(1-\epsilon)\text{pw}\log(2r+1)}\right)$. We argue that for sufficiently large p depending only on ϵ, there is a δ such that:

$$(1-\epsilon)\text{pw}\log(2r+1) = \delta n_0$$

which would complete the reduction. By Equation (4),

$$(1-\epsilon)\text{pw}\log(2r+1) = n_0\left(\frac{(1-\epsilon)p\log(2r+1)}{\lfloor p\log(2r+1)\rfloor} + \frac{O((1-\epsilon)rp)}{n_0\lfloor p\log(2r+1)\rfloor}\right)$$

The second term in the bracketed expression is $o(1)$ for large n_0; we show that the first term in the bracketed expression equal to δ which is less than 1 for sufficiently large p:

$$\frac{(1-\epsilon)p\log(2r+1)}{\lfloor p\log(2r+1)\rfloor} \leq \frac{\lfloor p\log(2r+1)\rfloor - \epsilon\lfloor p\log(2r+1)\rfloor + 1}{\lfloor p\log(2r+1)\rfloor} = 1 - \epsilon + \frac{1}{\lfloor p\log(2r+1)\rfloor}$$

Therefore, choosing p sufficiently large makes this expression smaller than $1 - \epsilon/2$.

Given an integer p, we assume, without loss of generality, that n is a multiple of $\lfloor p\log(2r+1)\rfloor$. Divide the n_0 variables of the SAT formula into t groups, $F_1, ..., F_t$, each of size $\lfloor p\log(2r+1)\rfloor$; $t = \frac{n_0}{\lfloor p\log(2r+1)\rfloor}$. We assume that $r \geq 2$. In the following, the length of a path is given by the number of edges in the path.

r-frame

An *r-frame* is a graph obtained from a grid of size $(r+1)\times(r+1)$, adding edges along the diagonal, removing vertices on one side of the diagonal, subdividing edges of the diagonal path and connecting the subdividing vertices to the vertices of the adjacent triangles. The vertex A is the *top* of the r-frame and the path BC is the *bottom path* of the r-frame. A necessary condition of the r-frame is that the center vertex of the bottom path must r-dominate the whole r-frame. An *r-frame avoiding a vertex p* of the bottom path is the graph obtained by deleting edges not in the bottom path and incident to p. We define *identification* to be the operation of identifying the bottom paths of one or more r-frames with a path of length $2r + 1$ (see Fig. 1).

The group gadget

We construct a gadget to represent each group of variables as follows. Let $\mathcal{P} = \{P_1, P_2, ..., P_p\}$ be a set of p paths, each of length $2r + 1$. For each path P_i, we construct a graph C_i from P_i

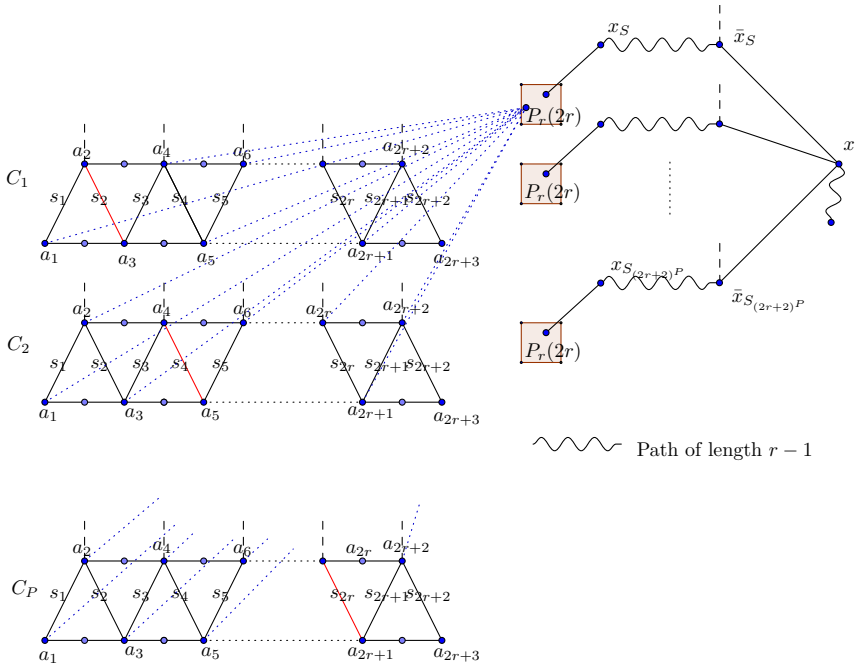


Figure 2 The group gadget. The red edges are edges of r-frames with top x_S.

by identifying two r-frames with top vertices g_1 and g_2, called *guards*, to P_i. In the remaining steps of the construction, we will only connect different parts of the gadget to the vertices of $\mathcal{P}$. In order to r-dominate the guards, we see:

▶ **Observation 7.** *At least one vertex of C_i will be required in the dominating set in order to r-dominate C_i.*

Let S be a set of p vertices, one selected from each path in $\mathcal{P}$. Let $\mathcal{S}$ be the collection of all such sets. We injectively map each set in $\mathcal{S}$ to a particular truth assignment for the corresponding group of variables. Since the number of sets in $\mathcal{S}$ maybe larger than the number of truth assignments, we remove the sets that are not mapped to any truth assignment. For every $S \in \mathcal{S}$, add a vertex x_S, and for each $P \in \mathcal{P}$, identify an r-frame with top x_S avoiding the vertex in $S \cap P$ with P (see Figure 2). Attach each x_S to a distinct vertex $\bar{x}_S$ via a path of length $r-1$. We then connect $\bar{x}_S$ to two new vertices x and x', for all $S \in \mathcal{S}$, and attach paths of length $r-1$ to each of x and x'. Since no vertex in $\mathcal{P}$ can r-dominate, for example, x, we get:

▶ **Observation 8.** *The group gadget requires at least $p+1$ vertices to be r-dominated.*

The super group gadgets

Recall that m is the number of clauses of the SAT instance. For each group F_i, create $m(2rpt+1)$ copies $\{\hat{B}_i^1, ..., \hat{B}_i^{(2rpt+1)m}\}$ of a group gadget. For every $j = 1, \ldots, (2rpt+1)m-1$ and $\ell = 1, \ldots, p$, connect the last vertex of P_ℓ in $\hat{B}_i^j$ to the first vertex of P_ℓ in $\hat{B}_i^{j+1}$. Add two vertices h_1 and h_2, connect h_1 to the first vertices of paths in $\hat{B}_i^1$ and h_2 to the last vertices of paths in $\hat{B}_i^{m(2rpt+1)}$ and attach two paths of length r to each of h_1 and h_2 (see Figure 3).

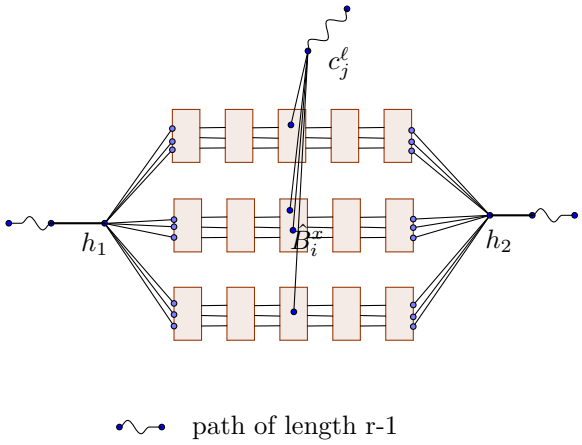


Figure 3 The super group gadgets. Each shaded square represents a group gadget. Each row of group gadgets represents one group of variables. Each column of group gadgets represents all groups.

Connecting the super group gadget to represent a SAT formula

Recall that each set $S \in \mathcal{S}$ for a particular group of variables corresponds to a particular truth assignment for that group of variables. For each clause j, we create $2rpt+1$ *clause* vertices c_j^ℓ for $\ell = 0, \ldots, 2ptr$ and connect each clause vertex to all vertices $\bar{x}_S$ in $\{\hat{B}_i^{m\ell+j} | i = 1, \ldots, t\}$, for all $S \in \mathcal{S}$ that correspond to truth assignments that satisfy the clause j. Connect a path of length $r-1$ to each clause vertex.

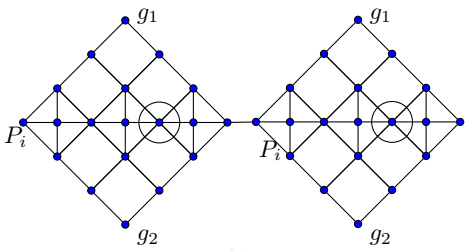
▶ **Lemma 9.** *If ϕ has a satisfying assignment, G has a r-dominating of size $(p+1)tm(2rpt+1)+2$.*

Proof. Given a satisfying assignment of ϕ, we construct the dominating set D of G as follows. For each group gadget $B_i^j, 1 \leq i \leq t, 1 \leq j \leq m(2rpt+1)$, we will select $\{\bar{x}_S\} \cup S$, for S corresponding to the satisfying assignment of the group variables, for the r-dominating set. S r-dominates:

- All the guards and some vertices of their r-frames within distance r from S.
- All the vertices $x_{S'}$ and some vertices of their r-frames within distance r from S for all $S' \in \mathcal{S} \backslash \{S\}$.
- All the vertices of the path P_i in B_i^j and maybe some vertices of its copies in B_i^{j+1} and B_i^{j-1} within distance r from S (see Figure 4)

The remaining vertices of the r-frames of guards and x_S for $S \in \mathcal{S}$ that are not r-dominated by S in B_i^j would be r-dominated by the set S of the nearby group gadgets. The set of vertices that are r-dominated by the vertex $\bar{x}_S$ include:

- The vertices of the path from x_S to $\bar{x}_S$.
- The clause vertex connected to $\bar{x}_S$ and its attached path.
- The vertex x and x' and their attached paths.
- The vertices $\bar{x}_{S'}$ and the vertices of the path from $x_{S'}$ to $\bar{x}_{S'}$ for $S' \in \mathcal{S} \backslash \{S\}$.



(a) a path in $\hat{B}_i^j$ (b) a path in $\hat{B}_i^{j+1}$

Figure 4 Two paths P_i in two consecutive gadgets for $r = 3$. Two circled vertices are in the r-dominating set D. The vertex p_i^1 of the gadget $\hat{B}_i^{j+1}$ is not dominated by the vertex p_i^5 of the same gadget but it is dominated by the vertex p_i^5 of $\hat{B}_i^j$. The distance between two circled vertices must be no larger than 7 $(= 2r + 1)$. The numbering of the vertices of the horizontal path is shown in Figure 1.

Taking the union over all t groups, and all $m(2rpt + 1)$ copies of the group gadgets in the super group gadgets gives $(p + 1)tm(2rpt + 1)$ vertices. Adding vertices h_1 and h_2 gives the lemma. ◀

▶ **Lemma 10.** *If G has a r-dominating set of size $(p + 1)tm(2ptr + 1) + 2$, then ϕ has a satisfying assignment.*

Proof. Let D be the r-dominating set of size $(p + 1)tm(2ptr + 1) + 2$. Since some vertices in the paths attached to h_1 and h_2 must be in D, we can replace these with h_1 and h_2. By observation 8, at least $(p + 1)$ vertices of each group gadget must be in D, which implies that exactly $(p + 1)$ vertices are chosen from each group gadget since there are $tm(2ptr + 1)$ group gadgets. Let $\hat{B}_i^j, 1 \le i \le t, 1 \le j \le m(2ptr + 1)$ be a group gadget and let $P_k \in \mathcal{P} = \{P_1, \ldots, P_p\}$ be a path of $\hat{B}_i^j$. By observation 7, at least one vertex from each $P_k, 1 \le k \le p$ must be included in D. To dominate the vertex x and x' and their attached paths, at least one vertex from the set $\{\bar{x}_S | S \in \mathcal{S}\}$ must be selected. Therefore, the set of $p + 1$ vertices in $D \cap \hat{B}_i^j$ includes:

- p vertices, one from each path $P_k, 1 \le k \le p$, which make up the set S.
- the vertex $\bar{x}_S$ that corresponds to x_S since x_S is not dominated by S.

We say that the dominating set D is *consistent* with a set of gadgets $\{\hat{B}_i\}_{i=1}^k$ iff $D \cap \mathcal{P}$ is the same for all $\hat{B}_i$. We show that there exists a number $\ell \in \{0, 1, \ldots, 2rtp\}$ such that D is consistent with the set of gadgets $\{\hat{B}_i^{m\ell+j} | 1 \le j \le m\}$ for each $1 \le i \le t$. For two consecutive gadgets $\hat{B}_i^q$ and $\hat{B}_i^{q+1}$, if two vertices p_i^a and p_i^b of the path P_i in $\hat{B}_i^q$ and of its copy in $\hat{B}_i^{q+1}$, respectively, are selected, the distance between them must be less than $2r + 1$ (see Figure 4). Therefore, we have $b \le a$. We call two consecutive gadgets $\hat{B}_i^q$ and $\hat{B}_i^{q+1}$ a *bad* pair if $b < a$. Since the distance between p_i^a and p_i^b is smaller than $2r+1$, there are at most $2pr$ consecutive bad pairs for each i and for t groups of variables $F_i, 1 \le i \le t$, the number of bad pairs is no larger than $2rpt$. By the pigeonhole principle, there exists a number $\ell \in \{0, 1, \ldots, 2rtp\}$ such that D is consistent with the set of gadgets $\hat{B}_i^{m\ell+j}, 1 \le j \le m$ for all i.

For each $i \in \{1, \ldots, t\}$, let $\{\hat{B}_i^{m\ell+j} | 1 \le j \le m\}$ for some $\ell \in \{0, 1, \ldots, 2prt\}$ be the set of group gadgets that is consistent with D and let F_i be the corresponding group of variables. We assign to the variables of group F_i the values of assignment corresponding to the selected set S. This assignment satisfies the clauses of ϕ that are connected to the vertices $\bar{x}_S$. Because all clauses of ϕ are r-dominated, the truth assignment of all groups $F_i, 1 \le i \le t$, makes up a satisfying assignment of ϕ. ◀

We prove the following bound on pathwidth using a *mixed search game* [17]. We view the graph G as a system of tunnels. Initially, all edges are contaminated by a gas. An edge can be *cleared* by placing two searchers at both ends of that edge simultaneously or by sliding a searcher along that edge. A cleared edge can be recontaminated if there is a path between this edge and a contaminated edge such that there is no searcher on this path. Set of rules for this game includes:

- Placing a searcher on a vertex.
- Removing a searcher from a vertex.
- Sliding a searcher on a vertex along an incident edge.

A search is a sequence of moves following these rules. A search strategy is *winning* if all edges of G are cleared after its termination. The minimum number of searchers required to win is the mixed search number of G, denoted by $\mathbf{ms}(G)$. The following relation is established in [17]:

$$\mathrm{pw}(G) \leq \mathbf{ms}(G) \leq \mathrm{pw}(G) + 1\,.$$

▶ **Lemma 11.** $\mathrm{pw}(G) \leq tp + O(rp)$.

Proof. We give a search strategy using at most $tp + O((2r+1)p)$ searchers. For a group gadget $\hat{B}$, we call the sets of vertices $\{P_i^1 | 1 \leq i \leq p\}$ and $\{P_i^{2r+1} | 1 \leq i \leq p\}$ the sets of *entry vertices* and *exit vertices*, respectively. We search the graph G in $m(2rpt+1)$ rounds. Initially, we place tp searchers on the entry vertices of t group gadgets $\hat{B}_i^1, 1 \leq i \leq t$. We use one more searcher to clear the path and the edges incident to h_1. In round $b, 1 \leq b \leq m(2prt+1)$ such that $b = m\ell + j, 0 \leq \ell \leq 2prt+1, 1 \leq j \leq m$, we keep tp searchers on the entry vertices of all group gadgets $\hat{B}_i^{ml+j}, 1 \leq i \leq t$. We clear the group gadget $\hat{B}_i^{ml+j}$ by using at most $5(2r+1)p+4$ searchers in which:

- $(2r+1)p$ searchers are placed on the vertices of p paths in $\mathcal{P}$.
- $2(2r+1)$ searchers to clear the guards and their r-frames.
- 3 searchers are placed on x, x' and c_j^ℓ and one more searcher to clear their attached paths.
- $2(2r+1)$ to clear x_S and their r-frames for all $S \in \mathcal{S}$.

After $\hat{B}_i^{ml+j}$ is cleared, we keep searchers on the exit vertices and c_j^ℓ, remove other searchers and reuse them to clear $\hat{B}_{i+1}^{ml+j}$. After all the group gadgets in round b are cleared, we slide searchers on the exit vertices of $\hat{B}_i^b$ to the entry vertices of $\hat{B}_i^{b+1}$ for all $1 \leq i \leq t$ and start a new round. When $b = m(2rpt+1)$, we need one more searcher to clear the path and the edges incident to h_2. In total, we use at most $tp + (5+p)(2r+1) + 4$ searchers which completes the proof of the lemma. ◀

Combined with Lemmas 9 and 10, we get Theorem 3.

4 Algorithm for Connected r-dominating Set

We apply the Cut&Count technique by Cygan et al. [6] to design a randomized algorithm which decides whether there is a connected r-dominating set of a given size in graphs of treewidth at most tw in time $O((2r+2)^{\mathrm{tw}} n^{O(1)})$ with probability of false negative at most $\frac{1}{2}$ and no false positives.

Rather than doubly introduce notation, we give an overview of the Cut&Count technique as applied to our connected r-dominating set problem. The goal is, rather than search over the set of all possible *connected* r-dominating sets, which usually results in $\Omega(\mathrm{tw}^{\mathrm{tw}})$ configurations for the dynamic programming table, to search over all possible r-dominating sets. Formally, let $\mathcal{S}$ be the family of connected subsets of vertices that r-dominate the input

graph and let $\mathcal{S}_k \subseteq \mathcal{S}$ be the subset of solutions of size k. Likewise, let $\mathcal{R}$ be the family of (not-necessarily-connected) subsets of vertices that r-dominate the input graph and similarly define $\mathcal{R}_k$. Note that $\mathcal{S}$ and $\mathcal{S}_k$ are subsets of $\mathcal{R}$ and $\mathcal{R}_k$, respectively. We wish to determine, for a given k, whether $\mathcal{S}_k$ is empty. We cannot, of course, simply determine whether $\mathcal{R}_k$ is empty. Instead, for every subset of vertices U, we derive a family $\mathcal{C}(U)$ whose size is odd only if $G[U]$ is connected. Further, we assign random weights ω to the vertices of the graph, so that, by the Isolation Lemma (formalized below), the subset of $\mathcal{S}_k$ contains a unique solution of minimum weight with high probability. We can then determine, for a given k, the parity of $|\cup_{U\in\mathcal{R}_k\,:\,\omega(U)=W}\mathcal{C}(U)|$ for every W. We will find at least one value of W to result in odd parity if $\mathcal{S}_k$ is non-empty.

The *Isolation Lemma* was first introduced by Valiant and Vazirani [18]. Given a universe $\mathbb{U}$ of $|\mathbb{U}|$ elements and a weight function $\omega : \mathbb{U} \to \mathbb{Z}$. For each subset $X \subseteq \mathbb{U}$, we define $\omega(X) = \sum_{x\in X}\omega(x)$. Let $\mathcal{F}$ be a family of subsets of $\mathbb{U}$. We say that ω *isolates* a family $\mathcal{F}$ if there is a *unique* set in $\mathcal{F}$ that has minimum weight.

▶ **Lemma 12** (Isolation Lemma). *For a set $\mathbb{U}$, a random weight function $\omega : \mathbb{U} \to \{1, 2, \ldots, N\}$, and a family $\mathcal{F}$ of subsets of $\mathbb{U}$:*

$$\mathit{Prob}[\omega \text{ isolates } \mathcal{F}] \geq 1 - \frac{|\mathbb{U}|}{N}\,.$$

Throughout the following, we fix a *root* vertex, ρ, of the graph $G = (V, E)$ and use a random assignment of weights to the vertices $\omega : V \to \{1, 2, \ldots, 2n\}$.

4.1 Cutting

Given a graph $G = (V, E)$, we say that an ordered bipartition (V_1, V_2) of V is a *consistent cut* of G if there are no edges in G between V_1 and V_2 and $\rho \in V_1$. We say that an ordered bipartition (C_1, C_2) of a subset C of V is a *consistent subcut* if there are no edges in G between C_1 and C_2, and, if $\rho \in C$ then $\rho \in C_1$.

▶ **Lemma 13** (Lemma 3.3 [6]). *Let C be a subset of vertices that contains ρ. The number of consistent cuts of $G[C]$ is $2^{cc(G[C])-1}$ where $cc(G[C])$ is the number of connected components of $G[C]$.*

Recall the definitions of $\mathcal{S}$, $\mathcal{S}_k$, $\mathcal{R}$ and $\mathcal{R}_k$ from above. We further let $\mathcal{S}_{k,W}$ be the subset of $\mathcal{S}_k$ with the further restriction of having weight W: $\mathcal{S}_{k,W} = \{U \in \mathcal{S}_k \;:\; \omega(U) = W\}$. Similarly, we define $\mathcal{R}_{k,W}$. Let $\mathcal{C}_{k,W}$ be the family of consistent cuts derived from $\mathcal{R}_{k,W}$ as:

$$\mathcal{C}_{k,W} = \{(C_1, C_2)) \;:\; C \in \mathcal{R}_{k,W} \text{ and } (C_1, C_2) \text{ is a consistent cut of } G[C]\}\,.$$

Since the number of consistent cuts of $G[C]$ for $C \in \mathcal{S}_{k,W}$ is odd by Lemma 13 and the number of of consistent cuts of $G[C]$ for $C \in \mathcal{R}_{k,W} \setminus \mathcal{S}_{k,W}$ is even by Lemma 13, we get:

▶ **Lemma 14** (Lemma 3.4 [6]). *For every W, $|\mathcal{S}_{k,W}| \equiv |\mathcal{C}_{k,W}| \pmod 2$.*

4.2 Counting

Lemma 14 allows us to focus on computing $|\mathcal{C}_{k,W}| \pmod 2$. In Appendix B, we give an algorithm to compute $|\mathcal{C}_{k,W}|$ for all k and W ($W \in \{1, 2, \ldots, 2n^2\}$):

▶ **Lemma 15.** *There is an algorithm which computes $|\mathcal{C}_{k,W}|$ for all k and W in time $O(k^2n^4(2r + 2)^{\mathrm{tw}})$.*

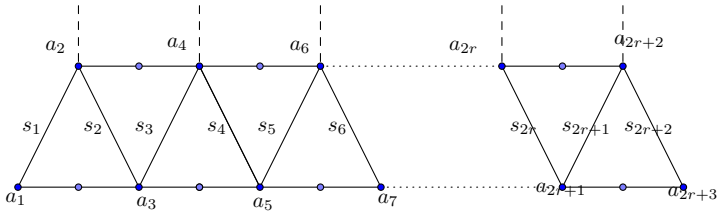


Figure 5 A core for group gadgets. The dashed lines represent paths of length $r+1$ connected to r_T.

Let k^* be the size of the smallest connected r-dominating set. Since the range of ω has size $2n$, by the Isolation Lemma, the smallest value W^* of W such that $\mathcal{S}_{k^*,W}$ is non-empty also implies that $|\mathcal{S}_{k^*,W^*}| = 1$ with probability $1/2$. By Lemma 14, $|\mathcal{C}_{k^*,W^*}|$ is also odd (with probability $1/2$). We can then find $|\mathcal{C}_{k^*,W^*}|$ by linear search over the possible values of W. Thus Lemma 15 implies Theorem 4

5 Lower Bound for Connected r-dominating Set

In this section, we prove Theorem 5. The main idea is similar to that of the previous section: a reduction from n_0-variable, m-clause SAT to an instance of connected r-dominating set in a graph of pathwidth pw such that:

$$\text{pw} \leq \frac{n_0 p}{\lfloor p \log(2r+2) \rfloor} + O\left((2r+2)^{2p}\right) \text{ for any integer } p.$$

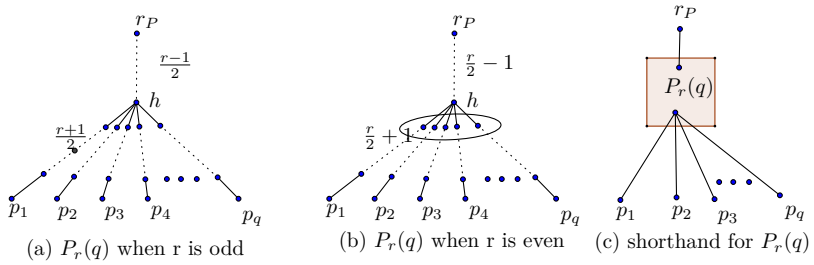
Given this reduction, the final argument for Theorem 5 is similar to the argument at the beginning of Section 3. Let ϕ be a SAT formula with n_0 variables and m clauses. For a given integer p, we assume that n_0 is divisible by $\lfloor p \log(2r+2) \rfloor$. We partition ϕ's variables into $t = \frac{n}{\lfloor p \log(2r+2) \rfloor}$ groups of variables $\{F_1, F_2, \ldots, F_t\}$ each of size $\lfloor p \log(2r+2) \rfloor$. We will speak of an r-dominating *tree* as opposed to a connected r-dominating set: the tree is simply a witness to the connectedness of the r-dominating set. We treat the problem as rooted: our construction has a global root vertex, r_T, which we will require to be in the rCDS solution. This can be forced by attaching a path of length r to r_T.

Core

A core is composed of a path with $2r+3$ vertices $a_1, a_2, \ldots, a_{2r+3}$, $2r+2$ edges $s_1, s_2, \ldots, s_{2r+2}$ (called *segments*), consecutive odd-indexed vertices connected by a subdivided edge and consecutive even-indexed vertices connected by a subdivided edge. The even indexed vertices $a_2, a_4, \ldots, a_{2r+2}$ are connected to the root r_T via paths of length $r+1$ (see Figure 5).

Pattern

A *pattern* $P_r(q)$ is a tree-like graph with q leaves and a single root r_P such that the distance from the root to the leaves is r. The structure depends on the parity of r; if r is even, the children of vertex h are connected by a clique (indicated by the oval). The dotted lines represent paths of length $\frac{r-1}{2}$ for r odd and $\frac{r}{2} - 1$ for r even (see Figure 6).



(a) $P_r(q)$ when r is odd (b) $P_r(q)$ when r is even (c) shorthand for $P_r(q)$

Figure 6 A pattern $P_r(q)$ with root r_P. The oval in (b) is a clique between neighboring vertices of the vertex h.

▶ **Observation 16.** *A leaf of a pattern r-dominates all but the other leaves of the pattern.*

Given a set of q vertices X, we say that pattern $P_r(q)$ is attached to set X if the leaves of $P_r(q)$ are identified with X.

Core gadget

We connect patterns to the core in such a way as to force a minimum solution to contain a path from r_T to the core, ending with a segment edge. To each core that we use in the construction, we attach one pattern $P_r(r+1)$ to the odd-indexed vertices $a_1, a_2, \ldots, a_{2r+1}$ (but not a_{2r+3}) and another pattern $P_r(r+1)$ to the even-indexed vertices $a_2, a_4, \ldots, a_{2r+2}$. In order to r-dominate the roots of these patterns, the dominating tree must contain a path from r_T to an odd-indexed vertex and to an even-indexed vertex. We attach additional *path-forcing* patterns to guarantee that, even after adding the rest of the construction, this path will stay in the dominating tree. For $i = 1, \ldots, r$, for the $r+1$ vertices that are i hops from r_T, we attach a pattern $P_r(r+1)$. As a result, at least one vertex at each distance from r_T must be included in the dominating tree. A core gadget is a subgraph of the larger construction such that edges from the remaining construction only attach to the vertices $a_1, a_2, \ldots, a_{2r+3}$ and r_T. The previous observations guarantee:

▶ **Observation 17.** *The part of a rCDS that intersects a core gadget can be modified to contain a path from r_T to an odd-indexed vertex (via a segment edge) without increasing its size.*

Group gadget

For each group F_i of variables, we construct a *group gadget* which consists of p cores $\{C_1, C_2, \ldots, C_p\}$. Let S be a set of p segments, one from each core, and let $\mathcal{S}$ be a collection of all possible such sets S; therefore $|\mathcal{S}| = (2r+2)^p$. Since a group represents $\lfloor p \log(2r+2) \rfloor$ variables, there are at most $2^{p \log(2r+2)} = (2r+2)^p$ truth assignments to each group of variables. We injectively map each set in $\mathcal{S}$ to a particular truth assignment for the corresponding group of variables. Since the number of sets in $\mathcal{S}$ maybe larger than the number of truth assignments, we remove the sets that are not mapped to any truth assignment. For each set $S \in \mathcal{S}$, we connect a corresponding *set* pattern $P_r(2rp)$ to the cores as follows. For each $i = 1, \ldots, p$, $P_r(2rp)$ is attached to

- the vertices $a_1, a_2, \ldots, a_{2r+2}$ of C_i *except* the endpoints of s_j if $s_j \in S$
- the vertices $a_2, a_3, \ldots, a_{2r+1}$ if $s_{2r+2} \in S$

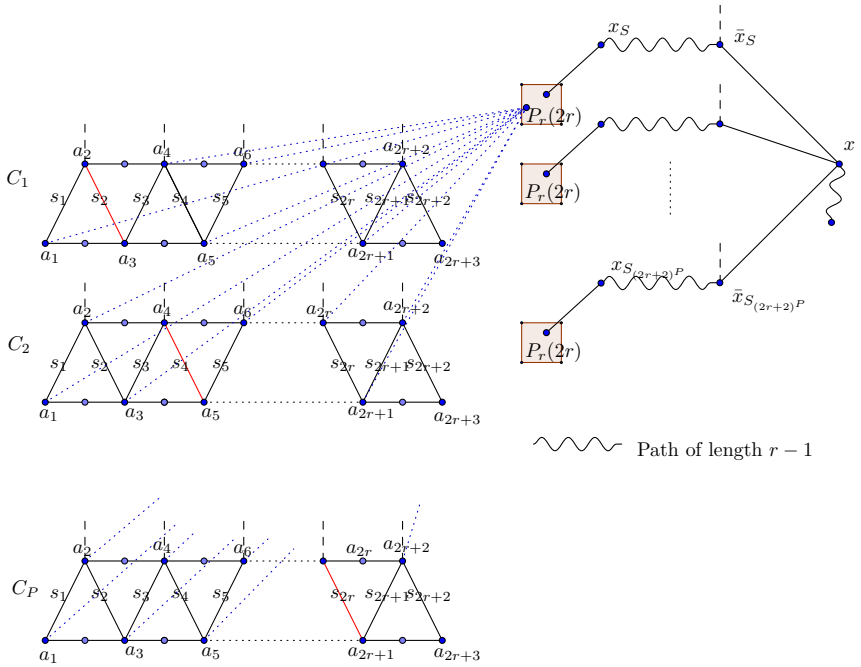


Figure 7 A group gadget. Red segments are segments of the set S. The vertex x_S is the root of the pattern shown. The dashed lines represent paths of length $r+1$ connected to r_T.

We label the root of this pattern by the set vertex x_S. We then connect these set patterns together. For each $S \in \mathcal{S}$, we connect:

- x_S to a new vertex $\bar{x}_S$ via a path of length $r-1$
- $\bar{x}_S$ to the root r_T via paths of length $r+1$
- $\bar{x}_S$ to a common vertex x, and
- x to a path of length $r-1$.

Similarly, as in the core gadget construction, for the set of vertices $\{\bar{x}_S | S \in \mathcal{S}\}$, we add *path forcing* patterns $P_r(|\mathcal{S}|)$ to each level of vertices along the paths from r to $\bar{x}_S, S \in \mathcal{S}$ (see Figure 7).

▶ **Observation 18.** *The part of a rCDS that intersects a group gadget can be modified to contain a path from r_T to a vertex in the set $\{\bar{x}_S | S \in \mathcal{S}\}$ without increasing its size.*

Super-path

A *super path* F_i is a graph that consists of $X = m\,((2r+1)pt+1)$ copies of the group gadget $B_i^1, B_i^2, \ldots, B_i^X$, which are assembled into a line (m is the number of clauses). Vertex a_{2r+s} of every core gadget of the group gadget B_i^j is identified with the vertex a_1 of the corresponding core gadget of the group gadget B_i^{j+1}. The vertices a_1 and a_{2r+3} of the core gadgets of B_i^1 and B_i^X are directedly connected to the root r_T. In order to dominate all the odd- and even-indexed vertices of the cores (without spanning more than one segment edge per core), we must have:

▶ **Observation 19.** *If an endpoint of segment edge s_j in the t^{th} core of B_i^k is in the rCDS, then there must be an endpoint of a segment $s_{j'}$ in the t^{th} core of B_i^{k+1} that is also in the rCDS, for $j' \leq j$.*

Representing clauses

For each clause C_j of ϕ, we introduce $((2r+1)pt+1)$ clause vertices c_j^ℓ, $0 \leq \ell \leq (2r+1)pt$. (There are $m((2r+1)pt+1)$ clause vertices in total.) For a fixed i $(1 \leq i \leq t)$ and for each c_j^ℓ, $(1 \leq j \leq m, 0 \leq \ell \leq (2r+1)pt)$, we connect c_j^ℓ to $B_i^{m\ell+j}$ by connecting it directly to the subset of vertices in the set $\{\bar{x}_S | S \in \mathcal{S}\}$ of $B_i^{m\ell+j}$ such that the truth assignments of the corresponding subsets in the collection $\mathcal{S}$ satisfy the clause C_j. Each clause vertex is attached to a path of length $r-1$. Denote the final constructed graph as G.

▶ **Lemma 20.** *If ϕ has a satisfying assignment, G has a connected r-dominating set of $((r+2)p+r+1)tm((2r+1)tp+1)+1$ vertices.*

Proof. Given a satisfying assignment of ϕ, we construct an r-dominating tree T as follows. For group i, let S_i be the set of p segments which corresponds to the truth assignment of variables of F_i. In addition to the root, T contains:

- The path of length $r+2$ from r_T that ends in each segment of S_i for every group in the construction. Each such path contains $r+2$ vertices in addition to the root. As there are $tm((2r+1)tp+1)$ groups and p cores per group, this takes $(r+2)ptm((2r+1)tp+1)$ vertices. By Observation 16, this set of vertices will r-dominate all of the non-leaf vertices of all the patterns in the core gadget, since all these patterns include a leaf in one of these paths. This set of vertices will also dominate x_S for every $S \neq S_i$ since S will connect to the endpoints of at least one segment edge that is not in S_i.
- For each group, the path of length $r+1$ from r_T to $\bar{x}_{S_i}$. Each such path contains $r+1$ vertices (not including the root). As there are $tm((2r+1)tp+1)$ groups, this takes $(r+1)tm((2r+1)tp+1)$ vertices (not including the root). The vertex $\bar{x}_{S_i}$ r-dominates the vertices on the path from x_{S_i} to $\bar{x}_{S_i}$, the vertex x and the path attached to it, the clause vertex connected to $\bar{x}_{S_i}$ and its attached path and the vertices on the path from x_S to $\bar{x}_S$, *not including* x_S, for every $S \neq S_i$. ◀

▶ **Lemma 21.** *If G has an r-dominating tree of $((r+2)p+r+1)tm((2r+1)tp+1)+1$ vertices, then ϕ has a satisfying assignment.*

Proof. Let T be the r-dominating tree; T contains the root r_T. By Observations 17 and 18, each group gadget requires at least $(r+2)p+r+1$ vertices (not including the root) in the dominating tree. Since the number of copies of group gadget is $tm((2r+1)tp+1)$, this implies exactly $(r+2)p+r+1$ vertices of each group gadget will be selected in which:

- For each core gadget, exactly one segment s_i in the set $\{s_1, s_2, \ldots, s_{2r+2}\}$ and a path connecting it to the root r_T are selected which totals $(r+2)p$ vertices for p cores. Denote the set of p selected segments by S.
- $r+1$ vertices on the path from $\bar{x}_S$ to the root r_T.

We say that T is consistent with a set of group gadgets iff the set of segments in T are the same for every group gadget. If two segments s_a and s_b of two consecutive cores in group gadgets B_i^q and B_i^{q+1}, respectively, are in T, by Observation 19, we have $b \leq a$. If $b < a$, we call s_a and s_b a *bad pair*. Since there are p cores in which there can be a bad pair, and each core has $2r+2$ segments, for each super-path, there can be at most $(2r+1)p$ consecutive bad pairs. Since we have t super-paths, there are at most $tp(2r+1)$ bad pairs. By the pigeonhole

principle, there exists a number $\ell \in \{0, 1, \ldots, tp(2r+1)\}$ such that T is consistent with the set of gadgets $\{B_i^{m\ell+j} | 1 \leq i \leq t, 1 \leq j \leq m\}$.

Let $\{B_i^{m\ell+j} | 1 \leq i \leq t, 1 \leq j \leq m\}$ be the set of group gadgets which is consistent with T. For each group gadget $B_i^{m\ell+j}$, we assign the truth assignment corresponding to the set of segments $S \in T \cap B_i^{m\ell+j}$ to variables in the group F_i. The assignment of variables in all F_i makes up a satisfying assignment of ϕ, since all clause vertices are r-dominated by T. ◀

▶ **Lemma 22.** $\mathrm{pw}(G) \leq tp + O((2r+2)^{2p})$.

By slightly adapting the proof of Lemma 11, we can prove Lemma 22 and we leave the details as an exercise to readers.

Acknowledgement. We thank the anonymous reviewers for helpful comments and for pointing out the mistake in the earlier version of the proof of Theorem 3.

References

1. B. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994. doi:10.1145/174644.174650.
2. H. L. Bodlaender, M. Cygan, S. Kratsch, and J. Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming*, ICALP'13, pages 196–207, 2013. doi:10.1007/978-3-642-39206-1_17.
3. H. L. Bodlaender and B. A. Fluiter. Reduction algorithms for graphs of small treewidth. *Information and Computation*, 167(2):86–119, 2001. doi:10.1006/inco.2000.2958.
4. G. Borradaile, E. Demaine, and S. Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Algorithmica*, 68(2):287–311, 2014. doi:10.1007/s00453-012-9662-2.
5. G. Borradaile, P. Klein, and C. Mathieu. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Transactions on Algorithms*, 5(3):31:1–31:31, 2009. doi:10.1145/1541885.1541892.
6. M. Cygan, J. Nederlof, M. Pilipczuk, J. M. M. van Rooij M. Pilipczuk, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS'11, pages 150–159, 2011. doi:10.1109/FOCS.2011.23.
7. E. Demaine, M. Hajiaghayi, and B. Mohar. Approximation algorithms via contraction decomposition. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'07, pages 278–287, 2007.
8. E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for the (k, r)-center in planar graphs and map graphs. In *Proceedings of the 30th International Conference on Automata, Languages and Programming*, ICALP'03, pages 829–844, 2003. doi:10.1007/3-540-45061-0_65.
9. D. Eisenstat, P. N. Klein, and C. Mathieu. Approximating k-center in planar graphs. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'14, pages 617–627, 2014.
10. D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000. doi:10.1007/s004530010020.
11. F. V. Fomin, D. Lokshtanov, and S. Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'14, pages 142–151, 2014.

12 R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

13 P. N. Klein. A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. *SIAM Journal on Computing*, 37(6):1926–1952, 2008. doi:10.1137/060649562.

14 Ton Kloks, editor. *Treewidth, Computations and Approximations*, volume 842. Springer Berlin Heidelberg, 1994. doi:10.1007/BFb0045375.

15 D. Lokshtanov, D. Marx, and S. Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'11, pages 777–789, 2011.

16 N. Robertson and P. 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.

17 A. Takahashi, S. Ueno, and Y. Kajitani. Mixed searching and proper-path-width. *Theoretical Computer Science*, 137(2):253–268, 1995. doi:10.1007/3-540-54945-5_50.

18 L. G. Valiant and V. V. Vazirani. **NP** is as easy as detecting unique solutions. *Theoretical Computer Science*, 47(0):85–93, 1986. doi:10.1016/0304-3975(86)90135-0.

19 J. M. M. van Rooij, H. L. Bodlaender, and P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Proceedings of 17th European Symposium on Algorithms*, ESA'09, pages 566–577, 2009. doi:10.1007/978-3-642-04128-0_51.

A Details of the Dynamic Programming Algorithm for r-dominating Set

▶ **Definition 23** (Nice tree decomposition)**.** A tree decomposition T of G is *nice* if the following conditions hold:

- T is rooted at node X_0.
- Every node has at most two children.
- Any node X_i of T is one of four following types:
  leaf node: X_i is a leaf of T,
  forget node: X_i has only one child X_j and $X_i = X_j \backslash \{v\}$,
  introduce node: X_i has only one child X_j and $X_j = X_i \backslash \{v\}$,
  join node: X_i has two children X_j, X_k and $X_i = X_j = X_k$.

We will show how to handle each of the four types of nodes (leaf, forget, introduce and join) in turn. We use $\#_0(X_i, c)$ to denote the number of vertices in X_i that are assigned label 0 in c in populating the tables for leaf and join nodes. We say v *positively resolves* u if $c(v) = c(u) - 1$ when $c(u) > 0$; we use this definition for leaf and introduce nodes.

Leaf Node

We populate the table A_i for a leaf node X_i as follows:

$$A_i[c] = \begin{cases} 0 & \text{if } c \text{ is locally valid and all negative} \\ \#_0(X_i, c) & \text{if } c \text{ is locally valid and all positive are positively resolved} \\ \infty & \text{otherwise} \end{cases} \tag{5}$$

We can populate the table for a leaf node in $O(n_i^2(2r+1)^{n_i})$ time. Also, we can prove the correctness invariant is maintained at leaf nodes by induction on the label of vertices.

Forget Node

Let X_i be a forget node with child X_j and $X_i = X_j \cup \{u\}$. Let c_i be a labeling of X_i. We consider extensions of c_i to labelings $c_j = c_i \times d$ of X_j as follows:

$$c_j(v) = \begin{cases} c_i(v) & \text{if } v \in X_i \\ d & \text{otherwise} \end{cases}.$$

We populate the table A_i for forget node X_i as follows:

$$A_i[c_i] = \min \begin{cases} A_j[c_i \times d] & \forall d < 0, \exists \text{ a } v \text{ in } X_i \text{ s.t. } c_i(v) = d + d_G(u,v) \\ A_j[c_i \times d] & \forall d \geq 0 \end{cases}. \tag{6}$$

Assuming the correctness invariant for A_j, we can show that the correctness invariant is maintained for forget nodes. For running time, we can check the condition for the first case of Equation (6) in time proportional to the degree of u in X_j ($O(n_i)$). Thus, we can populate the table for a forget node X_i with child X_j in $O(n_i(2r+1)^{n_j})$ time.

Introduce Node

Let X_i be an introduce node with its child X_j and $X_i = X_j \cup \{u\}$. We show how to compute $A_i[c_i]$ where $c_i = c_j \times d$ is the extension of a labeling c_j for X_j to X_i where u is labeled d. We define a map ϕ applied to the label $c_j(v)$ of a vertex v:

$$\phi(c_j(v)) = \begin{cases} -c_j(v) & \text{if } c_j(v) > 0 \text{ and } d_{G[V_i]}(v,u) = d_G(u,v) = c_j(v) - d \\ c_j(v) & \text{otherwise} \end{cases}.$$

We use $\phi(c_j)$ to define the natural extension this map to a full labeling of X_j. Clearly $\phi(c_j) \preceq c_j$. This map corresponds to the lowest ordering label that is $\preceq c_j$ that we use in conjunction with the Ordering Invariant. Note that ϕ will be used only for $d \geq 0$. There are three cases for computing $A_i[c_j \times d]$, depending on the value of d:

$d = 0$: In this case, u is in the dominating set. If a vertex $v \in X_j$ is to be r-dominated by u via a path contained in $G[V_i]$, it will be represented by the table entry in which $c_j(v) = -d_G(u,v)$. Therefore $A_j[c'_j] + 1$ corresponds to the size of a subset of V_i that induces the positive labels of $c_j \times 0$ for any $c'_j \preceq c_j$ and where $c_j(v) = d_G(u,v) = d_{G[V_i]}(v,u)$. The Ordering Invariant tells us that the best solution is given by the rule $A_i[c_j \times 0] = A_i[\phi(c_j) \times 0] = A_j[\phi(c_j)] + 1$.

$d > 0$: In this case, u is r-dominated is to be dominated by a vertex in V_i via a path contained by $G[V_i]$. Therefore, we require there be a neighbor v of u in X_j (with a label) that positively resolves u; otherwise, the labeling $c_j \times d$ is infeasible. Further, for other vertices v' of X_j which are r-dominated by a vertex of V_i by a path through u and contained by $G[V_i]$, the condition of the mapping ϕ must hold $d_G(v',u) = d_{G[V_i]}(v',u) = c_j(v') - d$. As for the previous case, the Ordering Invariant tells us that the best solution is given by the rule: $A_i[c \times \{t\}] = A_j[\phi(c)]$ if $v \in X_i$ s.t v positively resolves u and $A_i[c \times \{t\}] = +\infty$ otherwise.

$d < 0$: In this case, u is not r-dominated by a path contained entirely in $G[V_i]$. Therefore, the table entries for X_i are simply inherited from X_j (as long as $c_j \times d$ is locally valid). We set $A_i[c \times \{c_i(u)\}] = A_j[c]$ if $c \times \{c_i(u)\}$ is a locally valid labeling of X_i and $A_i[c \times \{c_i(u)\}] = +\infty$ otherwise.

Since the correctness invariant holds for X_j, and by the arguments above (using the Ordering Lemma), the correctness invariant is maintained for introduce nodes. Also, we can populate the dynamic programming table for an introduce bag in $O((2r+1)^{\mathrm{tw}}\mathrm{tw})$ time.

Join Nodes

In this section, we will refer to the tables of the previous sections as the original tables. We denote the *indication table* by N and the *convolution table* by $\bar{N}$. We will initialize N_j and N_k from A_j and A_k, then compute $\bar{N}_j$ from N_j and $\bar{N}_k$ from N_k, then combine $\bar{N}_j$ and $\bar{N}_k$ to give $\bar{N}_i$, then compute N_i from $\bar{N}_i$ and finally A_i from N_i. The tables $\bar{N}_j$ can be used to count the number of r-dominating sets; we view our method as incorrectly counting so that we can more efficiently compute A_i from A_j and A_k while still correctly computing A_i.

Let X_i be a join node with two children X_j and X_k and $X_i = X_j = X_k$. We say the labeling c_i (for X_i) is *consistent* with labelings c_j and c_k (for X_j and X_k, respectively) if for every $u \in X_i$:

1. If $c_i(u) = 0$, then $c_j(u) = 0$ and $c_k(u) = 0$.
2. If $c_i(u) = t < 0$, then $c_j(u) = t$ and $c_k(u) = t$.
3. If $c_i(u) = t > 0$, then $(c_j(u) = t) \wedge (c_k(u) = -t)$ or $(c_j(u) = -t) \wedge (c_k(u) = t)$ or $(c_j(u) = t) \wedge (c_k(u) = t)$.

$$A_i[c_i] = min(A_j[c_j] + A_k[c_k] - \#_0(X_i, c_i) | c_i \text{ is consistent with } c_j \text{ and } c_k) \tag{7}$$

Using the indication table, Equation 7 can be written as:

$$A_i[c_i] = \min\{\infty, \min\{x \ : \ N_i[c_i][x] > 0\}\} \tag{8}$$

where we define

$$N_i[c_i][x] = \sum_{\substack{x_j, x_k : x_j + x_k - \#_0(X_i, c_i) = x \\ c_i \text{ is consistent with } c_j \text{ and } c_k}} N_j[c_j][x_j] \cdot N_k[c_k][x_k] \,. \tag{9}$$

We guarantee that $N_i[c_i][x]$ is non-zero only if there is a subset of V_i of size x that induces the positive labels of c_i. This, along with the correctness invariant held for the children of X_i, we maintain the correctness invariant at join nodes.

The following observation, which is a corollary of Equation (7) and the definition of *consistent*, is the key to our algorithm:

▶ **Observation 24.** *If the vertex $u \in X_i$ has label $\bar{t}$, its label in X_j and X_k must also $\bar{t}$.*

Running time analysis

▶ **Lemma 25.** *Convolution tables can be computed from indication tables* and vice versa *in time $O(nn_i(2r+1)^{n_i})$.*

Proof. Consider the indicator table N_i and convolution table $\bar{N}_i$ for bag X_i; we order the vertices of X_i arbitrarily. We calculate Equation (3) by dynamic programming over the vertices in this order.

We first describe how to compute $\bar{N}_i$ from N_i. We initialize $\bar{N}_i[c] = N_i[c']$ where $c'(u) = c(u)$ if $c(u) \leq 0$ and $\overline{c'(u)} = c(u)$ if $c'(u) > 0$. We then correct the table by considering the barred labels of the vertices according to their order from left to right. In particular, suppose $c = c_1 \times \{\bar{t}\} \times c_2$, for some $t > 0$, be a labeling in which c_1 is a bar-labeling of the first ℓ vertices of X_i and c_2 is a bar-labeling of the last $n_i - \ell - 1$ vertices. We update $\bar{N}_i[c]$ in order from $\ell = 0, \ldots, n_i$ according to:

$$\bar{N}_i[c_1 \times \{\bar{t}\} \times c_2][x] := \bar{N}_i[c_1 \times \{\bar{t}\} \times c_2][x] + \bar{N}_i[c_1 \times \{-t\} \times c_2][x] \,. \tag{10}$$

It is easy to show that this process results in the same table as Equation (3).

We now describe how to compute N_i from $\bar{N}_i$, which is the same process, but in reverse. We initialize $N_i[c] = \bar{N}_i[c']$ where $c(u) = c'(u)$ if $c(u) \leq 0$ and $\overline{c(u)} = c'(u)$ if $c(u) > 0$. We then update $N_i[c]$ in reverse order of the vertices of X_i, i.e. for $\ell = n_i, n_i - 1, \ldots, 0$ according to:

$$N_i[c_1 \times \{t\} \times c_2][x] := N_i[c_1 \times \{t\} \times c_2][x] - N_i[c_1 \times \{-t\} \times c_2][x]\,. \tag{11}$$

Since the labeling c has length of n_i, the number of operations for the both forward and backward conversion is bounded by $O(nn_i(2r+1)^{n_i})$. ◀

▶ **Lemma 26.** *The convolution tables for X_i, X_j and X_k satisfy:*

$$\bar{N}_i[\bar{c}][x] = \sum_{x_i, x_j \,:\, x_i + x_j - \#_0(X_i, \bar{c}) = x} \bar{N}_j[\bar{c}][x_i] \cdot \bar{N}_k[\bar{c}][x_j] \tag{12}$$

Proof. If a labeling c is consistent with c_1, c_2, we write $c \sim (c_1, c_2)$.

$$\begin{aligned}
\bar{N}_i[\bar{c}][x] &= \sum_{c \,:\, \overline{|c(u)|} = \bar{c}(u)} N_i[c][x] \\
&= \sum_{c \,:\, \overline{|c(u)|} = \bar{c}(u)} \sum_{\substack{x_j, x_k \,:\, x_j + x_k - \#_0(X_i, c) = x \\ c_j, c_k \,:\, c \sim (c_j, c_k)}} (N_j[c_j][x_j] \cdot N_k[c_k][x_k]) \\
&= \sum_{x_j, x_k \,:\, x_j + x_k - \#_0(X_i, c) = x} \sum_{\substack{c \,:\, \overline{|c(u)|} = \bar{c}(u) \\ c_j, c_k \,:\, c \sim (c_j, c_k)}} (N_j[c_j][x_j] \cdot N_k[c_k][x_k]) \\
&= \sum_{x_j, x_k \,:\, x_j + x_k - \#_0(X_i, c) = x} \sum_{\substack{c_j \,:\, \overline{|c_j(u)|} = \bar{c}(u) \\ c_k \,:\, \overline{|c_k(u)|} = \bar{c}(u)}} (N_j[c_j][x_j] \cdot N_k[c_k][x_k]) \\
&= \sum_{x_j, x_k \,:\, x_j + x_k - \#_0(X_i, c) = x} \sum_{c_j \,:\, \overline{|c_j(u)|} = \bar{c}(u)} \bar{N}_j[\bar{c}][x_1] \sum_{c_k \,:\, \overline{|c_k(u)|} = \bar{c}(u)} N_k[c_k][x_k] \\
&= \sum_{x_j, x_k \,:\, x_j + x_k - \#_0(X_i, c) = x} \bar{N}_j[\bar{c}][x_1] \cdot \bar{N}_k[\bar{c}][x_2]
\end{aligned}$$

◀

▶ **Lemma 27.** *The time required to populate the dynamic programming table for join node X_i is $O(n^2(2r+1)^{n_i})$.*

Proof. We update the table A_i of the join node X_i by following steps: (1) computing the indication tables $N_j[c_j][x]$ and $N_k[c_k][x]$ for all possible c_j, c_k and x by Equation (2), (2) computing the convolution tables $\bar{N}_j[\bar{c}][x]$ and $\bar{N}_k[\bar{c}][x]$ via Lemma 25, (3) computing the table $\bar{N}_i[\bar{c}][x]$ of the join node X_i via Lemma 26, (4) computing the indication table $N_i[c_i][x]$ of the join node X_i via Lemma 25 and (5) computing the table $A_i[c_i]$ of the join node X_i by Equation (8). ◀

Proof of Theorem 2

Theorem 2 follows from the correctness and running time analyses for each of the types of nodes of the nice tree decomposition. Using the *finte integer index* property [3, 19], we can reduce the running time of Theorem 2 to $O((2r+1)^{\mathrm{tw}+1}\mathrm{tw}^2 n)$.

For a given bag X_i, let S_c be the minimum partial solution that is associated with a labeling c of X_i; $|S_c| = A[c]$. Let S_1 be the minimum partial solution that is associated with the labeling $c_1 = \{1, 1, \ldots, 1\}$ of X_i.

▶ **Lemma 28** (Claim 5.4 [3] – finite integer index property)**.** *For a given bag X_i, if the minimum partial solution S_c can lead to an optimal solution of G, we have:*

$$||S_c| - |S_1|| \leq n_i + 1\,.$$

▶ **Theorem 29.** *We can populate the dynamic programming table for join node X_i in $O(n_i^2(2r+1)^{n_i})$ time.*

Proof. By Lemma 28, for a fixed $\bar{c}$, there are at most $2n_i + 3$ values $x \in \{1, 2, \ldots, n\}$ such that $\bar{N}[\bar{c}][x] \neq 0$. Therefore, by maintaining non-zero values only, we can populate the table by: (1) computing the convolution tables $\bar{N}_j[\bar{c}][x]$ and $\bar{N}_k[\bar{c}][x]$ via Lemma 25, (2) computing the table $\bar{N}_i[\bar{c}][x]$ of the join node X_i via Lemma 26, (3) computing the indication table $N_i[c_i][x]$ of the join node X_i via Lemma 25 and (4) computing the table $A_i[c_i]$ of the join node X_i by Equation (8). ◀

Clearly, Theorem 29 implies an $O((2r+1)^{\mathrm{tw}+1}\mathrm{tw}^2 n)$ time algorithm for rDS problem.

B Counting Algorithm for connected r-dominating set

To determine $|\mathcal{C}_{k,W}|$ for each W, we use dynamic programming given a tree decomposition $\mathbb{T}$ of G. To simplify the algorithm, we use an *edge-nice* variant of $\mathbb{T}$. A tree decomposition $\mathbb{T}$ is edge-nice if each bag is one of following types:
Leaf: a leaf X_i of $\mathbb{T}$ with $X_i = \emptyset$
Introduce vertex: X_i has one child bag X_j and $X_i = X_j \cup \{v\}$
Introduce edge: X_i has one child bag X_j and $X_i = X_j$ and $E(X_i) = E(X_j) \cup \{e(u, v)\}$
Forget: X_i has one child bag X_j and $X_j = X_i \cup \{v\}$
Join: X_i has two children X_j, X_j and $X_i = X_j = X_k$
We root this tree-decomposition at a leaf bag. Let $G_i = (V_i, E_i)$ be the subgraph formed by the edges and vertices of descendant bags of the bag X_i.

As with the dynamic program for the r-dominating set problem, we use a *distance labeling*, except we have *two types* of 0 labels:

$$c : X_i \to \{-r, \ldots, -1, 0_1, 0_2, 1, \ldots, r\}$$

A vertex u is in a corresponding subsolution if $c(u) \in \{0_1, 0_2\}$ and the subscript of 0 denotes the side of the consistent cut of the subsolution that u is on. *Throughout, we only allow the special root vertex to be labeled* 0_1. We use the same notion of induces as for the non-connected version of the problem, with the additional requirement that we maintain bipartitions (cuts) of the solutions. Specifically, a cut (C_1, C_2) induces the labeling c for a subset X of vertices if $d(u, C_1 \cup C_2) = c(u)$ if $c(u) > 0$, $u \in C_1$ if $c(u) = 0_1$ and $u \in C_2$ if $c(u) = 0_2$. We limit ourselves to locally valid solutions as before.

A dynamic programming table A_i for a bag X_i of $\mathbb{T}$ is indexed by a distance labeling c of X_i, and integers $t \in \{0, \ldots, n\}$ and $W \in \{0, 1, \ldots, 2n^2\}$. $A_i[t, W, c]$ is the number of consistent subcuts (C_1, C_2) of G_i such that: (i) $|C_1 \cup C_2| = t$, (ii) $\omega(C_1 \cup C_2) = W$ and (iii) $C_1 \cup C_2$ induces the labeling c for X_i.

We show how to compute $A_i[t, W, c]$ of the bag X_i given the tables of it children.

Leaf

Let X_i be a non-root leaf of $\mathbb{T}$: $A_i[t, W, \emptyset] = 1$.

Introduce vertex

Let X_i be an introduce vertex bag with its child X_j and $X_i = X_j \cup \{u\}$. Let $c \times d$ is a labeling of X_i where c is a labeling of X_j and d is the label of u. There are four cases for computing $A_i[t, W, c \times d]$ depending on the value of d. Since u is isolated in G_i, we need not worry about checking for local validity.

$$A_i[t, W, c \times d] = \begin{cases} 0 & \text{if } d > 0 \\ A_j[t-1, W - \omega(u), c] & \text{if } d = 0_1 \\ A_j[t-1, W - \omega(u), c] & \text{if } d = 0_2 \\ A_j[t, W, c] & \text{if } d < 0 \end{cases}$$

Introduce edge

Let X_i be an introduce edge bag with its child X_j and $E_i = E_j \cup \{e(u, v)\}$. For any labeling that is not locally valid upon the introduction of uv, that is, if $|c(u) - c(v)| > 1$, we set $A_i(t, W, c) = 0$. If $c(u) = 0_1$ and $c(v) = 0_2$ (or vice versa), c cannot correspond to a consistent subcut, so $A_i(t, W, c) = 0$. If $c(u) = c(v) - d_{G_i}(u, v) \geq 0$ then u positively resolves v. We say that a vertex $x \in G_i$ is *uniquely resolved* by u at distance d if there is no vertex other than u that positively resolves v and $d_{G_i}(u, v) = d$. When the edge $e(u, v)$ is introduced to the bag X_i, some vertices will be positively resolved by u. The vertices $v \in X_i$ that are uniquely resolved by u at distance $d_{G_i}(u, v)$ have negative labels in X_j. We define a map ϕ applied to the label $c(x)$ of a vertex x:

$$\phi(c(x)) = \begin{cases} -c(x) & \text{if } x \text{ is uniquely resolved by } u \text{ at distance } d_{G_i}(u, x) \\ c(x) & \text{otherwise} \end{cases}$$

We use $\phi(c)$ to define the natural extension this map to a full labeling of X_i. We get $A_i[t, W, c] = A_j[t, W, c] + A_j[t, W, \phi(c)]$. In all other cases, the labeling is locally valid, the corresponding subcuts are valid and neither u nor v has been positively resolved, so $A_i[t, W, c] = A_j[t, W, c]$.

Forget

Let X_i be a forget bag with child X_j such that $X_j = X_i \cup \{u\}$. We compute $A_i[t, W, c]$ from $A_j[t, W, c \times d]$ where $c \times d$ is a labeling of X_j where u is labeled d. We say that the labeling $c \times d$ is *forgettable* if $d \geq 0$ or there is a vertex $v \in X_j$ such that $c(v) = d + d_{G_i}(u, v)$. In the first case, u has been dominated already; in the second case, the domination of u must be handled through other vertices in X_j in order for the labeling to be induced by a feasible solution.

$$A_i[t, W, c] = \sum_{d \,:\, c \times d \text{ is forgettable}} A_j[t, W, c \times d]\,.$$

Join

Let X_i be a join bag with children X_j and X_k and $X_i = X_j = X_k$. We say the labeling c_i (for X_i) is *consistent* with labelings c_j and c_k (for X_j and X_k, respectively) if for every $u \in X_i$

- If $c_i(u) = 0_j$ for $j \in \{1, 2\}$, then $c_j(u) = 0_j$ and $c_k(u) = 0_j$.
- If $c_i(u) = t < 0$, then $c_j(u) = t$ and $c_k(u) = t$.

- If $c_i(u) = t > 0$, then one of the following must be true: (a) $c_j(u) = t$ and $c_k(u) = -t$, (b) $c_j(u) = -t$ and $c_k(u) = t$ or (c) $c_j(u) = t$ and $c_k(u) = t$.

Given this, $A_i[t, W, c_i]$ is the product $A_j[t_1, W_1, c_j] \cdots A_k[t_2, W_2, c_k]$ summed over: (i) all t_1 and t_2 such that $t_1 + t_2 - t$ is equal to the number of vertices that are labeled 0_1 or 0_2 by c_i, (ii) all W_1 and W_2 such that $W_1 + W_2 - W$ is equal to the weight of vertices that are labeled 0_1 or 0_2 by c_i and (iii) all c_i and c_j that are consistent with c_j and c_k.

By using the bar-coloring formulation, as for the disconnected case, we can avoid summing over all pairs of consistent distance labellings and instead compute $A_i[t, W, \bar{c}_i]$ as the product $A_j[t_1, W_1, \bar{c}_i] \cdots A_k[t_2, W_2, \bar{c}_i]$ summed over all t_1 and t_2 and all W_1 and W_2 as described above. Using this latter formulation, we can compute A_i in time $O(k^2n^3(2r + 2)^{\mathrm{tw}})$.

Running Time

The number of configurations for each node of $\mathbb{T}$ is $O(kn^2(2r + 2)^{\mathrm{tw}})$. The running time to update leaf, introduce edge, introduce vertex, and forget bags is $O(kn^2(2r + 2)^{\mathrm{tw}})$. The running time to update join bags is $O(k^2n^3(2r + 2)^{\mathrm{tw}})$. Therefore, the total running time of the counting algorithm is $O(nk^2n^3(2r + 2)^{\mathrm{tw}}) = O(k^2n^4(2r + 2)^{\mathrm{tw}})$ after running the algorithm for all possible choices of root vertex ρ.

Fine-Grained Dichotomies for the Tutte Plane and Boolean #CSP*

Cornelius Brand[1], Holger Dell[2], and Marc Roth[3]

1 **Saarland University, Saarbrücken, Germany; and Cluster of Excellence "Multimodal Computing and Interaction" (MMCI), Saarbrücken, Germany**
2 **Saarland University, Saarbrücken, Germany; and Cluster of Excellence "Multimodal Computing and Interaction" (MMCI), Saarbrücken, Germany**
3 **Saarland University, Saarbrücken, Germany; and Cluster of Excellence "Multimodal Computing and Interaction" (MMCI), Saarbrücken, Germany**

Abstract

Jaeger, Vertigan, and Welsh [15] proved a dichotomy for the complexity of evaluating the Tutte polynomial at fixed points: The evaluation is #P-hard almost everywhere, and the remaining points admit polynomial-time algorithms. Dell, Husfeldt, and Wahlén [9] and Husfeldt and Taslaman [12], in combination with the results of Curticapean [7], extended the #P-hardness results to tight lower bounds under the counting exponential time hypothesis #ETH, with the exception of the line $y = 1$, which was left open. We complete the dichotomy theorem for the Tutte polynomial under #ETH by proving that the number of all acyclic subgraphs of a given n-vertex graph cannot be determined in time $\exp\big(o(n)\big)$ unless #ETH fails.

Another dichotomy theorem we strengthen is the one of Creignou and Hermann [6] for counting the number of satisfying assignments to a constraint satisfaction problem instance over the Boolean domain. We prove that all #P-hard cases cannot be solved in time $\exp\big(o(n)\big)$ unless #ETH fails. The main ingredient is to prove that the number of independent sets in bipartite graphs with n vertices cannot be computed in time $\exp\big(o(n)\big)$ unless #ETH fails.

In order to prove our results, we use the block interpolation idea by Curticapean [7] and transfer it to systems of linear equations that might not directly correspond to interpolation.

1998 ACM Subject Classification F.2.1 [Analysis of Algorithms and Problem Complexity] Numerical Algorithms and Problems, G.2.1 [Discrete Mathematics] Combinatorics

Keywords and phrases Computational complexity, counting problems, Tutte polynomial, exponential time hypothesis, forests, independent sets

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.9

1 Introduction

Counting combinatorial objects is at least as hard as detecting their existence, and often it is harder. Valiant [20] introduced the complexity class #P to study the complexity of counting problems and proved that counting the number of perfect matchings in a given bipartite graph is #P-complete. By a theorem of Toda [19], we know that PH $\subseteq$ $\mathrm{P}^{\#\mathrm{P}}$ holds; in particular, for every problem in the entire polynomial-time hierarchy, there is a

* This work was done while the authors were visiting the Simons Institute for the Theory of Computing.

polynomial-time algorithm that is given access to an oracle for counting perfect matchings. This theorem suggests that counting is much harder than decision.

When faced with a problem that is NP-hard or #P-hard, the area of exact algorithms strives to find the fastest exponential-time algorithm for a problem, or find reasons why faster algorithms might not exist. For example, the fastest known algorithm for counting perfect matchings in n-vertex graphs [1] runs in time $2^{n/2} \cdot \mathrm{poly}(n)$. It has been hypothesized that no $O\big(1.99^{n/2}\big)$-time algorithm for the problem exists, but we do not know whether such an algorithm has implications for the strong exponential time hypothesis, which states that for all $\varepsilon > 0$, there is some k such that the problem of deciding satisfiability of boolean formulas in k-CNF on n variables does not have an algorithm running in time $(2-\varepsilon)^n$. However, we know by [8] that the term $O(n)$ in the exponent is asymptotically tight, in the sense that a $2^{o(n)}$-time algorithm for counting perfect matchings would violate the (randomized) exponential time hypothesis (ETH) by Impagliazzo and Paturi [13]. Using the idea of block interpolation, Curticapean [7] strengthened the hardness by showing that a $2^{o(n)}$-time algorithm for counting perfect matchings would violate the (deterministic) counting exponential time hypothesis (#ETH).

Our main results are hardness results under #ETH for 1) the problem of counting all forests in a graph, that is, its acyclic subgraphs, and 2) the problem of counting the number of independent sets in a bipartite graph. If #ETH holds, then neither of these problems has an algorithm running in time $\exp(o(n))$ even in simple n-vertex graphs of bounded maximum degree. We use these results to lift two known "FP vs. #P-hard" dichotomy theorems to their more refined and asymptotically tight "FP vs. #ETH-hard" variants. Here FP is the class of functions computable in polynomial time. Note that #ETH is weaker than ETH, so that our results could also be stated under ETH.

1.1 The Tutte polynomial under #ETH

The Tutte polynomial of a graph G with $G = (V, E)$ is the bivariate polynomial $T(G; x, y)$ defined via

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

where $k(A)$ is the number of connected components of the graph (V, A). The Tutte polynomial captures many combinatorial properties of a graph in a common framework, such as the number of spanning trees, forests, proper colorings, and certain flows and orientations, but also less obvious connections to other fields, such as link polynomials from knot theory, reliability polynomials from network theory, and (perhaps most importantly) the Ising and Potts models from statistical physics. We make no attempt to survey the literature or the different applications for the Tutte polynomial, and instead refer to the upcoming CRC handbook on the Tutte polynomial [10].

Since $T(G; -2, 0)$ corresponds to the number of proper 3-colorings of G, we cannot hope to compute all coefficients of $T(G; x, y)$ in polynomial time. Instead, the literature and this paper focus on the complexity of evaluating the Tutte polynomial at fixed evaluation points. That is, for each $(x, y) \in \mathbf{Q}^2$, we consider the function $T_{x,y}$ defined as $G \mapsto T(G; x, y)$. Jaeger, Vertigan, and Welsh [15] proved that this function is either #P-hard to compute or has a polynomial-time algorithm. In particular, if (x, y) satisfies $(x-1)(y-1) = 1$, then $T_{x,y}$ corresponds to the 1-state Potts model and has a polynomial-time algorithm, and if (x, y) is one of the four points $(1, 1)$, $(-1, -1)$, $(0, -1)$, or $(-1, 0)$, it also has a polynomial-time

algorithm; the most interesting point here is $T(G;1,1)$, which corresponds to the number of spanning trees in G and happens to admit a polynomial-time algorithm.

A trivial algorithm to compute the Tutte polynomial runs in time $2^{O(m)}$, where m is the number of edges. Björklund et al. [2] proved that there is an algorithm running in time $\exp\big(O(n)\big)$, where n is the number of vertices. Dell et al. [8] proved for all hard points, except for points with $y = 1$, that an $\exp\big(o(n/\log^3 n)\big)$-time algorithm for $T_{x,y}$ on simple graphs would violate #ETH. Distressingly, this result not only left open one line, but also left a gap in the running time. Curticapean [7] introduced the technique of block interpolation to close the running time gap: Under #ETH, there does not exist an $\exp\big(o(n)\big)$-time algorithm for $T_{x,y}$ on simple graphs at any hard point (x, y) with $y \neq 1$.[1]

Our contributions: We resolve the complexity of the missing line $y = 1$ under #ETH. On this line, the Tutte polynomial counts forests weighted in some way, and the main result is the following theorem.

▶ **Theorem 1** (Forest counting is hard under #ETH). *If #ETH holds, then there exist constants $\varepsilon, C > 0$ such that no $O(\exp(\varepsilon n))$-time algorithm can compute the number of all forests in a given simple n-vertex graph with at most $C \cdot n$ edges.*

The fact that the problem remains hard even on simple sparse graphs makes the theorem stronger. The previously best known lower bound under #ETH was that forests cannot be counted in time $O\big(\exp(n^\delta)\big)$ where $\delta > 0$ is some constant depending on the instance blow-up caused by the known #P-hardness reductions for forest counting; the thesis of Taslaman [18] shows a detailed proof for $\delta = \frac{1}{8}$. Our approach yields a #P-hardness proof for forest counting that is simpler than the proofs we found in the literature.[2]

Combined with all previous results [15, 8, 7], we can now formally state a complete #ETH dichotomy theorem for the Tutte polynomial over the reals.

▶ **Corollary 2** (Dichotomy for the real Tutte plane under #ETH). *Let $(x, y) \in \mathbf{Q}^2$. If (x, y) satisfies*

$$(x-1)(y-1) = 1 \text{ or } (x,y) \in \big\{(1,1),(-1,-1),(0,-1),(-1,0)\big\},$$

then $T_{x,y}$ can be computed in polynomial time. Otherwise $T_{x,y}$ is #P-hard and, if #ETH is true, then there exists $\varepsilon > 0$ such that $T_{x,y}$ cannot be computed in time $\exp(\varepsilon n)$, even for simple graphs.

The result also holds for sparse simple graphs. We stated the results only for rational numbers in order to avoid discussions about how real numbers should be represented.

For the proof of Theorem 1, we establish a reduction chain that starts with the problem of counting perfect matchings on sparse graphs, which is known to be hard under #ETH. As an intermediate step, we find it convenient to work with the multivariate forest polynomial as defined, for example, by Sokal [17]. After a simple transformation of the graph, we are able to extract the number of perfect matchings of the original graph from the multivariate forest polynomial of the transformed graph, even when only two different variables are used. Subsequently, we use Curticapean's idea of block interpolation [7] to reduce the problem of

[1] The conference version of [7] does not handle the case $y = 0$, but the full version (to appear) does.

[2] For the #P-hardness of forest counting, [15] refers to private communication with Mark Jerrum as well as the PhD thesis of Vertigan [21]. A self-contained (but involved) proof appears in "Complexity of Graph Polynomials" by Steven D. Noble, chapter 13 of [11].

computing all coefficients of the bivariate forest polynomial to the problem of evaluating the univariate forest polynomial on multigraphs where all edge multiplicity are bounded by a constant. Finally, we replace parallel edges with parallel paths of constant length to reduce to the problem of evaluating the univariate forest polynomial on simple graphs.

1.2 #CSP over the Boolean domain under #ETH

In the second part of this paper, we consider constraint satisfaction problems (CSPs) over the Boolean domain, which are a natural generalization of the satisfiability problem for k-CNF formulas. A *constraint* is a relation $R \subseteq \{0,1\}^k$ for some $k \in \mathbf{N}$, and a set Γ of constraints is a *constraint language*. CSP(Γ) is the constraint satisfaction problem where all constraints occurring in the instances are of a type contained in Γ, and #CSP(Γ) is the corresponding counting version, which wants to compute the number of satisfying assignments. If all constraints happen to be *affine*, that is, they are linear equations over GF(2), then the number of solutions can be determined in polynomial time by applying Gaussian elimination and determining the size of the solution space. Creignou and Hermann [6] prove that, as soon as you allow even just one constraint type that is not affine, the resulting CSP problem is #P-hard.

Our contributions: We prove that all Boolean #CSPs that are #P-hard are also hard under #ETH. The #P-hardness is established in [6] by reductions from counting independent sets in bipartite graphs, which we prove to be hard in the following theorem.

▶ **Theorem 3** (Counting bipartite independent sets is hard under #ETH)**.** *If #ETH holds, then there exist constants $\varepsilon > 0$ and $D \in \mathbf{N}$ such that no $O(\exp(\varepsilon n))$-time algorithm can compute the number of independent sets in bipartite n-vertex graphs of maximum degree at most D.*

The fact that the problem is hard even on graphs of bounded degree makes the theorem stronger. We remark that the number of independent sets in bipartite graphs has a prominent role in counting complexity. Currently, the complexity of *approximating* this number is unknown, and many problems in approximate counting turn out to be polynomial-time equivalent to approximately counting independent sets in bipartite graphs. Theorem 3 shows that this mysterious situation does not occur for the exact counting problem in the exponential-time setting: it is just as hard as counting satisfying assignments of 3-CNFs.

The #P-hardness of counting independent sets in bipartite graphs was established by Provan and Ball [16]. The main ingredient in their proof is a system of linear equations that does not seem to correspond to polynomial interpolation directly. We prove Theorem 3 by transferring the block interpolation idea from [7] to this system of linear equations, which we do using a Kronecker power of the original system.

Theorem 3 combined with existing reductions in [6] yields the fine-grained dichotomy.

▶ **Corollary 4** (Creignou and Hermann under #ETH)**.** *Let Γ be a finite constraint language. If every constraint in Γ is affine, then #CSP(Γ) has a polynomial-time algorithm. Otherwise #CSP(Γ) is #P-complete, and if #ETH holds, it cannot be computed in time $\exp(o(n))$ where n is the number of variables.*

We consider Corollary 4 to be a first step towards understanding the fine-grained complexity of technically much more challenging dichotomies, such as the ones for counting CSPs with complex weights of Cai and Chen [3], or the dichotomy for Holant problems with symmetric signatures over the Boolean domain of Cai, Lu and Cia [4].

2 Preliminaries

Given a matrix A of size $m_1 \times n_1$ and a matrix B of size $m_2 \times n_2$ their *Kronecker product* $A \otimes B$ is a matrix of size $m_1 m_2 \times n_1 n_2$ given by

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \vdots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix}.$$

Let $A^{\otimes n}$ be the matrix defined by $A^{\otimes 1} = A$ and $A^{\otimes n+1} = A \otimes A^{\otimes n}$. Furthermore, if A and B are quadratic matrices of size n_a and n_b, respectively, it is known that $\det(A \otimes B) = \det(A)^{n_b} \cdot \det(B)^{n_a}$ holds.

The exponential time hypothesis (ETH) by Impagliazzo and Paturi [13] is that satisfiability of 3-CNF formulas cannot be computed substantially faster than by trying all possible assignments. The counting version of this hypothesis [8], which is a weaker assumption (clearly, counting the number of solutions entails deciding existence of a solution), reads as follows:

(#ETH) *There is a constant $c > 0$ such that no deterministic algorithm can compute #3-SAT in time $\exp(c \cdot n)$, where n is the number of variables.*

A different way of formulating #ETH is to say no algorithm can compute #3-SAT in time $\exp(o(n))$. The latter statement is clearly implied by the formal statement, and it will be more convenient for discussion to use this form.

The sparsification lemma by Impagliazzo, Paturi, and Zane [14] is that every k-CNF formula φ can be written as the disjunction of $2^{\varepsilon n}$ formulas in k-CNF, each of which has at most $c(k, \varepsilon) \cdot n$ clauses. Moreover, this disjunction of sparse formulas can be computed from φ and ε in time $2^{\varepsilon n} \cdot \mathrm{poly}(m)$. The density $c = c(k, \varepsilon)$ is the *sparsification constant*, and the best known bound is $c(k, \varepsilon) = (k/\varepsilon)^{3k}$ [5]. It was observed [8] that the disjunction can be made so that every assignment satisfies at most one of the sparse formulas in the disjunction, and so the sparsification lemma applies to #ETH as well. In particular, #ETH implies that #3-SAT cannot be computed in time $\exp(o(m))$, where m is the number of clauses.

We also make use of the following result, whose proof is based on block interpolation.

▶ **Theorem 5** (Curticapean [7]). *If #ETH holds, then there are constants $\varepsilon, D > 0$ such that neither of the following problems have $O(2^{\varepsilon n})$-time algorithms n-vertex graphs G, even if G is simple and of maximum degree at most D:*

- *Computing the number of perfect matchings of G.*
- *Computing the number of independent sets of G.*

In the proof, we recover a univariate graph polynomial of high degree, and a classical approach to do so is polynomial interpolation. Interpolation works by evaluating a polynomial at different points and then solving some linear equations. In our case, however, we can evaluate the univariate polynomial only at certain discrete points, which involves constructing graphs with a number of edges that grows roughly as the square of the number of evaluation points. Since we need roughly as many points as the (large) degree of the polynomial, the conditional lower bounds we can achieve are not optimal.

While univariate interpolation appears to be insufficient for obtaining tight bounds under ETH, the univariate polynomials we consider have natural multilinear variants. The univariate polynomial can be reconstructed from the multilinear one, but since the multilinear polynomial has linearly many variables, we would need exponentially many evaluation

points to reconstruct all coefficients of the multilinear polynomial. This is not feasible in a subexponential-time reduction.

Block interpolation is able to find a sweet spot between univariate interpolation, where the constructed graphs would grow super-linearly, and multilinear interpolation, where the number of evaluation points would be exponential. For this, we arbitrarily partition the variables of the multilinear polynomial into (almost) equal-size parts (or *blocks*) and identify any variables in the same part. By choosing large but constant-sized parts, the individual degree of each variable is constant, but their number is sub-linear, which allows a subexponential-time reduction to work. At the same time, the graphs that we need to query are larger by at most by a constant factor. After the block interpolation, we can recover the univariate polynomial as desired.

3 Counting forests is #ETH-hard

Let $\mathcal{F}(G)$ be the set of all *forests* of G, that is, edge subsets $A \subseteq E(G)$ such that the graph $(V(G), A)$ is acyclic. For $y = 1$, only the terms with $k(A) + |A| - |V| = 0$ survive, and we get the following:

$$T(G; x, 1) = \sum_{A \in \mathcal{F}(G)} (x-1)^{k(A)-k(E)} . \tag{2}$$

We want to prove that, for every fixed $x \neq 1$, computing the value $T(G; x, 1)$ for a given graph G is hard under #ETH. In particular, this is true for $T(G; 2, 1)$, which is the number of forests in G. The goal of this section is to prove the following theorem.

▶ **Theorem 6.** *Let $x \in \mathbf{R} \setminus \{1\}$. If #ETH holds, then there exist $\varepsilon, C > 0$ such that the function that maps simple n-vertex graphs G with at most $C \cdot n$ edges to the value $T(G; x, 1)$ cannot be computed in time $2^{\varepsilon n}$.*

Theorem 6 yields Theorem 1 as its special case with $x = 2$.

3.1 The multivariate forest polynomial

A *weighted graph* is a graph G in which every edge $e \in E(G)$ is endowed with a weight w_e, which is an element of some ring. We use the *multivariate forest polynomial*, defined e.g. by Sokal [17, (2.14)] as follows:

$$F(G; w) = \sum_{A \in \mathcal{F}(G)} \prod_{e \in A} w_e .$$

Projecting all weights w_e onto a single variable x yields the *univariate forest polynomial*:

$$F(G; x) = \sum_{A \in \mathcal{F}(G)} x^{|A|} = \sum_{k=0}^{|E(G)|} a_k(G) x^k ,$$

where $a_k(G)$ is the number of forests with k edges in G. For all $x \in \mathbf{R} \setminus \{1\}$, the formal relation between $T(G; x, 1)$ and the univariate forest polynomial is given by the identity

$$T(G; x, 1) = (x-1)^{|V|-k(E)} \sum_{A \in \mathcal{F}(G)} (x-1)^{-|A|} = (x-1)^{|V|-k(E)} \cdot F\Big(G; \frac{1}{x-1}\Big) . \tag{3}$$

The first equality follows from (2) and the fact that $k(A) + |A| - |V| = 0$ holds if and only if A is a forest.

In particular, evaluating the forest polynomial and evaluating the Tutte polynomial for $y = 1$ are polynomial-time equivalent.

For a forest $A \in \mathcal{F}(G)$, let $\mathcal{C}(A)$ be the family of all sets $U \subseteq V(G)$ such that $U \neq \emptyset$ and U is a maximal connected component in A; clearly, each such U is the vertex set of a tree in the forest, where we also allow trees with $|U| = 1$.

▶ **Lemma 7** (Adding an apex). *Let G be a weighted graph, and let G' be obtained from G by adding a new vertex a and joining it with each vertex $v \in V(G)$ using edges of weight z_v. Then*

$$F(G') = \sum_{A \in \mathcal{F}(G)} \prod_{e \in A} w_e \cdot \prod_{U \in \mathcal{C}(A)} \left(1 + \sum_{u \in U} z_u\right). \tag{4}$$

Moreover, when we set $z_v = -1$ for all $v \in V(G)$ and $w_e = w$ for all $e \in E(G)$, we have that the coefficient of $w^{n/2}$ in $F(G')$ is equal to the number of perfect matchings in G.

Proof. We first define a projection ϕ that maps a forest A' in the graph G' to the forest $A = \phi(A')$ in the original graph G. In particular, ϕ simply removes all edges added in the construction of G', that is, we define $\phi(A') = E(G) \cap A'$ for all $A' \in \mathcal{F}(G')$. Now $\phi(A')$ is a forest in G.

Next we characterize the forests A' that map to the same A under ϕ. Let A be a fixed forest in G. Then a forest A' in G' maps to A under ϕ if and only if set $X := A' \setminus A$ satisfies the following property:

(P) For all trees $U \in \mathcal{C}(A)$, at most one edge of X is incident to a vertex of U.

The forward direction of this claim follows from the fact that A' is a forest, and so in addition to any tree $U \in \mathcal{C}(A)$ it can contain at most one edge connecting U to a; otherwise the tree and the two edges to a would contain a cycle in A'. For the backward direction of the claim, observe that adding a set X with the property (P) to A cannot introduce a cycle.

Finally, we calculate the weight contribution of all A' that map to the same A. Let A' be a forest in G, let $A = \phi(A')$ and $X = A' \setminus A$. The weight contribution of A' in the definition of $F(G')$ is $\prod_{e \in A'} w'_e$. For all $e \in A$, we have $w'_e = w_e$. For each $e \in X$, we let $v_e \in V(G)$ be the vertex with $e = \{a, v_e\}$, and we have $w'_e = z_{v_e}$. Thus the overall weight contribution of all A' with $\phi(A) = A'$ is

$$\sum_{\substack{A' \in \mathcal{F}(G') \\ \phi(A') = A}} \prod_{e \in A} w'_e = \prod_{e \in A} w_e \cdot \sum_{X} \prod_{e \in X} z_{v_e} = \prod_{e \in A} w_e \cdot \prod_{U \in \mathcal{C}(A)} \left(1 + \sum_{u \in U} z_u\right). \tag{5}$$

The sum in the middle is over all X with the property (P), and the first equality follows from the bijection between forests A' and sets X with property (P). For the second equality, we use property (P) and the distributive law. We obtain (4) by taking the sum of equations (5) over all $A \in \mathcal{F}(G)$.

For the moreover part of the lemma, note that the stated settings of the edge weights for G' yields

$$F(G') = \sum_{A \in \mathcal{F}(G)} w^{|A|} \prod_{U \in \mathcal{C}(A)} (1 - |U|).$$

The coefficient of $w^{n/2}$ in $F(G')$ satisfies

$$[w^{n/2}]F(G') = \sum_{\substack{A \in \mathcal{F}(G) \\ |A|=n/2}} \prod_{U \in \mathcal{C}(A)} (1 - |U|)\,. \tag{6}$$

Since $(V(G), A)$ is an acyclic graph with exactly $n/2$ edges, it is either a perfect matching or it contains an isolated vertex. If it contains an isolated vertex v, then we have $\{v\} \in \mathcal{C}(A)$ and thus the product in (6) is equal to zero. It follows that A does not contribute to the sum if it is not a perfect matching. On the other hand, if A is a perfect matching, then we have $|U| = 2$ for all $U \in \mathcal{C}(A)$, so the product in (6) is equal to 1 or -1, depending on the parity of $n/2$. Overall, we obtain that $[w^{n/2}]F(G')$ is equal in absolute value to the number of perfect matchings of G. ◀

Lemma 7 shows that computing the multivariate forest polynomial is at least as hard as counting perfect matchings; moreover, this is true even if at most two different edge weights are used. Next we argue how to reduce from the multivariate forest polynomial with at most two distinct weights to the problem of evaluating the univariate polynomial in multigraphs. We do so via an oracle serf-reduction, whose queries are sparse multigraphs in which each edge has at most a constant number of parallel edges.

▶ **Lemma 8** (From two weights to small weights using block interpolation). *Let x and y be two variables, and let $z \in \mathbf{R} \setminus \{0\}$ be fixed. There is an algorithm as follows:*

1. *Its input is a weighted graph (G, w) with $w_e \in \{x, y\}$ for all $e \in E(G)$, and a real $\varepsilon > 0$.*
2. *It outputs all coefficients of the bivariate polynomial $F(G; w)$.*
3. *It runs in time $2^{\varepsilon |E(G)|} \cdot \mathrm{poly}(|G|)$.*
4. *It has access to an oracle that computes $F(G; z \cdot w')$, where w' can be any weight function that assigns integer weights w'_e satisfying $0 \leq w'_e \leq C_\varepsilon$ for some constant C_ε that only depends on ε.*

We remark that non-negative integer multiples of z, say $z \cdot w'_e$, can be thought of as w'_e parallel edges of weight z in a multigraph. The quantity $F(G'; z)$ for this multigraph G' is then equal to the value $F(G; z \cdot w')$ of the weighted forest polynomial of G at $z \cdot w'$. In particular, $F(G'; 1)$ is the number of forests in G'.

Proof. Let (G, w) with $w_e \in \{x, y\}$ for all $e \in E$ and $\varepsilon > 0$ be given as input, and let $m := |E(G)|$. We define $C_\varepsilon \in \mathbf{N}$ as a large enough constant to be determined later. The algorithm assigns a new weight $z \cdot w'_e$ to each edge e, where each w'_e is chosen from the set of indeterminates $X \cup Y$ with $X = \{x_1, \ldots, x_{m/C_\varepsilon}\}$ and $Y = \{y_1, \ldots, y_{m/C_\varepsilon}\}$ in the following way: If $w_e = x$, choose $w'_e \in X$, and if $w_e = y$, choose $w'_e \in Y$. We further demand that the number of edges sharing the same weight is at most C_ε for each weight in $X \cup Y$. Among all such assignments $z \cdot w'$, we pick an arbitrary one. We now consider the polynomial $F(G; z \cdot w')$. It has at most $2m/C_\varepsilon$ variables and the maximum degree of each variable is at most C_ε, so $F(G; z \cdot w')$ has at most $(C + 1)^{2m/C_\varepsilon}$ monomials. The coefficients of this polynomial can be reconstructed when its values are given for all evaluation points in $z \cdot [0, C_\varepsilon]^{2m/C_\varepsilon}$, which is an $2m/C_\varepsilon$-dimensional grid dilated by a factor z (that is, for any two grid points, the distance between any two entries in the same coordinate is z).

Since each evaluation point only uses non-negative integer multiples of z between 0 and $z \cdot C_\varepsilon$, we can obtain the values at these evaluation points by querying the oracle for $F(G; z \cdot w')$ that we are given. The number of evaluation points in the grid is equal to $(C_\varepsilon + 1)^{2m/C_\varepsilon}$. The claim on the running time follows since the interpolation can be

performed in time $\text{poly}\left((C_\varepsilon+1)^{2m/C_\varepsilon}\right)$, which is at most $C_\varepsilon \cdot 2^{\varepsilon m}$ when C_ε is chosen large enough depending on ε.

In order to obtain the coefficient of $x^i y^j$ in $F(G;w)$, we compute the image of $F(G; z \cdot w')$ under the projection that maps all variables in X to x/z and all variables in Y to y/z. That is, we sum up the coefficients of $F(G; z \cdot w')$ corresponding to the same monomial $x^i y^j$, and divide by the factor z^{i+j}. ◀

The combination of Lemma 7 and Lemma 8 shows, for all fixed $x \neq 0$, that it is hard to evaluate $F(G;x)$ for multigraphs with at most a constant number of parallel edges. Next we apply a stretch to make the graphs simple. To this end, we calculate the effect of a k-stretch on the univariate forest polynomial of a graph.

▶ **Lemma 9** (The forest polynomial under a k-stretch)**.** *Let G be a multigraph with m edges, where every edge is weighted with $w \in \mathbf{R}$ and let k be a positive integer such that the number $g_k(w)$ with*

$$g_k(w) = \frac{w^k}{(w+1)^k - w^k}$$

is well-defined. Let G' be the simple graph obtained from G by replacing every edge by a path of k edges. Then we have

$$F(G';w) = \left((w+1)^k - w^k\right)^m \cdot F\big(G; g_k(w)\big)\,.$$

Proof. We define a mapping ϕ that maps forests in G' to forests in G as follows: We add an edge $e \in E(G)$ to $A = \phi(A')$ if and only if A' contains all k edges of G' that e got stretched into. That is, subgraphs A' that only differ by edges in "incomplete paths" are mapped to the same multigraph A by ϕ.

Clearly, ϕ partitions $\mathcal{F}(G')$ into sets of forests with the same image under ϕ. Let A be a forest in G, and let us describe a way to generate all A' with $\phi(A') = A$. First, for each $e \in A$, we add its corresponding path in G' of length k to A'. Moreover, for each edge $e \in E(G) \setminus A$, we can add to A' any proper subset of edges from the k-path in G' that corresponds to e. Therefore, at each $e \in E(G) \setminus A$ independently, there are $\binom{k}{i}$ ways to extend A' by i edges to a forest in G'. A forest A' can be obtained in this fashion if and only if $\phi(A') = A$ holds.

For a fixed A, let us consider all summands $w^{|A'|}$ in $F(G';w)$ with $\phi(A') = A$. By the above considerations, the total weight contribution of these summands is $w^{k\cdot|A|} \cdot \left(\sum_{i=0}^{k-1} \binom{k}{i} w^i\right)^{m-|A|}$, which equals $w^{k\cdot|A|} \cdot \left((w+1)^k - w^k\right)^{m-|A|}$ by the binomial theorem. These remarks justify the following calculation for the forest polynomial:

$$\begin{aligned} F(G';w) &= \sum_{A \in \mathcal{F}(G)} \sum_{\substack{A' \in \mathcal{F}(G') \\ \phi(A') = A}} w^{|A'|} = \sum_{A \in \mathcal{F}(G)} w^{k\cdot|A|} \cdot \Big((w+1)^k - w^k\Big)^{m-|A|} \\ &= \Big((w+1)^k - w^k\Big)^m \cdot \sum_{A \in \mathcal{F}(G)} \left(\frac{w^k}{(w+1)^k - w^k}\right)^{|A|}. \end{aligned}$$

Since the sum in the last line is equal to $F\big(G; g_k(w)\big)$, this concludes the proof. ◀

We are now in position to formally prove the main theorem of this section.

Proof of Theorem 6. Let $x \in \mathbf{R} \setminus \{1\}$. Suppose that, for all $\varepsilon > 0$, there exists an algorithm B to compute the mapping $G \mapsto T(G; x, 1)$ in time $2^{\varepsilon n}$ for given simple graphs G

with at most $C'_\varepsilon n$ edges, where C'_ε will be chosen later. By (3), algorithm B can be used to compute values $F(G;(x-1)^{-1})$ with no relevant overhead in the running time, so let $t = (x-1)^{-1}$. Given such an algorithm (or family of algorithms), we devise a similar algorithm for counting perfect matchings, which together with Theorem 5 implies that #ETH is false.

Let G be a simple n-vertex graph with at most $C \cdot n$ edges. Let G' be the graph obtained from G as in Lemma 7 by adding an apex, labeling the edges incident to the apex with the indeterminate z, and all other edges with the indeterminate w. By Lemma 7, the coefficients of the corresponding bivariate forest polynomial of G' are sufficient to extract the number of perfect matchings of G, so it remains to compute these coefficients.

To obtain the coefficients, we use Lemma 8. The reduction guaranteed by the lemma produces $2^{\varepsilon m}$ multigraphs H, all with the same vertex set $V(G')$. Moreover, each H has at most $C_\varepsilon|E(G')| = C_\varepsilon(|E(G)|+n) \leq O(C_\varepsilon n)$ edges, and the multiplicity of each edge is at most C_ε. Finally, each edge of each H is assigned the same weight z, which we will choose later.

The reduction makes one query for each H, where it asks for the value $F(H;z)$. Our assumed algorithm however only works for simple graphs, so we perform a 3-stretch to obtain a simple graph H' with at most $3|E(H)| \leq O(C_\varepsilon n)$ edges. Lemma 9 allows us to efficiently compute the value $F(H;z)$ when we are given the value $F(H';t)$ and $z = g_3(t)$ holds. Since g_k is a total function whenever k is a positive odd integer, and 3 is indeed odd, the value $g_3(t)$ is well-defined, and we set $z = g_3(t)$.

We set C'_ε large enough so that $E(H') \leq C'_\varepsilon \cdot n$ holds. Tracing back the reduction chain, we can use algorithm B to compute $T(H';x,1)$ in time $2^{\varepsilon n}$ any $\varepsilon > 0$. Using (3), we get the value of $F(H';t)$ since $x \neq 1$. This, in turn, yields the value of $F(H;z)$ since $(z+1)^k - z^k \neq 0$ and $g_3(t) = z$. We do this for each of the $2^{\varepsilon m}$ queries H that the reduction in Lemma 8 makes. Finally, the latter reduction outputs the coefficients of the bivariate forest polynomial of G', which contains the information on the number of perfect matchings of G.

To conclude, assuming the existence of the algorithm family B, we are able to count perfect matching in time $\text{poly}(2^{\varepsilon m})$ for all $\varepsilon > 0$, which implies via Theorem 5 that #ETH is false. ◀

Note that the construction from the proof of Theorem 1 implies hardness of $T(G;x,1)$ for tripartite G, and also in the bipartite case whenever $x \neq -1$.

4 Counting solutions to Boolean CSPs under #ETH

In this section, we prove that the #P-hard cases of the dichotomy theorem for Boolean CSPs by Creignou and Hermann [6] are also hard under #ETH. The main difficulty is to establish #ETH-hardness of counting independent sets in bipartite graphs. We do so first, and afterwards observe that all other reductions in [6] can be used without modification.

4.1 Counting Independent Sets in Bipartite Graphs is #ETH-hard

We prove that the problem of counting independent sets in bipartite graphs admits no subexponential algorithm under #ETH, even for sparse and simple graphs.

Proof of Theorem 3. We reduce from the problem of counting independent sets in graphs of bounded degree; by Theorem 5, this problem does not have a subexponential-time algorithm. First we note that a set is an independent set if and only its complement is a vertex cover.

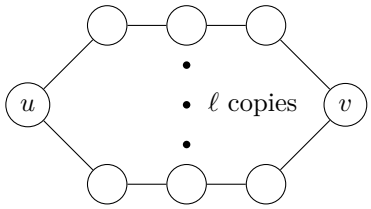


Figure 1 The gadget H_ℓ of Provan and Ball [16] as used in the proof of Theorem 3. It corresponds to an ℓ-fattening of the edge $\{u, v\}$, followed by a 4-stretch of each of the ℓ parallel edges.

Hence their numbers are equal. We devise a subexponential-time oracle reduction family to reduce counting vertex covers in general to counting them in bipartite graphs.

Given a graph G with n vertices and m edges, and a running time parameter $d \in \mathbf{N}$, the reduction works as follows. We partition the edges into $\frac{|E|}{d}$ blocks of size at most d each. We denote the blocks by $B_1, \ldots, B_{\frac{m}{d}}$. Next, for each $\vec{\ell} = (\ell_1, \ldots, \ell_{\frac{m}{d}}) \in \mathbf{N}^{m/d}$, we denote $G_{\vec{\ell}}$ as the graph obtained from G by replacing each edge $e \in B_i$ with a copy of the gadget H_{ℓ_i} shown in Figure 1. Note that $G_{\vec{\ell}}$ is bipartite.

▶ **Observation 10** (Provan and Ball). *The number of vertex covers of H_ℓ containing neither u nor v is 2^ℓ, the number of vertex covers containing a particular one of u or v is 3^ℓ, and the number of vertex covers containing both u and v is 5^ℓ.*

We follow the proof of Provan and Ball, but do so in a block-wise fashion. To this end, let T be the set of all $(m/d) \times 3$ matrices with entries from $\{0, \ldots, d\}$. The *type* of a set $S \subseteq V(G)$ is the matrix $t \in T$ such that, for all $i = 1 \ldots \frac{m}{d}$,

1. t_{i1} is equal to the number of edges $e \in B_i$ with $|e \cap S| = 0$,
2. t_{i2} is equal to the number of edges $e \in B_i$ with $|e \cap S| = 1$, and
3. t_{i3} is equal to the number of edges $e \in B_i$ with $|e \cap S| = 2$.

Every set $S \subseteq V(G)$ has exactly one type. Let x_t be the number of all sets $S \subseteq V(G)$ that have type t.

We classify vertex covers $C \subseteq V(G_{\vec{\ell}})$ of $G_{\vec{\ell}}$ by their intersection with $V(G)$, so let $S = C \cap V(G)$ and let t be the type of S. By Observation 10 and the fact that all inserted gadgets act independently after conditioning on the intersection of the vertex covers of $G_{\vec{\ell}}$ with $V(G)$, there are exactly $\prod_{i=1}^{m/\ell} (2^{t_{i1}} 3^{t_{i2}} 5^{t_{i3}})^{\ell_i}$ vertex covers C' whose intersection with $V(G)$ is S. Moreover, the number of sets S of type t is equal to x_t. Hence the number $N_{\vec{\ell}}$ of vertex covers of $G_{\vec{\ell}}$ satisfies

$$N_{\vec{\ell}} = \sum_{t \in T} x_t \cdot \prod_{i=1}^{m/d} \left(2^{t_{i1}} 3^{t_{i2}} 5^{t_{i3}}\right)^{\ell_i} \tag{7}$$

Since $G_{\vec{\ell}}$ is bipartite, our reduction can query the oracle to obtain the numbers $N_{\vec{\ell}}$ for all $\vec{\ell} \in [(d+1)^3]^{\frac{n}{d}}$. This yields a system of linear equations of type (7), where the x_t for $t \in T$ are the unknowns; note that we have exactly $|T|$ equations and unknowns. Let M be the corresponding $|T| \times |T|$ matrix, so that the system can be written as $N = M \cdot x$.

It remains to prove that M is invertible. For this, we observe that M can be decomposed into a tensor product of smaller matrices as follows. Let A be the $(d+1)^3 \times (d+1)^3$ where the row indices ℓ are from $[(d+1)^3]$, the column indices τ are from $\{0, \ldots, d\}^3$, and the entries are defined via $A_{\ell\tau} = (2^{\tau_1} 3^{\tau_2} 5^{\tau_3})^\ell$. Provan and Ball, as well as the reader, observe that A is the transpose of a Vandermonde matrix. Due to the uniqueness of the prime factorization,

the evaluation points $2^{\tau_1}3^{\tau_2}5^{\tau_3}$ are distinct for distinct τ, and thus $\det(A) \neq 0$. Furthermore, we observe that $M = A^{\otimes \frac{n}{d}}$ holds, which implies $\det(M) \neq 0$ and that M is invertible.

Since M is invertible, we can solve the equation system $N = M \cdot x$ in time polynomial in its size, and compute x_t for all $t \in T$. Finally, we compute the sum of x_t over all matrices t whose first column contains only zeros. This yields the number of all sets $S \subseteq V(G)$ that intersect every edge of G at least once, that is, the number of vertex covers of G which equals, as mentioned above, the number of independent sets of G.

Assume that #ETH holds, and let $\varepsilon, D > 0$ be the constants from Theorem 5, which are such that no algorithm can count independent sets in general graphs of maximum degree D in time $2^{\varepsilon n}$. We apply our reduction to such a graph; it makes at most $2^{O(\log d \cdot m/d)}$ queries to the oracle. Since $m \leq Dn$ holds, and the running time for solving the linear equation system is polynomial in the number of queries, we can choose $d \in \mathbf{N}$ to be a large enough constant depending on $\varepsilon > 0$ to achieve an overall running time of $O(2^{\frac{1}{2}\varepsilon n})$ for the reduction. Also note that the queries to the oracle for bipartite graphs have degree at most $(d+1)^3 \cdot D$, which is a constant that only depends on ε. If there was an algorithm for counting independent sets in bipartite graphs that ran in time $O(2^{\frac{1}{2}\varepsilon n})$, we would get a combined algorithm for counting independent sets in general graphs that would be faster than the choice of ε and D would allow. Hence, under #ETH, there are constants $\varepsilon', D' > 0$ such that no $O(2^{\varepsilon' n})$-time algorithm can count all independent sets on graphs of maximum degree at most D'. ◀

4.2 The Boolean CSP dichotomy

Instances of the constraint satisfaction problem #CSP(Γ) are conjunctions of relations in Γ applied to variables over the Boolean domain and the goal is to compute the number of satisfying assignments. A satisfying assignment is an assignment to the variables such that the formula evaluates to true, that is, every relation in the conjunction evaluates to true. A more detailed description of the problem can be found in the paper of Creignou and Hermann [6].

Creignou and Hermann prove Theorem 4 by reducing either from #Pos2Sat, the problem of counting satisfying assignments of a 2-CNF where every literal is positive, or from #Imp2Sat, the problem of counting satisfying assignments of a 2-CNF where every clause contains exactly one positive and one negative literal. A straightforward analysis of the construction reveals that the reductions only lead to a linear overhead. More precisely:

▶ **Observation 11.** *Given an instance of* #Pos2Sat *or* #Imp2Sat *with n variables and a set Γ of logical relations such that at least one of the relations is not affine, the construction of Creignou and Hermann results in an instance of* #CSP(Γ) *of size $c \cdot n$ where c only depends on the size of the largest non-affine relation in Γ.*

Therefore it suffices to establish that neither #Pos2Sat nor #Imp2Sat have a $2^{o(n)}$-time algorithm. Since #Pos2Sat is identical to counting vertex covers in (general) graphs, Theorem 5 applies here. The #ETH-hardness of #Imp2Sat follows by a known reduction from counting independent sets in bipartite graphs, which we include here for completeness.

▶ **Lemma 12.** *Assuming* #ETH*, there is no algorithm that solves* #Imp2Sat *in time $2^{o(n)}$ where n is the number of variables.*

Proof. Given a bipartite graph $G = (V \dot\cup U, E)$ with constant degree we construct a 2-CNF F by adding a clause $(v \to u)$ for every edge $\{v, u\} \in E$. Now the number of independent sets in G equals the number of satisfying assignments of F. Furthermore the existence of an algorithm that solves #Imp2Sat in time $2^{o(n)}$ would imply the existence of an algorithm that

solves #BIS in time $2^{o(n)}$. Applying Theorem 3 we obtain that such an algorithm would refute #ETH. ◀

We sketch how to obtain the #ETH dichotomy theorem for Boolean CSPs.

Proof of Corollary 4. If every relation in Γ is affine then we can solve #CSP(Γ) in polynomial time using Gaussian elimination as in [6]. Otherwise, the problem is #P-hard by [6]. If, in addition, #ETH holds, #CSP(Γ) cannot be solved in time $2^{o(n)}$ as a subexponential algorithm could also be used to solve #Pos2Sat or #Imp2Sat (see Observation 11) in time $2^{o(n)}$ which is not possible assuming #ETH (by Theorem 5 and Lemma 12). ◀

Acknowledgments. We thank Radu Curticapean for various fruitful discussions and interactions, and also Leslie Ann Goldberg, Miki Hermann, Mark Jerrum, John Lapinskas, David Richerby, and all other participants of the "dichotomies" work group at the Simons Institute in the spring of 2016. Moreover, we thank Tyson Williams who in 2013 pointed the second author to [11], and Thore Husfeldt for encouraging us to pursue the publication of this manuscript.

References

1 Andreas Björklund. Counting perfect matchings as fast as Ryser. In *Proceedings of the 23rd Symposium on Discrete Algorithms, SODA 2012*, pages 914–921, 2012. doi:10.1137/1.9781611973099.73.

2 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Computing the Tutte polynomial in vertex-exponential time. In *Proceedings of the 47th annual IEEE Symposium on Foundations of Computer Science, FOCS 2008*, pages 677–686, 2008. doi:10.1109/FOCS.2008.40.

3 Jin-Yi Cai and Xi Chen. Complexity of counting CSP with complex weights. In *Proceedings of the 44th Symposium on Theory of Computing, STOC 2012*, pages 909–920, 2012. doi:10.1137/100814585.

4 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. Computational complexity of holant problems. *SIAM Journal on Computing*, 40(4):1101–1132, 2011. doi:10.1137/100814585.

5 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *Proceedings of the 21st Annual IEEE Conference on Computational Complexity*, CCC'06, pages 252–260, Washington, DC, USA, 2006. IEEE Computer Society. doi:10.1109/CCC.2006.6.

6 Nadia Creignou and Miki Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125(1):1–12, 1996. doi:10.1006/inco.1996.0016.

7 Radu Curticapean. Block interpolation: A framework for tight exponential-time counting complexity. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming, ICALP 2015*, pages 380–392. Springer, 2015. doi:10.1007/978-3-662-47672-7_31.

8 Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlén. Exponential time complexity of the permanent and the Tutte polynomial. *ACM Transactions on Algorithms*, 10, 2014. doi:10.1145/2635812.

9 Holger Dell, Thore Husfeldt, and Martin Wahlén. Exponential time complexity of the permanent and the Tutte polynomial. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming, ICALP 2010*, pages 426–437, 2010. doi:10.1007/978-3-642-14165-2_37.

10 Joanna Ellis-Monaghan and Iain Moffatt. *CRC handbook on the Tutte polynomial and related topics*. CRC press, in preparation.

11 Geoffrey Grimmett and Colin McDiarmid, editors. *Combinatorics, Complexity, and Chance: A Tribute to Dominic Welsh*. Oxford Lecture Series in Mathematics and Its Applications (Book 34). Oxford University Press, 2007.

12 Thore Husfeldt and Nina Taslaman. The exponential time complexity of computing the probability that a graph is connected. In *Proceedings of the 5th International Symposium on Parameterized and Exact Computation, IPEC 2010*, pages 192–203, 2010. doi:10.1007/978-3-642-17493-3_19.

13 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

14 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

15 François Jaeger, Dirk L. Vertigan, and Dominic J. A. Welsh. On the computational complexity of the Jones and Tutte polynomials. *Mathematical proceedings of the Cambridge Philosophical Society*, 108(1):35–53, 1990. doi:10.1017/S0305004100068936.

16 J. Scott Provan and Michael O Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983. doi:10.1137/0212053.

17 Alan D. Sokal. The multivariate Tutte polynomial (alias Potts model) for graphs and matroids. In *Surveys in Combinatorics*, volume 327 of *London Mathematical Society Lecture Note Series*, pages 173–226, 2005.

18 Nina Sofia Taslaman. *Exponential-Time Algorithms and Complexity of NP-Hard Graph Problems*. PhD thesis, IT-Universitetet i København, 2013.

19 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991. doi:10.1137/0220053.

20 Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

21 Dirk Llewellyn Vertigan. *On the computational complexity of Tutte, Jones, Homfly and Kauffman invariants*. PhD thesis, University of Oxford, 1991.

A Parameterized Algorithmics Framework for Degree Sequence Completion Problems in Directed Graphs*

Robert Bredereck[1], Vincent Froese[2], Marcel Koseler[3], Marcelo Garlet Millani[4], André Nichterlein[†5], and Rolf Niedermeier[6]

1 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
robert.bredereck@tu-berlin.de
2 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
vincent.froese@tu-berlin.de
3 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
marcel.koseler@campus.tu-berlin.de
4 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
marcelo.garletmillani@campus.tu-berlin.de
5 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
andre.nichterlein@tu-berlin.de
6 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
rolf.niedermeier@tu-berlin.de

Abstract

There has been intensive work on the parameterized complexity of the typically NP-hard task to edit undirected graphs into graphs fulfilling certain given vertex degree constraints. In this work, we lift the investigations to the case of directed graphs; herein, we focus on arc insertions. To this end, our general two-stage framework consists of efficiently solving a problem-specific number problem transferring its solution to a solution for the graph problem by applying flow computations. In this way, we obtain fixed-parameter tractability and polynomial kernelizability results, with the central parameter being the maximum vertex in- or outdegree of the output digraph. Although there are certain similarities with the much better studied undirected case, the flow computation used in the directed case seems not to work for the undirected case while f-factor computations as used in the undirected case seem not to work for the directed case.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases NP-hard graph problem, graph realizability, graph modification, arc insertion, fixed-parameter tractability, kernelization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.10

1 Introduction

Modeling real-world networks (e.g., communication, ecological, social) often requests *directed* graphs (digraphs for short). We study a class of specific "network design" (in the sense of

* A full version of the paper is available at https://arxiv.org/abs/1604.06302.
† From Feb. 2016 to Jan. 2017 on postdoctoral leave to Durham University (GB), funded by DAAD.

11th International Symposium on Parameterized and Exact Computation (IPEC 2016).
Editors: Jiong Guo and Danny Hermelin; Article No. 10; pp. 10:1–10:14

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

constructing a specific network topology) or "graph realization" problems. Here, our focus is on inserting arcs into a given digraph in order to fulfill certain vertex degree constraints. These problems are typically NP-hard, so we choose parameterized algorithm design for identifying relevant tractable special cases. The main parameter we work with is the maximum in- or outdegree of the newly constructed digraph. To motivate the problems we deal with, consider the following three application scenarios.

1. Assume we are given a directed network representing a system's current state. Then, each individual node might have certain desired states of connectivity in terms of the numbers of in- and outgoing arcs which we want to satisfy by inserting arcs between the nodes. For instance, in a peer-review network we have an arc from one author reviewing a paper of another author. Depending on research experience, the authors might have different requests with respect to the number of own papers to be reviewed by others and other papers which they are reviewing. This leads to the Digraph Degree Constraint Completion problem as studied in Section 4.1.
2. Assume that we have two different data sources: A network which is an incomplete measurement of some unreliable source and the true degree sequence of the target network. The goal is to reconstruct the original network by inserting arcs such that we obtain the target degree sequence (in a sense, the network matches the given degree sequence). In the presence of labeled input networks this might for example reveal communication patterns between users in social networks. The corresponding problem is called Digraph Degree Sequence Completion and studied in Section 4.2.
3. Assume we want to "k-anonymize" a social network, that is, after inserting a minimum number of arcs each degree, that is, each combination of in- and outdegree, occurs either zero or at least k times. This leads to the Digraph Degree Anonymity problem as studied in Section 4.3.

All three problems are NP-hard. Based on a general framework presented in Section 3, we derive several fixed-parameter tractability results for them, mainly exploiting the parameter "maximum vertex degree" in the output digraph. Moreover, the three problems above are special cases of the Digraph Degree Constraint Sequence Completion problem which we will define next.

These three problems as well as Digraph Degree Constraint Sequence Completion are special graph modification problems: given a graph, can it be changed by a minimum number of graph modifications such that the resulting graph adheres to specific constraints for its degree sequence?

In the most basic variant a degree sequence is a sequence of positive integers specifying (requested) vertex degrees for a fixed ordering of the vertices. Typically, the corresponding computational problems are NP-hard. In recent years, research in this direction focused on undirected graphs [8, 10, 11, 14, 18, 20]. In this work, we investigate parameterized algorithms on digraphs. As Gutin and Yeo [12] observed, much less is known about the structure of digraphs than that of undirected graphs making the design of parameterized algorithms for digraphs more challenging. In particular, we present a general framework for a class of degree sequence modification problems, focusing on the case of arc insertions (that is, completion problems).

The most general degree completion problem for digraphs we consider in this work is as follows.

Digraph Degree Constraint Sequence Completion (DDConSeqC)

Input: A digraph $D = (V, A)$, a non-negative integer s, a "degree list function" $\tau\colon V \to 2^{\{0,\dots,r\}^2}$, and a "sequence property" Π.

Question: Is it possible to obtain a digraph D' by inserting at most s arcs in D such that the degree sequence of D' fulfills Π and $\deg_{D'}(v) \in \tau(v)$ for all $v \in V$?

We emphasize that there are two types of constraints – one (specified by the function τ) for the individual vertices and one (specified by Π) for the whole list of degree tuples. For instance, a common Π as occurring in the context of data privacy applications is to request that the list is *k-anonymous*, that is, every combination of in- and outdegree that occurs in the list occurs at least k times (see the third motivating example before).

Since DDConSeqC and its special cases as studied here all turn out to be NP-hard [19, 15], a parameterized complexity analysis seems the most natural fit for understanding the computational complexity landscape of these kinds of problems – this has also been observed in the above mentioned studies for the undirected case. Our main findings are mostly on the positive side. That is, although seemingly more intricate to deal with due to the existence of in- and outdegrees, many positive algorithmic results which hold for undirected graphs can also be achieved for digraphs (albeit using different techniques). In particular, we present a maximum-flow-based framework that, together with the identification and solution of certain number problems, helps to derive several fixed-parameter tractability results with respect to the parameter maximum possible in- or outdegree Δ^* in any solution digraph. Notably, the corresponding result in the undirected case was based on f-factor computations [8] which do not transfer to the directed case, and, vice versa, the flow computation approach we present for the directed case seemingly does not transfer to the undirected case. For special cases of DDConSeqC, we can move further and even derive some polynomial-size problem kernels, again for the parameter Δ^*.

We consider the parameter Δ^* for the following reasons. First, it is always at most r, a natural parameter in the input. Second, in combination with Π, we might get an even smaller upper bound for Δ^*. Third, bounded-degree graphs are well studied and our work extends this since we only require Δ^* to be small, not to be constant.

Related Work. Most of the work on graph modification problems for realizing degree constraints has focused on undirected graphs [8, 10, 11, 14, 18, 20]. Closest to our work is the framework for deriving polynomial-size problem kernels for undirected degree sequence completion problems [8], which we complement by our results for digraphs. Generally, we can derive similar results, but the technical details differ and the landscape of problems is richer in the directed case. As to digraph modification problems in general, we are aware of surprisingly little work. We mention work studying arc insertion for making a digraph transitive [22] or for making a graph Eulerian [7], both employing the toolbox of parameterized complexity analysis. Somewhat related is also work about the insertion of edges into a mixed graph to satisfy local edge-connectivity constraints [1] or about orienting edges in a partially oriented graph to make it an oriented graph [2].

Our Results. In Section 3, we present our general framework for DDConSeqC. That is, based on flow computations, in a two-stage approach we show that it is fixed-parameter tractable with respect to the parameter Δ^*. To this end, we identify a specific pure number problem that needs to be fixed-parameter tractable with respect to the largest integer in the input. Next, presenting applications of the framework, in Section 4.1, we show that if there is no constraint Π concerning the degree sequence (that is, Digraph Degree Constraint

Completion), then we not only obtain fixed-parameter tractability but also a polynomial-size problem kernel for parameter Δ^* can be obtained. Then, in Section 4.2 we show an analogous result if there is one exactly specified degree sequence to be realized (Digraph Degree Sequence Completion). Finally, in Section 4.3, we show that if we request the degree sequence to be k-anonymous (that is, Digraph Degree Anonymity), then we can at least derive a polynomial-size problem kernel for the combined parameter (s, Δ_D), where Δ_D denotes the maximum in- or outdegree of the input digraph D. Also, we take a first step outlining the limitations of our framework for digraphs. In contrast to the undirected case (which is polynomial-time solvable [17]), the corresponding number problem of Digraph Degree Anonymity surprisingly is weakly NP-hard and presumably not polynomial-time solvable. Due to lack of space, several proofs are deferred to a full version (available at `https://arxiv.org/abs/1604.06302`).

2 Preliminaries

We consider *digraphs* (without multiarcs or loops) $D = (V, A)$ with $n := |V|$ and $m := |A|$. For a vertex $v \in V$, $\deg_D^-(v)$ denotes the *indegree* of v, that is, the number of incoming arcs of v. Correspondingly, $\deg_D^+(v)$ denotes the *outdegree*, that is, the number of outgoing arcs of v. We define the *degree* $\deg_D(v) := (\deg_D^-(v), \deg_D^+(v))$. The set $V(A') := \{v \in V \mid ((v,w) \in A' \vee (w,v) \in A') \wedge w \in V\}$ contains all vertices incident to an arc in $A' \subseteq V^2$. For a set of arcs $A' \subseteq V^2$, $D + A'$ denotes the digraph $(V, A \cup A')$, while $D[A']$ denotes the subdigraph $(V(A'), A')$. Analogously, for a set of vertices $V' \subseteq V$, $D[V']$ denotes the induced subdigraph $(V', A \cap (V')^2)$ which only contains the vertices V' and the arcs between vertices from V'. The set $N_D^+(v) := \{w \in V \mid (v,w) \in A\}$ denotes the set of *outneighbors* of v. Analogously, $N_D^-(v) := \{w \in V \mid (w,v) \in A\}$ denotes the set of *inneighbors*. Furthermore, we define the maximum indegree $\Delta_D^- := \max_{v \in V} \deg_D^-(v)$, the maximum outdegree $\Delta_D^+ := \max_{v \in V} \deg_D^+(v)$, and $\Delta_D := \max\{\Delta_D^+, \Delta_D^-\}$.

A *digraph degree sequence* $\sigma = \{(d_1^-, d_1^+), \ldots, (d_n^-, d_n^+)\}$ is a multiset of nonnegative integer tuples, where $d_i^-, d_i^+ \in \{0, \ldots, n-1\}$ for all $i \in \{1, \ldots, n\}$. We define

$$\Delta_\sigma^- := \max\{d_1^-, \ldots, d_n^-\}, \quad \Delta_\sigma^+ := \max\{d_1^+, \ldots, d_n^+\}, \text{ and } \quad \Delta_\sigma := \max\{\Delta_\sigma^-, \Delta_\sigma^+\}.$$

For a digraph $D = (\{v_1, \ldots, v_n\}, A)$ we denote by $\sigma(D) := \{\deg_D(v_1), \ldots, \deg_D(v_n)\}$, the digraph degree sequence of D. Let $d = (d^-, d^+)$ be a nonnegative integer tuple. For a digraph D, the *block* $B_D(d)$ *of degree* d is the set of all vertices having degree d, formally $B_D(d) := \{v \in V \mid \deg_D(v) = d\}$. We define $\lambda_D(d)$ as the number of vertices in D with degree d, that is, $\lambda_D(d) := |B_D(d)|$. Similarly, we define $B_\sigma(t)$ as the multiset of all tuples equal to t and $\lambda_\sigma(t)$ as the number of occurrences of the tuple t in the multiset σ. For two integer tuples (x_1, y_1), (x_2, y_2), we define the sum $(x_1, y_1) + (x_2, y_2) := (x_1 + x_2, y_1 + y_2)$.

3 The Framework

Our goal is to develop a framework for deriving fixed-parameter tractability for a general class of completion problems in directed graphs. To this end, recall our general setting for DDConSeqC which is as follows. We are given a digraph and want to insert at most s arcs such that the vertices satisfy certain degree constraints τ, and additionally, the degree sequence of the digraph fulfills a certain property Π. Formally, the sequence property Π is given as a function that maps a digraph degree sequence to 1 if the sequence fulfills the property and otherwise to 0. We restrict ourselves to properties where the corresponding

function can be encoded with only polynomially many bits in the number of vertices of the input digraph and can be decided efficiently.[1] We remark that it is not always the case that there are both vertex degree constraints (as defined by τ) and degree sequence constraints (as defined by Π) requested. This can be handled by either setting τ to the trivial degree list function with $\tau(v) = \{0, \ldots, n-1\}^2$ for all $v \in V$ or setting Π to allow all possible degree sequences.

In this section, we show how to derive (under certain conditions) fixed-parameter tractability with respect to the maximum possible in- or outdegree Δ^* of the *output* digraph for DDConSeqC. Note that Δ^* in general is not known in advance. In practice, we might therefore instead consider upper bounds for Δ^* which depend on the given input. For example, it always holds $\Delta^* \leq \min\{r, \Delta_D + s\}$ since we are only inserting at most s arcs in D. Clearly, Δ^* might also be upper-bounded depending on Π (or even depending on r, s, Δ_D, and Π) in some cases. Our generic framework consists of two main steps: First, we prove fixed-parameter tractability with respect to the combined parameter (s, Δ_D) in Section 3.1. This step generalizes ideas for the undirected case [8]. Note that $\Delta_D \leq \Delta^*$ trivially holds. Second, we show in Section 3.2 how to upper-bound the number s of arc insertions polynomially in Δ^* by solving a certain problem specific numerical problem. For this step, we develop a new key argument based on a maximum flow computation (the undirected case was based on f-factor arguments).

3.1 Fixed-parameter tractability with respect to (s, Δ_D)

We show that DDConSeqC is fixed-parameter tractable with respect to the combination of the maximum number s of arcs to insert and the maximum in- or outdegree Δ_D of the input digraph D. The basic idea underlying this result is that two vertices v and w with $\deg_D(v) = \deg_D(w)$ and $\tau(v) = \tau(w)$ are interchangeable. Accordingly, we will show that it suffices to consider only a bounded number of vertices with the same "degree properties". In particular, if there is a solution, then there is also a solution that only inserts arcs between a properly chosen subset of vertices of bounded size. To formalize this idea, we introduce the notion of an *α-block-type set* for some positive integer α.

To start with, we define the types of a vertex via the numbers of arcs that τ allows to add to this vertex. Let (D, s, τ, Π) be a DDConSeqC instance. A vertex v is of *type* $t \in \{0, \ldots, \Delta^*\}^2$ if $\deg_D(v) + t \in \tau(v)$. Observe that one vertex can be of several types. The subset of $V(D)$ containing all vertices of type t is denoted by $T_{D,\tau}(t)$. A vertex v of type $(0,0)$ (that is, $\deg_D(v) \in \tau(v)$) is called *satisfied*. A vertex which is not satisfied is called *unsatisfied*. We next define our notion of α-block-type sets and its variants.

▶ **Definition 1.** Let α be a positive integer and let $U \subseteq V(D)$ denote the set of all unsatisfied vertices in D. A vertex subset $C \subseteq V(D)$ with $U \subseteq C$ is called

- *α-type set* if, for each type $t \neq (0,0)$, C contains exactly $\min\{|T_{D,\tau}(t) \setminus U|, \alpha\}$ satisfied vertices of type t;
- *α-block set* if, for each degree $d \in \sigma(D)$, C contains exactly $\min\{|B_D(d) \setminus U|, \alpha\}$ satisfied vertices with degree d;
- *α-block-type set* if, for each degree $d \in \sigma(D)$ and each type $t \neq (0,0)$, C contains exactly $\min\{|(B_D(d) \cap T_{D,\tau}(t)) \setminus U|, \alpha\}$ satisfied vertices of degree d and type t.

[1] All specific properties in this work can be easily decided in polynomial time. Indeed, in many cases even fixed-parameter tractability with respect to the maximum integer in the sequence would suffice.

As a first step, we prove that these sets defined above can be computed efficiently.

▶ **Lemma 2.** *An α-type/α-block/α-block-type set C as described in Definition 1 can be computed in $O(m + |\tau| + r^2)$ / $O(m + n + \Delta_D^2)$ / $O(m + |\tau| + \Delta_D^2 r^2)$ time.*

We move on to the crucial lemma stating that a solution (that is, a set of arcs), if existing, can always be found in between vertices of an α-block-type set C given that C contains "enough" vertices of each degree and type. Here, enough means $\alpha := 2s(\Delta_D + 1)$.

▶ **Lemma 3.** *Let (D, s, τ, Π) be a* DDConSeqC *instance and let $C \subseteq V(D)$ be a $2s(\Delta_D + 1)$-block-type set. If (D, s, τ, Π) is a yes-instance, then there exists a solution $A^* \subseteq C^2$ for (D, s, τ, Π), that is, $|A^*| \leq s$, $\sigma(D + A^*)$ fulfills Π, and $\deg_{D+A^*}(v) \in \tau(v)$ for all $v \in V(D)$.*

If there are no restrictions on the resulting degree sequence (as it is the case for the Digraph Degree Constraint Completion problem (DDConC) in Section 4.1), then we can replace the $2s(\Delta_D + 1)$-block-type set in Lemma 3 by a $2s(\Delta_D + 1)$-type set:

▶ **Lemma 4.** *Let (D, s, τ) be a* DDConC *instance and let $C \subseteq V(D)$ be a $2s(\Delta_D + 1)$-type set. If (D, s, τ) is a yes-instance, then there exists a solution $A^* \subseteq C^2$ for (D, s, τ), that is, $|A^*| \leq s$ and $\deg_{D+A^*}(v) \in \tau(v)$ for all $v \in V(D)$.*

Similarly, if there are no restrictions on the individual vertex degrees, that is, τ is the degree list function $\tau(v) = \{0, \ldots, n-1\}^2$ for all $v \in V(D)$, then we can replace the $2s(\Delta_D + 1)$-block-type set by a $2s(\Delta_D + 1)$-block set.

▶ **Lemma 5.** *Let (D, s, τ, Π) be a* DDConSeqC *instance where $\tau(v) = \{0, \ldots, n-1\}^2$ for all $v \in V(D)$ and let $C \subseteq V(D)$ be a $2s(\Delta_D + 1)$-block set. If (D, s, τ, Π) is a yes-instance, then there exists a solution $A^* \subseteq C^2$ for (D, s, τ, Π), that is, $|A^*| \leq s$ and $\sigma(D + A^*)$ fulfills Π.*

Lemma 3 implies a fixed-parameter algorithm by providing a reduced search space for possible solutions, namely any $2s(\Delta_D + 1)$-block-type set C: Simply try out all possibilities to insert at most s arcs with endpoints in C and check whether in one of the cases the degrees and the degree sequence of the resulting graph satisfy the requirements τ and Π. As $|C| \leq 2s(\Delta_D + 1) \cdot (\Delta_D + 1)^2 (\Delta^*)^2$ and $\Delta^* \leq \Delta_D + s$, there are at most $O(2^{(2s(\Delta_D+1)^3(\Delta_D+s)^2)^2})$ possible subsets of arcs to insert. Altogether, this leads to the following theorem.

▶ **Theorem 6.** *If deciding Π is fixed-parameter tractable with respect to the maximum integer in the sequence, then* DDConSeqC *is fixed-parameter tractable with respect to (s, Δ_D).*

3.2 Bounding the solution size s polynomially in Δ^*

This subsection constitutes the major part of our framework. The rough overall scheme is analogous to the undirected case as described by Froese et al. [8]. By dropping the graph structure and solving a simpler problem-specific number problem on the degree sequence of the input digraph, we show how to solve DDConSeqC instances with "large" solutions provided that we can solve the associated number problem efficiently. The number problem is defined so as to simulate the insertion of arcs to a digraph on an integer tuple sequence. Note that inserting an arc increases the indegree of a vertex by one and increases the outdegree of another vertex by one. Inserting s arcs can thus be represented by increasing the tuple entries in the degree sequence by an overall value of s in each component. Formally, the corresponding number problem (abbreviated as #DDConSeqC) is defined as follows.

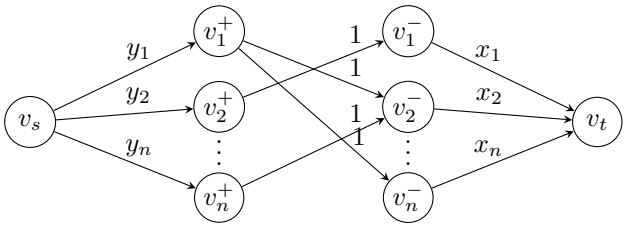


Figure 1 A flow network as described in Construction 8. For each vertex v_i in the digraph D there are two vertices v_i^+ and v_i^-. We connect a vertex v_i^+ to a vertex v_j^- if the arc (v_i, v_j) is not in D. Inserting the arc (v_i, v_j) is then represented by setting the flow on the arc (v_i^+, v_j^-) to one.

Numbers Only Digraph Degree Constraint Sequence Completion

Input: A sequence $\sigma = (c_1, d_1), \ldots, (c_n, d_n)$ of n nonnegative integer tuples, a positive integer s, a "tuple list function" $\tau\colon \{1, \ldots, n\} \to 2^{\{0,\ldots,r\}^2}$, and a sequence property Π.

Question: Is there a sequence $\sigma' = (c'_1, d'_1), \ldots, (c'_n, d'_n)$ such that $\sum_{i=1}^n c'_i - c_i = \sum_{i=1}^n d'_i - d_i = s$, $c_i \leq c'_i$, $d_i \leq d'_i$, and $(c'_i, d'_i) \in \tau(i)$ for all $1 \leq i \leq n$, and σ' fulfills Π?

If we plug the degree sequence of a digraph into #DDConSeqC, then an integer tuple (c'_i, d'_i) of a solution tells us to add $x_i := c'_i - c_i$ incoming arcs and $y_i := d'_i - d_i$ outgoing arcs to the vertex v_i. We call the tuples (x_i, y_i) *demands*. Having computed the demands, we can then try to solve our original DDConSeqC instance by searching for a set of arcs to insert that exactly fulfills the demands. Such an arc set, however, might not always exist. Hence, the remaining problem is to decide whether it is possible to realize the demands in the given digraph. The following lemma shows (using flow computations) that this is in fact always possible if the number s of arcs to insert is large compared to Δ^*.

▶ **Lemma 7.** *Let $D = (V = \{v_1, \ldots, v_n\}, A)$ be a digraph and let $x_1, \ldots, x_n$, $y_1, \ldots, y_n$, and Δ^* be nonnegative integers such that*

(I) $\Delta^* \leq n - 1$,
(II) $\deg_D^-(v_i) + x_i \leq \Delta^*$ *for all* $i \in \{1, \ldots, n\}$,
(III) $\deg_D^+(v_i) + y_i \leq \Delta^*$ *for all* $i \in \{1, \ldots, n\}$,
(IV) $\sum_{i=1}^n x_i = \sum_{i=1}^n y_i =: s$, *and*
(V) $s > 2(\Delta^*)^2$.

Then, there exists an arc set $A' \subseteq V^2 \setminus A$ of size s such that for the digraph $D' := D + A'$ it holds $\deg_{D'}(v_i) = \deg_D(v_i) + (x_i, y_i)$ for all $v_i \in V$. Moreover, the set A' can be computed in $O(n^3)$ time.

Proof. The proof is based on a flow network which we construct such that the corresponding maximum flow yields the set A' of arcs to be inserted in D in order to obtain our target digraph D'.

▶ **Construction 8.** *We build a flow network $N = (V_N, A_N)$ according to the following steps.*

- *Add a* source *vertex v_s and a* sink *vertex v_t to N;*
- *for each vertex $v_i \in V$, add two vertices v_i^+, v_i^- to N;*
- *for each $i \in \{1, \ldots, n\}$, insert the arc (v_s, v_i^+) with capacity y_i;*
- *for each $i \in \{1, \ldots, n\}$, insert the arc (v_i^-, v_t) with capacity x_i;*
- *for each $(v_i, v_j) \in V^2 \setminus A$ with $i \neq j$, insert the arc (v_i^+, v_j^-) with capacity one.*

The network N contains $|V_N| \in O(n)$ vertices and $|A_N| \in O(n^2 - m)$ arcs (since $m \leq n^2 - n$, we also have $|A_N| \in \Omega(n)$) and can be constructed in $O(n^2)$ time. See Figure 1

for an illustration. Inserting an arc (v_i, v_j) in D corresponds to sending flow from v_i^+ to v_j^-. Since, by definition, each vertex v_i^+ will only receive at most y_i flow from v_s and each vertex v_i^- will send at most x_i flow to v_t, we cannot insert more than s arcs (Condition (IV)).

We claim that for $s > 2(\Delta^*)^2$ (Condition (V)), the maximum flow in the network is indeed s. To see this, let $V_N^+ := \{v_i^+ \in V_N \mid i \in \{1, \dots, n\}\}$ and let $V_N^- := \{v_i^- \in V_N \mid i \in \{1, \dots, n\}\}$. In the following, a vertex $v_i^+ \in V_N^+$ ($v_j^- \in V_N^-$) is called *saturated* with respect to a flow $f\colon A_N \to \mathbb{R}^+$, if $f(v_s, v_i^+) = y_i$ ($f(v_j^-, v_t) = x_j$). Suppose that the maximum flow f has a value less than s. Then, there exist non-saturated vertices $v_i^+ \in V_N^+$ and $v_j^- \in V_N^-$. Let $X \subseteq V_N^-$ be the vertices to which v_i^+ has an outgoing arc in the residual graph and let $Y \subseteq V_N^+$ be the vertices which have an outgoing arc to v_j^- in the residual graph. Observe that $\deg_N^+(v_i^+) = n - 1 - \deg_D^+(v_i)$ and $\deg_N^-(v_j^-) = n - 1 - \deg_D^-(v_j)$. Consequently, $|X| > n - 1 - \deg_D^+(v_i) - y_i \geq n - 1 - \Delta^*$ holds due to Condition (III). Since v_i^+ is not saturated, we know that $|X| \geq n - \Delta^* \geq 1$ (due to Condition (I)). By the same reasoning (using Conditions (II) and (I)) it follows that $|Y| \geq n - \Delta^* \geq 1$.

Remember that f is a flow of maximum value. Hence, each vertex in X and each vertex in Y is saturated. Otherwise, there would be an augmenting path in the residual graph, contradicting our assumption of f being maximal. If a vertex $x \in X$ would receive flow from a vertex $y \in Y$, then this implies a backward arc in the residual graph resulting in an augmenting path $v_s \to v_i^+ \to x \to y \to v_j^- \to v_t$, again contradicting our maximality assumption for f. Thus, we can conclude that all the flow that goes into X has to come from the remaining vertices in $V_N^+ \setminus (Y \cup \{v_i^+\})$. This set has size at most $n - |Y| \leq n - (n - \Delta^*) = \Delta^*$. But since $y_\ell \leq \Delta^*$ for all $\ell \in \{1, \dots, n\}$ (by Condition (III)), those Δ^* vertices can cover at most a flow of value $(\Delta^*)^2$ and hence,

$$\sum_{v_i^- \in X} x_i \leq \sum_{v_i^+ \in V_N^+ \setminus (Y \cup \{v_i^+\})} y_i \leq (\Delta^*)^2. \tag{1}$$

Since X is saturated, and since also $x_\ell \leq \Delta^*$ holds for all $\ell \in \{1, \dots, n\}$ (Condition (II)), we obtain from Condition (IV)

$$\begin{aligned} s = \sum_{i=1}^{n} x_i &= \sum_{v_i^- \in X} x_i + \sum_{v_i^- \in V_N^- \setminus X} x_i \overset{(1)}{\leq} (\Delta^*)^2 + \sum_{v_i^- \in V_N^- \setminus X} \Delta^* \\ &= (\Delta^*)^2 + |V_N^- \setminus X| \cdot \Delta^* = (\Delta^*)^2 + (n - |X|) \cdot \Delta^* \\ &\leq (\Delta^*)^2 + \Delta^* \cdot \Delta^*. \end{aligned}$$

This contradicts $s > 2(\Delta^*)^2$ (Condition (V)) and hence proves the claim.

Now, let f be a maximum flow in N (computable in $O(|V_N||E_N|) = O(n(n^2 - m))$ time [21]) and let $A' := \{(v_i, v_j) \in V^2 \mid f((v_i^+, v_j^-)) = 1\}$ and note that $|A'| = s$ and $A' \cap A = \emptyset$. Clearly, for the digraph $D' := D + A'$ it holds $\deg_{D'}(v_i) = \deg_D(v_i) + (x_i, y_i)$ for all $v_i \in V$. ◀

We remark that similar flow-constructions as given in the proof above have been used before [9, 6]. The difference here is that we actually argue about the size of the flow and not only about polynomial-time solvability. Consequently, our proof uses different arguments.

With Lemma 7 we have the key which allows us to transfer solutions of #DDConSeqC to solutions of DDConSeqC. The following lemma is immediate.

▶ **Lemma 9.** *Let $I := (D = (V, A), s, \tau, \Pi)$ with $V = \{v_1, \dots, v_n\}$ be an instance of* DDConSeqC *with $s > 2(\Delta^*)^2$. If there exists an s' with $2(\Delta^*)^2 < s' \leq s$ such that $I' := (\deg_D(v_1), \dots, \deg_D(v_n), s', \tau', \Pi)$ with $\tau'(i) := \tau(v_i)$ for all $v_i \in V$ is a yes-instance of* #DDConSeqC*, then also I is a yes-instance of* DDConSeqC*.*

Figure 2 Two example instances of DDConC with $s = 1$. The left instance is solvable by inserting the (dashed) arc from the right vertex to the middle vertex. The right instance is a no-instance since one cannot add an outgoing arc to the left vertex or to the middle vertex but one has to add an incoming arc to the right vertex (loops are not allowed).

We now have all ingredients for our first main result, namely transferring fixed-parameter tractability with respect to the combined parameter (s, Δ^*) to fixed-parameter tractability with respect to the single parameter Δ^*, provided that #DDConSeqC is fixed-parameter tractable with respect to the largest possible integer ξ in the output sequence. The idea is to search for large solutions based on Lemma 9 using #DDConSeqC. If there are no large solutions (that is, $s \leq 2(\Delta^*)^2$), then we run an FPT-algorithm with respect to (s, Δ^*).

▶ **Theorem 10.** *If* DDConSeqC *is fixed-parameter tractable with respect to* (s, Δ^*) *and* #DDConSeqC *is fixed-parameter tractable with respect to the largest possible integer* ξ *in the output sequence, then* DDConSeqC *is fixed-parameter tractable with respect to* Δ^*.

Our second main result allows to transfer a polynomial-size problem kernel with respect to (s, Δ^*) to a polynomial-size problem kernel with respect to Δ^* if #DDConSeqC is polynomial-time solvable. The proof is analogous to the proof of Theorem 10.

▶ **Theorem 11.** *If* DDConSeqC *admits a problem kernel containing* $g(s, \Delta^*)$ *vertices computable in* $p(n)$ *time and* #DDConSeqC *is solvable in* $q(n)$ *time for polynomials* p *and* q*, then* DDConSeqC *admits a problem kernel with* $g(2(\Delta^*)^2, \Delta^*)$ *vertices computable in* $O(s \cdot q(n) + p(n))$ *time.*

4 Applications

In the following, we show how the framework described in Section 3 can be applied to three special cases of DDConSeqC. These special cases naturally extend known problems on undirected graphs to the digraph setting.

4.1 Digraph Degree Constraint Completion

In this section, we investigate the NP-hard special case of DDConSeqC[2] where the property Π allows any possible degree sequence, see Figure 2 for two illustrating examples.

Digraph Degree Constraint Completion (DDConC)

Input: A digraph $D = (V, A)$, a positive integer s, and a "degree list function" $\tau \colon V \to 2^{\{0,\dots,r\}^2}$.

Question: Is it possible to obtain a digraph D' by inserting at most s arcs in D such that $\deg_{D'}(v) \in \tau(v)$ for all $v \in V$?

DDConC is the directed (completion) version of the well-studied undirected Degree Constraint Editing problem [18, 10] for which problem kernel with $O(r^5)$-vertices is

[2] This special case was investigated more specifically in the Bachelor thesis of Koseler [15] (online available).

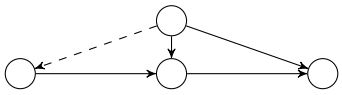


Figure 3 Example instance of DDSeqC. Inserting the dashed arc in the input digraph (solid arcs) with degree sequence $\{(0,1),(0,2),(2,0),(2,1)\}$ yields a digraph with the given target sequence σ.

known [8]. We subsequently transfer the polynomial-size problem kernel for the undirected case to a polynomial-size problem kernel for DDConC with respect to Δ^*. Note that the parameter Δ^* is clearly at most r. Since it is trivial to decide Π in this case, we obtain fixed-parameter tractability of DDConC with respect to (s, Δ_D) due to Theorem 6, which is based on a bounded search space, namely a $2s(\Delta_D + 1)$-type set (see Definition 1 and Lemma 4). We further strengthen this result by removing all vertices that are not in the $2s(\Delta_D + 1)$-type set and adjusting the degree list function τ appropriately. Lemma 4 then yields the correctness of this approach resulting in a polynomial-size problem kernel with respect to (s, Δ^*).

▶ **Theorem 12.** DDConC *admits a problem kernel containing* $O(s(\Delta^*)^3) \subseteq O(sr^3)$ *vertices. It is computable in* $O(m + |\tau| + r^2)$ *time.*

The goal now is to use our framework (Theorem 11) to transfer the polynomial-size kernel with respect to (s, Δ^*) to a polynomial-size kernel with respect to Δ^*. To this end, we show that the corresponding number problem (#DDConC) is polynomial-time solvable. Here, #DDConC is the special case of #DDConSeqC without the sequence property Π. #DDConC can be solved in pseudo-polynomial time by a dynamic programming algorithm. Note that pseudo-polynomial time is sufficient for our purposes since all occurring numbers will be bounded by $O(n^2)$ when creating the #DDConC instance from the given DDConC instance. (In fact, we conjecture that #DDConC is weakly NP-hard and a reduction from Partition should be possible as in the case for #DDA in Section 4.3, Theorem 19.)

▶ **Lemma 13.** #DDConC *is solvable in* $O(n(sr)^2)$ *time.*

Combining Theorem 12 and Lemma 13 yields the following corollary of Theorem 11.

▶ **Corollary 14.** DDConC *admits a problem kernel containing* $O((\Delta^*)^5) \subseteq O(r^5)$ *vertices. It is computable in* $O(m + ns^3r^2)$ *time.*

4.2 Digraph Degree Sequence Completion

In this section, we investigate the NP-hard special case of DDConSeqC[3] where τ does not restrict the allowed degree of any vertex and Π is fulfilled by exactly one specific degree sequence σ (see Figure 3 for an example). The undirected problem variant is studied by Golovach and Mertzios [11].

Digraph Degree Sequence Completion (DDSeqC)

Input: A digraph $D = (V, A)$, a digraph degree sequence σ containing $|V|$ integer tuples.

Question: Is it possible to obtain a digraph D' by inserting arcs in D such that $\sigma(D') = \sigma$?

[3] Although not stated explicitly, the NP-hardness follows from the proof of Theorem 3.2 of the Bachelor thesis of Millani [19] (online available) as the construction therein allows for only one feasible target degree sequence.

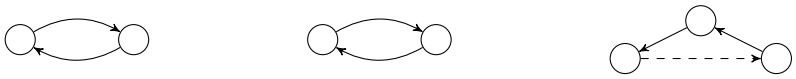


Figure 4 Example instance of DDA. The input digraph with three components (solid arcs) is 1-anonymous since there is only one vertex with degree $(0, 1)$. By inserting the dashed arc, the digraph becomes 7-anonymous since all vertices have degree $(1, 1)$.

For DDSeqC, the parameter Δ^* is by definition equal to Δ_σ. Moreover, note that the number s of arcs to insert (if possible) is determined by the target sequence σ by $s := \sum_{(c,d)\in\sigma} c - \sum_{v\in V(D)} \deg^-_D(v)$. We henceforth assume that

$$s = \sum_{(c,d)\in\sigma} c - \sum_{v\in V(D)} \deg^-_D(v) = \sum_{(c,d)\in\sigma} d - \sum_{v\in V(D)} \deg^+_D(v) \geq 0$$

holds since otherwise we have a trivial no-instance.

Since deciding Π (that is, deciding whether $\sigma(D') = \sigma$) can be done in polynomial time, we immediately obtain fixed-parameter tractability of DDSeqC with respect to (s, Δ_D) due to Theorem 6. We further strengthen this result by developing a polynomial-size problem kernel for DDSeqC with respect to (s, Δ_σ). The kernelization is inspired by the $O(s\Delta_\sigma^2)$-vertex problem kernel for the undirected problem by Golovach and Mertzios [11]. The main idea is to only keep the vertices of a $2s(\Delta_D + 1)$-block set (see Definition 1) together with some additional "dummy" vertices and to adjust the digraph degree sequence σ properly.

▶ **Theorem 15.** DDSeqC *admits a problem kernel containing* $O(s\Delta_\sigma^3)$ *vertices computable in* $O(n + m + \Delta_\sigma^2)$ *time.*

The corresponding number problem #DDSeqC is the special case of #DDConSeqC asking for the specific target sequence σ. #DDSeqC can be solved in polynomial time by finding perfect matchings in an auxiliary graph.

▶ **Lemma 16.** #DDSeqC *is solvable in* $O(n^{2.5})$ *time.*

Combining Theorem 15 and Lemma 16 yields the following corollary of Theorem 11.

▶ **Corollary 17.** DDSeqC *admits a problem kernel containing* $O(\Delta_\sigma^5)$ *vertices. It is computable in* $O(sn^{2.5})$*-time.*

4.3 Degree Anonymity

We extend the definition of Degree Anonymity in undirected graphs due to Liu and Terzi [17] to digraphs and obtain the following NP-hard problem [19] (Figure 4 presents an example):

Digraph Degree Anonymity (DDA)

Input: A digraph $D = (V, A)$ and two positive integers k and s.

Question: Is it possible to obtain a digraph D' by inserting at most s arcs in D such that D' is k-anonymous, that is, for every vertex $v \in V$ there are at least $k - 1$ other vertices in D' with degree $\deg_{D'}(v)$?

The (parameterized) complexity as well as the (in-)approximability of the undirected version called Degree Anonymity are well-studied [5, 14, 3]. There also exist many heuristic approaches to solve the undirected version [4, 13]. Notably, our generic approach shown

in Section 3.2 originates from a heuristic of Liu and Terzi [17] for Degree Anonymity. Later, Hartung et al. [14] used this heuristic to prove that "large" solutions of Degree Anonymity can be found in polynomial time and Froese et al. [8] extended this approach to a more general class of problems. The property Π (that is, k-anonymity) can clearly be checked for a given input digraph degree sequence in polynomial time. Hence, Theorem 6 yields fixed-parameter tractability of DDA with respect to (s, Δ_D). Again, we develop a polynomial-size problem kernel with respect to (s, Δ_D). Somewhat surprisingly, we cannot transfer this problem kernel to a problem kernel with respect to Δ^* since we are not able to solve the corresponding number problem in polynomial time. In fact, we will show that it is at least weakly NP-hard.

We first provide a problem kernel based on Lemma 5 in a similar fashion as in the proof of Theorem 15: We keep a $2s(\Delta_D + 1)$-block set C in the kernel and remove all other vertices. In order to not change the degrees of the vertices we kept, we introduce "dummy" vertices that will have a very high degree so that there is no interference with the vertices we kept. The approach is inspired by the polynomial-size problem kernel of Hartung et al. [14].

▶ **Theorem 18.** DDA *admits a problem kernel containing* $O(\Delta_D^5 s)$ *vertices. It is computable in* $O(\Delta_D^{10} s^2 + \Delta_D^3 sn)$ *time.*

In contrast to both number problems in Sections 4.1 and 4.2, we were unable to find a polynomial-time algorithm for the number problem for DDA, which is the special case of #DDConSeqC asking for a k-anonymous target sequence. We can show that #DDA is weakly NP-hard by a polynomial-time many-one reduction from Partition.

▶ **Theorem 19.** #DDA *is (weakly) NP-hard even if* $k = 2$.

Note that the hardness from Theorem 19 does not translate to instances of #DDA originating from digraph degree sequences because in such instances all numbers in the input sequence σ and also in the output sequence σ' are bounded by $n - 1$ where n is the number of tuples in σ. Since there are pseudo-polynomial-time algorithms for Partition, Theorem 19 leaves open whether #DDA is strongly NP-hard or can be solved in polynomial time for instances originating from digraphs.

To again apply our framework (Theorem 10), we show that #DDA is at least fixed-parameter tractable with respect to the largest possible integer ξ in the output sequence. To this end, we develop an integer linear program that contains at most $O(\xi^4)$ integer variables and apply the a famous result due to Lenstra [16].

▶ **Theorem 20.** #DDA *is fixed-parameter tractable with respect to the largest possible integer* ξ *in the output sequence.*

Combining Theorems 6, 10, and 20 yields fixed-parameter tractability for DDA with respect to Δ^*. Hartung et al. [14] showed fixed-parameter tractability with respect to Δ_G in the undirected setting. This result was based on showing that $\Delta^* \leq \Delta_G^2 + 5\Delta_G^2 + 2$. In the directed setting, however, we can only show that $\Delta^* \leq 4k(\Delta_D + 2)^2$.

▶ **Lemma 21.** *Let* D *be a digraph and let* S *be a minimum size arc set such that* $D + S$ *is* k*-anonymous. Then the maximum degree in* $D + S$ *is at most* $4k(\Delta_D + 2)^2 + \Delta_D$.

Consequently, combining Theorems 6, 10, 20, and Lemma 21, we obtain the following.

▶ **Corollary 22.** DDA *is fixed-parameter tractable with respect to* Δ^* *and* (k, Δ_D).

It remains open whether DDA is fixed-parameter tractable with respect to Δ_D. We remark that the problems DDConC and DDSeqC are both NP-hard for $\Delta_D = 3$. This follows from an adaption of the construction given by Millani [19, Theorem 3.2].

5 Conclusion

We proposed a general framework for digraph degree sequence completion problems and demonstrated its wider applicability in case studies. Somewhat surprisingly, the presumably more technical case of digraphs allowed for some elegant tricks (based on flow computations) that seem not to work for the presumably simpler undirected case. Once having established the framework (see Section 3), the challenges then associated with deriving fixed-parameter tractability and kernelizability results usually boil down to the question for fixed-parameter tractability and (pseudo-)polynomial-time solvability of a simpler problem-specific number problem. While in most cases we could develop polynomial-time algorithms solving these number problems, in the case of Digraph Degree Anonymity the polynomial-time solvability of the associated number problem remains open. Moreover, a widely open field is to attack weighted versions of our problems. Finally, we believe that due to the fact that many real-world networks are inherently directed (e.g., representing relations such as "follower", "likes", or "cites") further studies (e.g., exploiting special digraph properties) of digraph degree sequence completion problems are desirable.

References

1 Jørgen Bang-Jensen, András Frank, and Bill Jackson. Preserving and increasing local edge-connectivity in mixed graphs. *SIAM Journal on Discrete Mathematics*, 8(2):155–178, 1995.

2 Jørgen Bang-Jensen, Jing Huang, and Xuding Zhu. Completing orientations of partially oriented graphs. *CoRR abs/1509.01301*, 2015.

3 Cristina Bazgan, Robert Bredereck, Sepp Hartung, André Nichterlein, and Gerhard J. Woeginger. Finding large degree-anonymous subgraphs is hard. *Theoretical Computer Science*, 622:90–110, 2016.

4 Jordi Casas-Roma, Jordi Herrera-Joancomartí, and Vicenç Torra. An algorithm for k-degree anonymity on large networks. In *Proceedings of the International Conference on Advances in Social Networks Analysis and Mining (ASONAM'13)*, pages 671–675. ACM, 2013.

5 Sean Chester, Bruce Kapron, Gautam Srivastava, and S. Venkatesh. Complexity of social network anonymization. *Social Network Analysis and Mining*, 3(2):151–166, 2013.

6 Marek Cygan, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Ildikó Schlotter. Parameterized complexity of eulerian deletion problems. *Algorithmica*, 68(1):41–61, 2014.

7 Frederic Dorn, Hannes Moser, Rolf Niedermeier, and Mathias Weller. Efficient algorithms for eulerian extension and rural postman. *SIAM Journal on Discrete Mathematics*, 27(1):75–94, 2013.

8 Vincent Froese, André Nichterlein, and Rolf Niedermeier. Win-win kernelization for degree sequence completion problems. *Journal of Computer and System Sciences*, 82(6):1100–1111, 2016.

9 D. Gale. A theorem on flows in networks. *Pacific Journal of Mathematics*, 7:1073–1082, 1957.

10 Petr A. Golovach. Editing to a graph of given degrees. *Theoretical Computer Science*, 591:72–84, 2015.

11 Petr A. Golovach and George B. Mertzios. Graph editing to a given degree sequence. In *Proceedings of the 11th International Computer Science Symposium in Russia (CSR'16)*, volume 9691 of *LNCS*, pages 177–191. Springer, 2016.

12 Gregory Gutin and Anders Yeo. Some parameterized problems on digraphs. *Computer Journal*, 51(3):363–371, 2008.

13 Sepp Hartung, Clemens Hoffmann, and André Nichterlein. Improved upper and lower bound heuristics for degree anonymization in social networks. In *Proceedings of the 13th International Symposium on Experimental Algorithms (SEA'14)*, volume 8504 of *LNCS*, pages 376–387. Springer, 2014.

14 Sepp Hartung, André Nichterlein, Rolf Niedermeier, and Ondřej Suchý. A refined complexity analysis of degree anonymization in graphs. *Information and Computation*, 243:249–262, 2015.

15 Marcel Koseler. Kernelization for degree-constraint editing on directed graphs. Bachelor thesis, TU Berlin, November 2015. URL: `http://fpt.akt.tu-berlin.de/publications/theses/BA-marcel-koseler.pdf`.

16 Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.

17 Kun Liu and Evimaria Terzi. Towards identity anonymization on graphs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD'08, pages 93–106. ACM, 2008.

18 Luke Mathieson and Stefan Szeider. Editing graphs to satisfy degree constraints: A parameterized approach. *Journal of Computer and System Sciences*, 78(1):179–191, 2012.

19 Marcelo Garlet Millani. Algorithms and complexity for degree anonymization in directed graphs. Bachelor thesis, TU Berlin, March 2015. URL: `http://fpt.akt.tu-berlin.de/publications/theses/BA-marcelo-millani.pdf`.

20 Hannes Moser and Dimitrios M. Thilikos. Parameterized complexity of finding regular induced subgraphs. *Journal of Discrete Algorithms*, 7(2):181–190, 2009.

21 James B. Orlin. Max flows in $o(nm)$ time, or better. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC'13)*, pages 765–774. ACM, 2013.

22 Mathias Weller, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. On making directed graphs transitive. *Journal of Computer and System Sciences*, 78(2):559–574, 2012.

On the Parameterized Complexity of Biclique Cover and Partition*

Sunil Chandran[1], Davis Issac[2], and Andreas Karrenbauer[3]

1 **Indian Institute of Science, Bangalore, India**
2 **Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany**
3 **Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany**

Abstract

Given a bipartite graph G, we consider the decision problem called BicliqueCover for a fixed positive integer parameter k where we are asked whether the edges of G can be covered with at most k complete bipartite subgraphs (a.k.a. bicliques). In the BicliquePartition problem, we have the additional constraint that each edge should appear in exactly one of the k bicliques. These problems are both known to be NP-complete but fixed parameter tractable. However, the known FPT algorithms have a running time that is doubly exponential in k, and the best known kernel for both problems is exponential in k. We build on this kernel and improve the running time for BicliquePartition to $\mathcal{O}^*(2^{2k^2+k\log k+k})$ by exploiting a linear algebraic view on this problem. On the other hand, we show that no such improvement is possible for BicliqueCover unless the Exponential Time Hypothesis (ETH) is false by proving a doubly exponential lower bound on the running time. We achieve this by giving a reduction from 3SAT on n variables to an instance of BicliqueCover with $k = \mathcal{O}(\log n)$. As a further consequence of this reduction, we show that there is no subexponential kernel for BicliqueCover unless $P = NP$. Finally, we point out the significance of the exponential kernel mentioned above for the design of polynomial-time approximation algorithms for the optimization versions of both problems. That is, we show that it is possible to obtain approximation factors of $\frac{n}{\log n}$ for both problems, whereas the previous best approximation factor was $\frac{n}{\sqrt{\log n}}$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Biclique Cover/Partition, Linear algebra in finite fields, Lower bound

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.11

1 Introduction

The problems of covering or partitioning the edge set of bipartite graphs have a long history. It was shown by Orlin in 1977 that the covering problem, i.e., to decide for a given bipartite graph G and given integer k, whether the edges of G can be covered by at most k complete bipartite subgraphs (also known as bicliques), is NP-complete [13]. He conjectured the partitioning problem, i.e., where each edge of G must appear in exactly one of the k bicliques, to also be NP-complete, which has since been answered in the affirmative in [10].

The minimum number of bicliques to cover the edges of a graph is also called the *bipartite dimension* [5], and Orlin called the minimum k that admits a partition of the edge set into k

* This work is supported in part by DFG-grant KA 3042/3-1 and the Max Planck Center for Visual Computing and Communication (http://www.mpc-vcc.org).

bicliques the *bicontent* [13]. We prefer the terms *biclique cover number* and *biclique partition number*, respectively, to avoid any confusion.

There are numerous applications for biclique covering and partitioning. Related work can be found in the areas of bioinformatics [11, 12], computer security [3], database tiling [7], finite automata [9], and graph drawing [4].

Besides these applications, computing a biclique cover of a graph is equivalent to other important notions in mathematics: Given an m-by-n matrix A, the *Boolean rank* of A is the minimum k for which there exist two 0-1 matrices B and C of dimensions $m \times k$ and $k \times n$, respectively, such that $A = B \odot C$, where $\odot$ denotes the matrix product over boolean arithmetic. It has been shown that computing the Boolean rank of a matrix is equivalent to computing the biclique cover number of a bipartite graph (see [8]). Similarly, the *binary rank* of a matrix $A \in \{0,1\}^{m \times n}$ is defined as the minimum k for which there are $B \in \{0,1\}^{m \times k}$ and $C \in \{0,1\}^{k \times n}$ such that $A = B \cdot C$ using the standard arithmetic over the reals. It can be shown that the binary rank of the adjacency matrix of a bipartite graph is equal to its biclique partition number.

Low-rank decompositions are of particular interest in Data Analytics. However, it is even NP-hard to distinguish bipartite graphs or binary matrices that allow $k \in O(n^{\varepsilon})$ from ones that require $k \in \Omega(n^{1-\varepsilon})$ for all $\varepsilon > 0$ for both BicliqueCover as well as BicliquePartition [1]. In the same paper, only approximation factors of $\frac{n}{\sqrt{\log n}}$ were obtained with polynomial time algorithms.

From the parameterized complexity point of view, the picture is much brighter: BicliqueCover and BicliquePartition are in FPT when parameterized with k [6], which even holds when the input graph is not bipartite. The authors obtain this result by providing rules for obtaining kernels with at most 3^k vertices for general graphs and with at most 2^{k+1} vertices for bipartite graphs. We will use this kernel to obtain some of our results.

1.1 Our contribution

We present an algorithm that decides whether a given bipartite graph has a BicliquePartition of size at most k in time $\mathcal{O}^*(2^{2k^2+k\log k+k})$. This drastically improves the previous best bound [6], which is $\mathcal{O}^*(2^{2^{2k}\log k+3k})$ [12].[1] In contrast to this result, we prove that BicliqueCover seems to be much harder, i.e., there is no algorithm running in $\mathcal{O}^*(2^{2^{o(k)}})$ unless the Exponential Time Hypothesis (ETH) is false. This almost closes the gap between lower and best known upper bound, which is $\mathcal{O}^*(2^{2^k\log k+2k+\log k}/k!)$ for the latter [12]. Moreover, we prove an exponential lower bound for kernels for BicliqueCover that holds unless $P = NP$. We conclude by showing how the kernel for BicliqueCover and BicliquePartition improves the best known approximation factors for these two problems to $\mathcal{O}(n/\log n)$.

1.2 Preliminaries

For a bipartite graph G, we denote the two vertex bipartitions by $U(G)$ and $V(G)$. The edges are denoted by $E(G)$. For a subgraph H of G, $U(H)$ denotes the set of vertices of H that are in $U(G)$ and $V(H)$ denotes the set of vertices of H that are in $V(G)$. All edges in this paper are undirected edges, and we may use uv or vu to denote an edge between vertices u and v. For a graph G and vertex v, we use $N_G(v)$ to denote the set of all the vertices that are adjacent to v in G. We may choose to omit the subscript G when the graph is clear from

[1] Note that the bound reported in [6] is inaccurate as also observed in [12].

the context. A **domino graph** is the graph G with $U(G) = \{u_1, u_2, u_3\}$, $V(G) = \{v_1, v_2, v_3\}$ and $E(G) = \{u_1v_1, u_1v_2, u_2v_1, u_2v_2, u_2v_3, u_3v_2, u_3v_3\}$ (See S_i in Figure 1). Two vertices w_1 and w_2 in graph G are said to be **twins** of each other iff $N(w_1) = N(w_2)$. For a matrix A, we say that the i^{th} row is a_i^T, the j^{th} column is A_j, and the entry corresponding to the i^{th} row and j^{th} column is a_{ij}. For a prime number p, we use $GF(p)$ to denote the Galois Field over $\{0, 1, \ldots, p-1\}$ with modulo p multiplication and addition.

2 FPT algorithm for BicliquePartition

This section is dedicated to the proof of the following theorem.

▶ **Theorem 1.** BicliquePartition *has an algorithm which runs in* $\mathcal{O}^*(2^{2k^2+k\log k+k})$*-time.*

As mentioned above, the biclique partition number of a bipartite graph is equal to the binary rank of its adjacency matrix. Moreover, the binary decomposition $B \cdot C$ of a binary matrix A with $A \in \{0,1\}^{m\times n}$, $B \in \{0,1\}^{m\times k}$, and $C \in \{0,1\}^{k\times n}$ also gives the set of bicliques in the corresponding biclique partition of the graph represented by A. Therefore, we consider the following problem in the remainder of this section: Given a binary matrix A, does A have binary rank at most k? We develop an $\mathcal{O}^*(2^{2k^2+k\log k+k})$-time algorithm for this problem. Moreover, our algorithm also returns the binary decomposition BC of A if A is a YES instance.

Let A be the given $m \times n$ binary matrix. If A has binary rank k, then there exist an $m \times k$ binary matrix B and a $k \times n$ binary matrix C such that $BC = A$. Let p be the smallest prime that is greater than k. It is a well-known fact that p is at most $2k$ and can be found in time polynomial in k. We will work with arithmetic over the prime field $GF(p)$. We know that $BC = A$ holds even over $GF(p)$. We can assume that $m, n \leq 2^k$ because of the kernel given in [6]. Let s be the rank of B over $GF(p)$. Clearly, s is at most k. Thus, we can guess the value of s. Observe that permuting rows do not change the binary rank of A. Moreover, the binary decomposition BC still holds, provided we permute the rows of B with the same permutation used for rows of A. We guess a permutation of the rows of B such that the first s rows of B are linearly independent over $GF(p)$. Then, we apply the same permutation to the rows of A. Note that it suffices to try $\binom{m}{s}$ possibilities to find one such permutation. Since $m \leq 2^k$ due to the kernel in [6] and $s \leq k$, we get that $\binom{m}{s} \leq 2^{k^2}$, and hence, this step is fine with respect to the running time that we want to achieve. Now, we guess the entries of the first s rows of B. Since each row only has 2^k possibilities, there are only at most $(2^k)^s \leq 2^{k^2}$ possibilities for this guess.

▶ **Lemma 2.** *For each* $i \in [m]$*, there exists a vector* $\lambda(i) \in \{0, 1, \ldots, p-1\}^s$ *such that* $\sum_{t=1}^{s} \lambda(i)_t a_t \equiv a_i \pmod p$ *and* $\sum_{t=1}^{s} \lambda(i)_t b_t \pmod p$ *is a* 0-1 *vector.*

Proof. For an arbitrary i, we exhibit a $\lambda(i)$, which satisfies the conditions stated in the statement of the lemma. Since the first s rows of B span all its rows over $GF(p)$, there exists $\lambda(i) \in \{0, 1, \ldots, p-1\}^s$ such that $\sum_{t=1}^{s} \lambda(i)_t b_t \equiv b_i \pmod p$, which is a 0-1 vector. Furthermore, $\sum_{t=1}^{s} \lambda(i)_t a_t^T \equiv \sum_{t=1}^{s} \lambda(i)_t b_t^T C \equiv b_i^T C \equiv a_i^T \pmod p$. ◀

For each $i \in [m]$, we find a vector $\lambda(i)$ as given by Lemma 2 and let b'_i be the corresponding 0-1 vector given by $\sum_{t=1}^{s} \lambda(i)_t b_t$. For each $i \in [m]$, this can be done by trying all possibilities of $\lambda(i)_1, \ldots, \lambda(i)_s$ and takes only $\mathcal{O}(p^s) = \mathcal{O}(2^{k\log k+k})$ time. Hence, finding all the $\lambda(i)$ for all $i \in [m]$ takes only $\mathcal{O}(2^{k\log k+k}m)$ time. We would like to highlight that the choice of $\lambda(i)$ is done independently for each $i \in [m]$. Note that since we do not yet know the value of

b_i for $i > s$, the $\lambda(i)$ that we fix is not guaranteed to be the one exhibited in the proof of Lemma 2, i.e., b'_i could differ from b_i, but they are the same for $i \leq s$.

Let B' be the matrix whose rows are $b'^T_1, \ldots, b'^T_m$. Let $\tilde{B}$ be the matrix B restricted to its first s rows, and let $\tilde{A}$ be the matrix A restricted to its first s rows. For each $j \in [n]$, we find a vector $C'_j \in \{0,1\}^k$ such that $\tilde{B}C'_j = \tilde{A}_j$. For each $j \in [n]$, this can be done by iterating over all possible binary vectors of length k, which only takes $\mathcal{O}(2^k)$ time. Note that such a C'_j should exist for all $j \in [n]$ if A has binary rank k, provided that our guess about the s linearly independent rows in B is correct. Again, we would like to highlight that the choice of C'_j is done independently for each $j \in [n]$. Let C' denote the matrix whose columns are $C'_1, C'_2, \ldots, C'_n$.

The following lemma is the core idea that our algorithm uses. The lemma essentially says that although we choose the b'_i's and C'_j's independently, they actually combine to give a binary decomposition of A over $GF(p)$.

▶ **Lemma 3.** *For each $i \in [m], j \in [n]$, $b'^T_i C'_j \equiv a_{ij} \pmod p$.*

Proof. $b'^T_i C'_j \equiv \left(\sum_{t=1}^{s} \lambda(i)_t b_t\right)^T C'_j \pmod p \equiv \sum_{t=1}^{s} \lambda(i)_t a_{tj} \pmod p \equiv a_{ij} \pmod p$ ◀

It only remains to eliminate the $GF(p)$ arithmetic, which is done by the following Lemma.

▶ **Lemma 4.** *For each $i \in [m], j \in [n]$, $b'^T_i C'_j = a_{ij}$.*

Proof. From Lemma 3, we have that $b'^T_i C'_j \equiv a_j \pmod p$. Since b'_i and C'_j are 0-1 vectors of length $k < p$, we have $b^T_i C'_j = (b'^T_i C'_j \mod p) = a_{ij}$. ◀

From Lemma 4, we have that $B'C' = A$. Since B' is an $m \times k$ binary matrix and C' is a $k \times n$ binary matrix, we have a binary factorization of A with binary rank k. A pseudocode for the algorithm is given in Algorithm 1, and in Lemma 5, we prove that its running time is $\mathcal{O}^*(2^{2k^2+k\log k+k})$.

▶ **Lemma 5.** *The running time of Algorithm 1 is $\mathcal{O}^*(2^{2k^2+k\log k+k})$.*

Proof. The number of iterations of the outer loop is at most $k \cdot \binom{m}{s} \cdot 2^{sk} = \mathcal{O}^*(2^{2k^2})$, which follows from $m \leq 2^k$ and $s \leq k$. The two inner loops only have n iterations at most. Step 5 takes $O^*(p^s) = O^*(2^{k+k\log k})$ time, and step 9 takes $O^*(2^k)$ time. All other steps take time polynomial in m and n. Hence, the total time taken by the algorithm is $\mathcal{O}^*(2^{2k^2} \cdot (2^{k+k\log k} + 2^k)) = \mathcal{O}^*(2^{2k^2+k\log k+k})$. ◀

3 FPT and kernel lower bounds for BicliqueCover

In this section, we prove the following theorem, which has consequences for the complexity of BicliqueCover as stated in the corollaries below.

▶ **Theorem 6.** *There exists a polynomial time reduction that, given a 3-SAT instance ψ on n variables and m clauses, produces a bipartite graph G with $|U(G)| + |V(G)| = \mathcal{O}(n+m)$ such that there exists a positive integer $k = \mathcal{O}(\log n)$ for which G has a biclique cover of size at most k if and only if ψ is satisfiable.*

▶ **Corollary 7.** BicliqueCover *cannot be solved in time $\mathcal{O}^*(2^{2^{o(k)}})$-time unless the Exponential Time Hypothesis is false.*

Proof. Follows directly from Theorem 6. ◀

Algorithm 1: FPT algorithm for BicliquePartition

Input : An $m \times n$ binary matrix A and positive integer k such that $m, n \leq 2^k$.
Output : Either report that binary rank of A is greater than k or output $m \times k$ binary matrix B' and $k \times n$ binary matrix C' such that $B'C' = A$

Find p, the smallest prime greater than k.
foreach $s \in [k]$,$\{i_1, i_2, \cdots, i_s\} \subseteq [m]$, *and* $\tilde{B} \in \{0,1\}^{s \times k}$ **do** `// loop 1`
  Permute the rows of A such that rows $i_1, i_2, \cdots, i_s$ become the first s rows of A. Let this permutation be σ;
  for $i \leftarrow 1$ **to** m **do**
    Find a $\lambda(i) \in \{0, 1, \cdots, p-1\}^s$ such that $\sum_{t=1}^{s} \lambda(i)_t a_t \equiv a_i \pmod p$ and $\sum_{t=1}^{s} \lambda(i)_t b_t \pmod p$ is a 0-1 vector; **if** there is no such $\lambda(i)$, **then go to** the next iteration of loop 1;
    $b'_i \leftarrow \sum_{t=1}^{s} \lambda_t(i) \tilde{b}_t$;
  end
  for $j \leftarrow 1$ **to** n **do**
    Find $C'_j \in \{0,1\}^k$ such that $\tilde{B}C'_j = \tilde{A}_j$ where $\tilde{A}$ is the matrix A restricted to first s rows; **if** there is no such C'_j, **then go to** the next iteration of loop 1;
  end
  let B' be the matrix with $b'^T_1, b'^T_2, \cdots, b'^T_m$ as the rows and C' be the matrix with $C'_1, C'_2, \cdots, C'_n$ as the columns;
  Apply the inverse permutation of σ to the rows of B';
  output B' and C' and **terminate**.
end
report that binary rank of A is greater than k.

▶ **Corollary 8.** *There exists a constant $\delta > 0$ such that, unless $P = NP$, there is no polynomial time algorithm that produces a kernel for* BicliqueCover *of size less than $2^{\delta k}$.*

Proof. We give a proof sketch and refer to [2] for the details where the authors prove a similar statement for EdgeCliqueCover. By Theorem 6, we have an algorithm A that takes an instance of 3-SAT and gives an equivalent instance of BicliqueCover with parameter $k = \mathcal{O}(\log n)$. Suppose there is a kernelization algorithm B that produces a kernel with less than $2^{\delta k}$ size for some δ to be fixed later. Since BicliqueCover is NP-complete, there exists an algorithm C that takes an instance of BicliqueCover and gives an equivalent instance of 3-SAT in polynomial time. By composing the algorithms A, B, and C and fixing the parameter δ appropriately, we get an algorithm D that, given a 3-SAT instance as input, produces an equivalent smaller 3-SAT instance as output. We can apply D repeatedly to solve 3-SAT. Hence, algorithm B cannot exist. ◀

The proof of Theorem 6 gives a reduction from 3-SAT to BicliqueCover, which is a modification of the one given in [2] from 3-SAT to EdgeCliqueCover.[2] The main difference is that we introduce an additional gadget consisting of $\log_2 n$ domino graph gadgets in order to make the reduction work for BicliqueCover, where n is the number of variables in the input 3-SAT formula. This gadget replaces the independent set of size $\log_2 n$ used in [2], i.e.,

[2] A parameter preserving reduction from EdgeCliqueCover to BicliqueCover would have been better than having to redo the whole reduction from scratch. We could not find any such reduction.

we replace each vertex there by a domino graph here. We also modify some of the adjacencies in the construction such that the graph becomes bipartite. Moreover, we have simplified the reduction of [2] by using a simple trick. The trick is to make one of the domino graphs special by adding edges between this domino graph and clause gadgets so that the biclique covering this domino graph corresponds to a satisfying assignment. For EdgeCliqueCover, this corresponds to making one of the vertices in the independent set special by adding edges from it to the clause gadgets. We give the complete reduction for BicliqueCover here for being self-contained.

In [2], the authors use cocktail party graphs as the main gadget in their reduction. We use the bipartite analogue called *crown graphs*. A crown graph is basically a complete bipartite graph minus a perfect matching. It is formally defined as follows.

▶ **Definition 9** (Crown Graph, H_r)**.** A crown graph on $2r$ vertices denoted by H_r is a bipartite graph with bipartitions $U(G) = \{u_1, u_2, \ldots, u_r\}$ and $V(G) = \{v_1, v_2, \ldots, v_r\}$ such that there is an edge from u_i to v_j iff $i \neq j$. In other words, the edges missing between $U(H_r)$ and $V(H_r)$ form a perfect matching given by $\{u_i v_i : i \in [r]\}$. (See $H = H_n$ in Figure 1.)

If we pick exactly one vertex from each of the edges of the missing perfect matching of the crown graph, then we get a maximal biclique provided that we pick at least one vertex from each of the bipartitions. The complement of this vertex set also forms a maximal biclique. These pair of bicliques are called duplex bicliques, formally defined as follows.

▶ **Definition 10** (Duplex Biclique[3])**.** A duplex biclique of a crown graph H_r is defined as a pair of bicliques $\{B_1, B_2\}$ such that $U(B_1) \cap U(B_2) = \emptyset, V(B_1) \cap V(B_2) = \emptyset$, and $U(B_1) \cup U(B_2) = U(H_r)$, and $V(B_1) \cup V(B_2) = V(H_r)$.

We go on to define a duplex biclique cover as follows.

▶ **Definition 11** (Duplex Biclique Cover)**.** A duplex biclique cover of a crown graph is defined as a set of duplex bicliques that together cover all the edges of the graph. When we say size of a duplex biclique cover, we mean the number of bicliques in the cover, which is twice the number of duplex bicliques.

We prove the following two lemmas about crown graphs.

▶ **Lemma 12.** *H_r has a duplex biclique cover of size $2\lceil \log r \rceil$ that can be found in time polynomial in n.*

Proof. We exhibit such a biclique cover. Let $\ell = \lceil \log r \rceil$. For any $x \in \{0, \ldots, 2^\ell - 1\}$ and $j \in [\ell]$, let $\langle x \rangle_j$ denote the j-th bit of the ℓ-bit binary representation of x. For each $j \in [\ell]$, we define the j-th duplex biclique $\{T_j^1, T_j^2\}$ as follows: T_j^1 is the subgraph induced by $\{u_i : \langle i-1 \rangle_j = 1\} \cup \{v_i : \langle i-1 \rangle_j = 0\}$, and T_j^2 is the subgraph induced by $\{u_i : \langle i-1 \rangle_j = 0\} \cup \{v_i : \langle i-1 \rangle_j = 1\}$, where u_i and v_i are defined as in Definition 9. It is easy to see that $\{T_j^1, T_j^2\}$ is indeed a duplex biclique. It is also easy to see that any pair of vertices u_i, v_j such that $i \neq j$ should be present in at least one of the ℓ duplex bicliques. ◀

▶ **Lemma 13.** *Given a duplex biclique $\{B_1, B_2\}$ of H_r such that $|U(B_1)| = |V(B_1)|$, we can in polynomial time find a duplex biclique cover of H_r with size $2\lceil \log_2 r \rceil$ such that $\{B_1, B_2\}$ is one of the duplex bicliques forming the biclique cover.*

[3] Duplex Bicliques correspond to Twin Cliques in [2]. We use this name to avoid confusion with twin vertices.

Proof. Using the definition of duplex bicliques and $|U(B_1)| = |V(B_1)|$, it follows that $[r]$ can be partitioned into 2 sets

$$J_1 = \{j \in [r] : u_j \in U(B_1) \wedge v_j \in V(B_2)\} \text{ and } J_2 = \{j \in [r] : u_j \in U(B_2) \wedge v_j \in V(B_1)\}$$

which are each of size $\frac{r}{2}$. We can reorder the indices of the vertices such that $J_1 = \left\{1, 2, \cdots \frac{r}{2}\right\}$ and $J_2 = \left\{\frac{r}{2}+1, \frac{r}{2}+2, \cdots, r\right\}$. Let $\ell = \log_2 r$. We define the duplex biclique $\{T_i^1, T_i^2\}$ for all $i \in [\ell]$ the same way as in the proof of Lemma 12. It is clear that the duplex biclique $\{B_1, B_2\}$ is the same as the duplex biclique $\{T_1^1, T_1^2\}$. Thus, the set of bicliques $\left\{T_1^1, T_2^1, \ldots, T_\ell^1\right\} \cup \left\{T_1^2, T_2^2, \ldots, T_\ell^2\right\}$ gives the required duplex biclique cover. ◀

Now, we state the following fact about twin vertices.

▶ **Fact 14.** *If a_1 and a_2 are twins and $E(G \setminus a_2)$ can be covered with k bicliques, then $E(G)$ can be covered with k bicliques by adding a_2 to all the bicliques containing a_1.*

When we say we apply **twin-reduction** to a pair of twin vertices, we mean the operation of deleting one of the twin vertices from the graph. When we say we apply twin-reduction to a graph, we mean to repeatedly apply twin-reduction until there are no more twins in the graph.

Let ψ be the input 3-SAT formula with n variables and m clauses. Let $x_1, \ldots, x_n$ be the variables of ψ and $C_1, \ldots, C_m$ be the clauses. Let C_i^1, C_i^2, and C_i^3 denote the 3 literals of clause C_i. For $1 \leq a \leq 3$, we say that $C_i^a = (x_j, 1)$ if the a^{th} literal in clause C_i is the variable x_j appearing in positive form, and we say $C_i^a = (x_j, 0)$ if the a^{th} literal in C_i is the variable x_j appearing in negated form.

Assumptions about the input 3-SAT formula: We assume that the number of variables is a power of 2. We also assume that if the instance is satisfiable, then there is a satisfying assignment A such that half of the variables are assigned true in A and the other half false. These assumptions can be handled easily by introducing some extra variables as shown in [2]. Note that this increases the number of variables by at most 4 times.

Let ℓ be such that $2^\ell = n$. We have that $\ell \in \mathbb{Z}$ since n was assumed to be a power of 2. Before giving the reduction, we give the following useful definition.

▶ **Definition 15** (Bisimplicial Edge)**.** An edge uv is said to be bisimplicial with respect to a biclique B iff $N(u) \cup N(v) = U(B) \cup V(B)$.

Now, we give the reduction from 3-SAT to BicliqueCover.

Construction: Given ψ, we construct a bipartite graph G. See Figure 1 for an illustration of the construction. A vertex with superscript u indicates that it belongs to $U(G)$, and a superscript v indicates that it belongs to $V(G)$. The edges of G are divided into two sets, a set of important edges E^{imp} and a set of free edges E^{free}. The number of bicliques required to cover E^{imp} will be different depending on whether ψ is satisfiable or not, whereas the number of bicliques required to cover E^{free} will depend only on the number of variables and clauses of ψ but not on whether ψ is satisfiable or not. There are 5 main gadgets in our construction of G as given below.

1. A graph H isomorphic to the crown graph H_n: Let the vertices of $U(H)$ be $h_1^u, h_2^u, \ldots, h_n^u$ and that of $V(H)$ be $h_1^v, h_2^v, \ldots, h_n^v$. The edges of H are in E^{imp}. The vertices h_i^u and h_i^v correspond to the i-th variable of ψ. h_i^u corresponds to the variable in positive form and the vertex h_i^v corresponds to the variable in negative form.

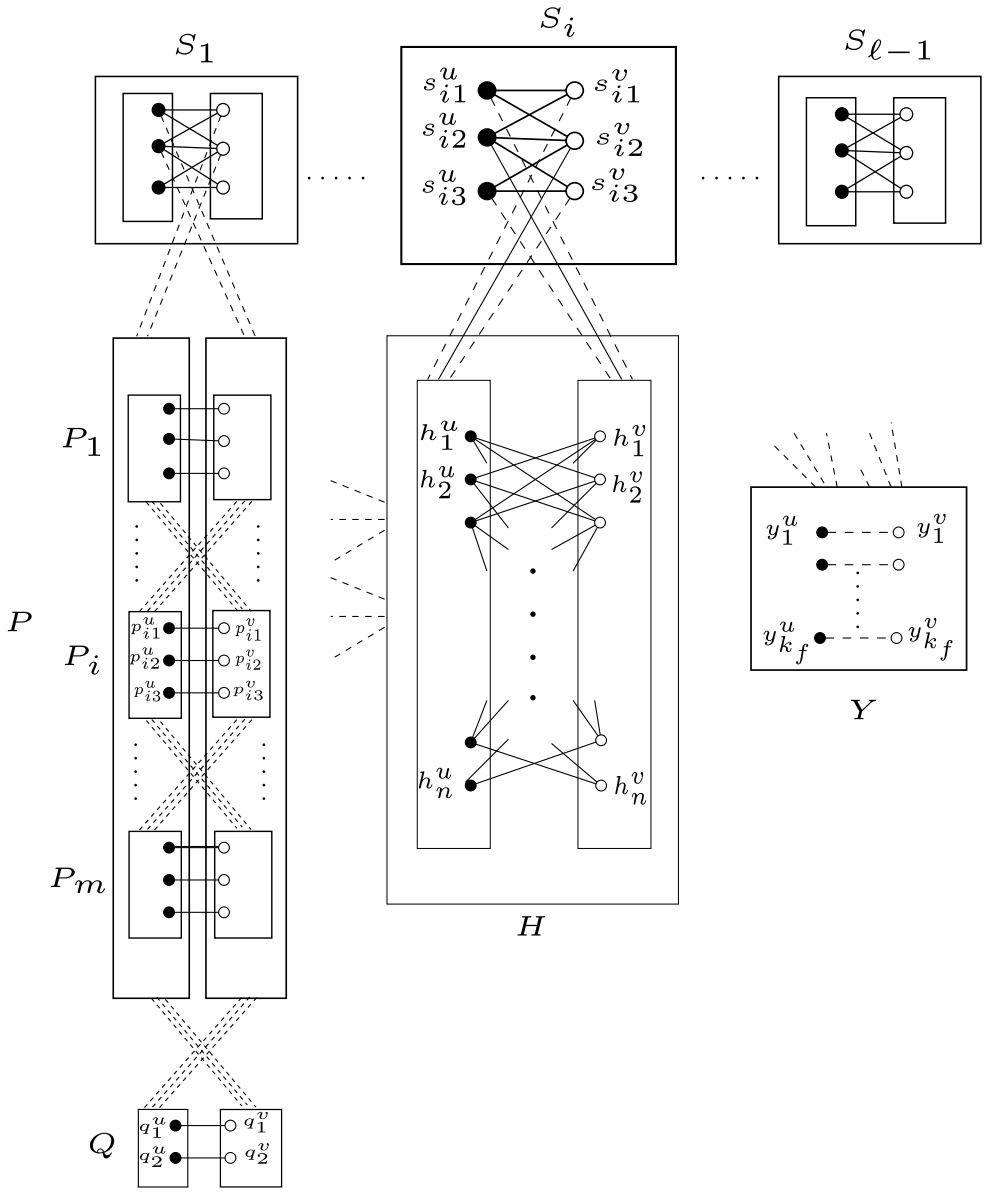


Figure 1 Illustration of the construction of G. The black nodes denote the vertices in $U(G)$, and the white nodes denote the vertices in $V(G)$. The solid edges represent edges in E^{imp}, and the dashed edges denote edges in E^{free}. The edges between P and H and those between Y and $G \setminus Y$ are not shown. The edges shown between S_i and H are present for all $i \in [\ell - 1]$, whereas the edges shown between S_1 and P are only present for S_1 and not for any S_i for $i \geq 2$.

2. A set P of clause gadgets $P_1, \ldots, P_m$: Each P_i is an induced matching of size 3. Let $U(P_i) = \{p^u_{i1}, p^u_{i2}, p^u_{i3}\}$ and $V(P_i) = \{p^v_{i1}, p^v_{i2}, p^v_{i3}\}$ and let the 3 edges of P_i be $p^u_{i1}p^v_{i1}, p^u_{i2}p^v_{i2}$, and $p^u_{i3}p^v_{i3}$. These edges are in E^{imp}. For all $i \in [m]$, P_i corresponds to the clause C_i in ψ, and the 3 edges in P_i correspond to the 3 literals in the clause, i.e., edge $p^u_{ia}p^v_{ia}$ corresponds to literal C^a_i for $a \in \{1, 2, 3\}$.
3. A set S of ℓ domino graphs $S_1, S_2, \ldots, S_\ell$ that are disconnected with each other: Let $U(S_i) = \{s^u_{i1}, s^u_{i2}, s^u_{i3}\}$ and $V(S_i) = \{s^v_{i1}, s^v_{i2}, s^v_{i3}\}$. The edges within each S_i are in E^{imp}.

4. An induced matching Y of size k_f where $k_f = \mathcal{O}(\log n)$ will be fixed later in Lemma 16: Y consists of edges $y_1^u y_1^v, \ldots, y_{k_f}^u y_{k_f}^v$, which are in E^{free}. For all $i \in [k_f]$, the edge $y_i^u y_i^v$ will be made bisimplicial with respect to the biclique $\tilde{B}_i^f$, which will be defined later. This is done to ensure that we need k_f bicliques to cover the free edges.

We also have the following edges between gadgets.

- Between H and S: For all $i \in [\ell]$ and $j \in [n]$, we add the following edges: $s_{i2}^u h_j^v$ and $s_{i2}^v h_j^u$ to E^{imp}; and, $s_{i1}^u h_j^v$, $s_{i3}^u h_j^v$, $s_{i1}^v h_j^u$, and $s_{i3}^v h_j^u$ to E^{free}.
- Between P_i and P_j: For all $i \neq j \in [m]$, add edges between all pairs of vertices u, v such that $u \in U(P_i)$ and $v \in V(P_j)$. These edges are in E^{free}.
- Between P and Q: For all $i \in [m]$, add edges between all pairs of vertices u, v such that $u \in U(Q)$ and $v \in V(P_i)$. Similarly, for all $i \in [m]$, add edges between all pairs of vertices u, v such that $u \in U(P_i)$ and $v \in V(Q)$. These edges are in E^{free}.
- Between H and P: For all $i \in [m]$ and $a \in [3]$, add edges between p_{ia}^u and h_j^v unless $C_i^a = (x_j, 1)$ and between p_{ia}^v and h_j^u unless $C_i^a = (x_j, 0)$. These edges are in E^{free}.
- Between S and P: The only vertices in S that will have edges to any P_i are the 4 vertices $s_{11}^u, s_{12}^u, s_{11}^v$, and s_{12}^v. From s_{11}^u and s_{12}^u, add edges to all vertices in $V(P_i)$ for all $i \in [m]$. Similarly, from s_{11}^v and s_{12}^v, add edges to all vertices in $U(P_i)$ for all $i \in [m]$. These edges are in E^{free}.
- Between Y and $G \setminus Y$: These edges are added in such a way that edge $y_i^u y_i^v$ is bisimplicial w.r.t. a biclique that will be defined later. We will give the exact description of these edges after we define the bicliques B_i^f for $i \in [k_f]$. These edges belong to E^{free}.

Summary of $\mathbf{E^{imp}}$: All the edges within H, S, and Q; all edges within each P_i; edges $s_{i2}^u h_j^v$ and $s_{i2}^v h_j^u$ for all $i \in [\ell - 1], j \in [n]$.

Summary of $\mathbf{E^{free}}$: All the edges between P_i and P_j for $i \neq j$; all edges within Y; all edges between Y and $G \setminus Y$; edges $s_{i1}^u h_j^v$, $s_{i3}^u h_j^v$, $s_{i1}^v h_j^u$, and $s_{i3}^v h_j^u$ for all $i \in [\ell - 1], j \in [n]$; all edges between P and Q, between H and P, and between S_1 and P.

First, we show how to take care of the edges in E^{free} without interfering with the budget of E^{imp}. Let E^y be the set of all edges of G with at least one end point in $U(Y) \cup V(Y)$.

▶ **Lemma 16.** *The edges in $E^{free} \setminus E^y$ can be covered using $k_f = 4\log_2 n + 2\lceil \log_2 m \rceil + 6$ bicliques of G such that none of these bicliques contains an edge from E^{imp}, and these bicliques can be found in time polynomial in $n + m$.*

Proof. According to our construction, there are the following types of edges in $E^{free} \setminus E^y$. For each of these types, we show how to cover it in polynomial time using bicliques that do not contain any edges from E^{imp} such that the total number of bicliques used is at most k_f.

- Edges between H and S: These edges can be covered with 2 bicliques, B_1^{HS} and B_2^{HS} defined as follows. $U(B_1^{HS}) = U(H)$, $V(B_1^{HS}) = \{s_{i1}^v : 1 \leq i \leq \ell\} \cup \{s_{i3}^v : 1 \leq i \leq \ell\}$, $U(B_2^{HS}) = \{s_{i1}^u : 1 \leq i \leq \ell\} \cup \{s_{i3}^u : 1 \leq i \leq \ell\}$, and $V(B_2^{HS}) = V(H)$. From the construction of G, it is easy to see that both B_1^{HS} and B_2^{HS} are indeed bicliques, and none of them contains an edge in E^{imp}.
- $E(P) \cap E^{free}$: These edges can be covered with $2\lceil \log_2 m \rceil$ bicliques. Consider the subgraph of G given by the edges $E(P) \setminus E^{imp}$. Let this graph be G_1. The vertices p_{i1}^u, p_{i2}^u, and p_{i3}^u are twins of each other in G_1 for all $i \in [m]$. Similarly, the vertices p_{i1}^v, p_{i2}^v, and p_{i3}^v are twins of each other in G_1 for all $i \in [m]$. Let G_2 be the graph obtained by applying twin-reduction to G_1. It is clear that G_2 is isomorphic to the crown graph H_m. Hence,

$E(G_2)$ can be covered by $\lceil 2\log_2 m \rceil$ bicliques in polynomial time by Lemma 12. Then, by using Fact 14, we can find $\lceil 2\log_2 m \rceil$ bicliques that cover $E(G_1) = E(P) \setminus E^{imp}$.

- Edges between P and Q: We can cover these edges with 2 bicliques B_1^{PQ} and B_2^{PQ}, defined as $U(B_1^{PQ}) = U(P)$, $V(B_1^{PQ}) = V(Q)$; and, $U(B_2^{PQ}) = U(Q)$, $V(B_2^{PQ}) = V(P)$.
- Edges between H and P: We cover these edges with $4\log_2 n$ bicliques. We will show how to cover the edges between $U(H)$ and $V(P)$ with $2\log_2 n$ bicliques. Symmetrically, the edges between $V(H)$ and $U(P)$ can be covered with another $2\log_2 n$ bicliques. Let $\langle j \rangle_i$ denote the i^{th} bit in the binary representation of j. We will now give the description of a set of $2\log_2 n$ bicliques $\mathcal{B}^{HP} = \{B_1^{HP}, \ldots, B_{\log_2 n}^{HP}\} \cup \{\tilde{B}_1^{HP}, \ldots, \tilde{B}_{\log_2 n}^{HP}\}$ covering the edges between $U(H)$ and $V(P)$. We define $U(B_i^{HP}) = \{h_j^u : \langle j \rangle_i = 1\}$, $U(\tilde{B}_i^{HP}) = \{h_j^u : \langle j \rangle_i = 0\}$, $V(B_i^{HP}) = \bigcap_{u \in U(B_i^{HP})} N(u) \cap V(P)$, $V(\tilde{B}_i^{HP}) = \bigcap_{u \in U(\tilde{B}_i^{HP})} N(u) \cap V(P)$. It is clear that these are indeed bicliques from the definitions of $V(B_i^{HP})$ and $V(\tilde{B}_i^{HP})$. Since there are no edges of E^{imp} between $U(H)$ and $V(P)$, we do not cover any edges in E^{imp}. We now show that we have covered every edge between $U(H)$ and $V(P)$. Suppose for the sake of contradiction that the edge $h_j^u p_{ia}^v$ was not covered. Let p_{ia}^v correspond to variable x_t. Recall that the only vertex in $U(H)$ that can possibly not have edge to p_{ia}^v is h_t^u. Edge $h_j^u p_{ia}^v$ not being covered by any biclique in $\mathcal{B}$ can happen only if every biclique in $\mathcal{B}^{HP}$ that contains h_j^u also contains h_t^u and if there is no edge between h_t^u and p_{ia}^v. But if every biclique in $\mathcal{B}$ containing h_j^u also contains h_t^u, then $j = t$. This means that there is no edge between h_j^u and p_{ia}^v, which is a contradiction.
- Edges between S and P: These edges can be covered with 2 bicliques B_1^{PS} and B_2^{PS}, which is defined as follows. $U(B_1^{PS}) = U(P)$, $V(B_1^{PS}) = \{s_{11}^v, s_{12}^v\}$, $U(B_2^{PS}) = \{s_{11}^u, s_{12}^u\}$ and $V(B_2^{PS}) = V(P)$. ◀

We fix k_f as given by Lemma 16. By Lemma 16, we know that there are k_f bicliques that together cover all edges in $E^{free} \setminus E^y$ and do not cover any edges in E^{imp}. We will call these bicliques $B_1^f, \ldots, B_{k_f}^f$.

Now, we give the description of the edges from Y to $G \setminus Y$. Recall that these edges are contained in $E^y \subset E^{free}$. For each $i \in [k_f]$, we add edges from y_i^u to all the vertices in $V(B_i^f)$ and from y_i^v to all vertices in $U(B_i^f)$. Observe that now the edge $y_i^u y_i^v$ is bisimplicial with respect to B_i^f. This together with Lemma 16 gives the following Lemma about the edge set E^{free}.

▶ **Lemma 17.** *Let $k_f = 4\log_2 n + 2\lceil \log_2 m \rceil + 6$.*

1. *The edge set E^{free} can be covered using k_f bicliques of G such that none of these bicliques contains an edge from E^{imp}, and these bicliques can be found in time polynomial in $n+m$.*
2. *Any set of bicliques covering E^{free} has k_f bicliques that do not contain any edges from E^{imp}.*

Proof. From Lemma 16, we know that $E^{free} \setminus E^y$ can be covered by k_f bicliques that do not cover any edges from E^{imp}. Given these k_f bicliques $B_1^f, \ldots, B_{k_f}^f$, we extend them to the bicliques $\tilde{B}_i^f, \ldots, \tilde{B}_{k_f}^f$ as follows to cover all the edges of E^{free} : $U(\tilde{B}_i^f) = U(B_i^f) \cup \{y_i^u\}$, and $V(\tilde{B}_i^f) = V(B_i^f) \cup \{y_i^v\}$. It is clear that $\tilde{B}_1^f, \ldots, \tilde{B}_n^f$ are all indeed bicliques, they together cover all edges of E^{free}, and do not cover any edges of E^{imp}.

Since the edges within Y form an induced matching of size k_f, no two of them can be present in the same biclique. Hence, we need at least k_f bicliques to cover the edges in E^{free}. We now show that if a biclique contains an edge from E^{imp}, it cannot have an edge from Y, which will complete the proof of the lemma. Suppose the edge $y_i^u y_i^v$ and an important edge $z^u z^v \in E^{imp}$ are in the same biclique for the sake of contradiction. But since $N(y_i^u) = V(B_i^f) \cup \{y_i^v\}$, we

have that $z^v \in V(B_i^f)$. Symmetrically, we can argue that $z^u \in U(B_i^f)$. Then, however, B_i^f contains the important edge $z^u z^v$, which is a contradiction. ◀

We set our budget k as $k_f + 2\ell + 2$, which means a budget of $2\ell + 2$ for the edges in E^{imp} due to Lemma 17. Now, we argue the completeness of the reduction in the following Lemma.

▶ **Lemma 18.** *If ψ is satisfiable, then the edges in E^{imp} can be covered by $2\ell + 2$ bicliques of G. (These bicliques might contain some edges from E^{free} as well.)*

Proof. We know that there exists a satisfying assignment A of ψ such that exactly half of the variables are assigned *true* in A. Each clause C_i has at least one literal which satisfies the clause. For each clause C_i, we fix one such literal. This literal corresponds to one of the 3 edges of P_i. Let us denote this edge by e_i.

We use two bicliques, B_1^g and B_2^g, to cover the 2 guard edges of Q and 2 edges from each P_i. Each of B_1^g and B_2^g covers 1 edge from Q and 1 edge from each P_i. It is clear that this can be done. Now, each P_i has one edge still to be covered. We will assign B_1^g and B_2^g such that the edge left uncovered in P_i is e_i, i.e., the literal corresponding to this edge evaluates to *true* in A.

Let B_1 be the biclique defined as follows: $U(B_1) = \{h_i^u : A(x_i) = true, i \in [n]\}$ and $V(B_1) = \{h_i^v : A(x_i) = false, i \in [n]\}$. Also, define $\bar{B}_1$ as the biclique defined by the vertex sets $U(\bar{B}_1) = U(H) \setminus U(B_1)$ and $V(\bar{B}_1) = V(H) \setminus V(B_1)$. B_1 and $\bar{B}_1$ are indeed bicliques of H because no literal evaluates to both *true* and *false*, and thus, the missing edges corresponding to the missing perfect matching in H are avoided. Likewise, they are duplex bicliques due to the manner in which $\bar{B}_1$ is defined. Moreover, $|U(B_1)| = |V(B_1)|$ since A has half of the variables assigned *true* and the other half $false$. Therefore, by Lemma 13, there exist $\ell - 1$ other duplex bicliques $\left\{B_2, \bar{B}_2\right\}, \ldots, \left\{B_\ell, \bar{B}_\ell\right\}$ such that $B_1, \bar{B}_1, B_2, \bar{B}_2, \ldots, B_\ell$, and $\bar{B}_\ell$ together cover $E(H)$. Now we extend these bicliques with additional vertices so that these bicliques together with B_1^g and B_2^g cover E^{imp}, which is done as follows. For $2 \leq j \leq \ell$, we define biclique B'_j as $U(B'_j) = U(B_j) \cup \left\{s_{j1}^u, s_{j2}^u\right\}$ and $V(B'_j) = V(B_j) \cup \left\{s_{j1}^v, s_{j2}^v\right\}$. For $1 \leq j \leq \ell$, we define $\bar{B}'_j$ as $U(\bar{B}'_j) = U(\bar{B}_j) \cup \left\{s_{j2}^u, s_{j3}^u\right\}$ and $V(\bar{B}'_j) = V(B_j) \cup \left\{s_{j2}^v, s_{j3}^v\right\}$. B'_1 is defined as $U(B'_1) = U(B_1) \cup \{s_{11}^u, s_{12}^u\} \cup \bigcup_{i \in [m]} U(e_i)$ and $V(B'_1) = V(B_1) \cup \{s_{11}^v, s_{12}^v\} \cup \bigcup_{i \in [m]} V(e_i)$. It is clear that each B'_i and $\bar{B}'_i$ is indeed a biclique of G and that the bicliques $B'_1, B'_2, \ldots, B'_\ell, \bar{B}'_1, \bar{B}'_2, \ldots, \bar{B}'_\ell, B_1^g$, and B_2^g together cover E^{imp}. Hence, E^{imp} can be covered by $2\ell + 2$ bicliques of G. ◀

Now, we argue the soundness of our reduction in the next Lemma.

▶ **Lemma 19.** *If E^{imp} can be covered by using $2\ell + 2$ bicliques of G, then ψ is satisfiable.*

Proof. The edge set $M = \{q_1^u q_1^v, q_2^u q_2^v\} \cup \left\{s_{j1}^u s_{j1}^v : j \in [\ell]\right\} \cup \left\{s_{j3}^u s_{j3}^v : j \in [\ell]\right\}$ forms an induced matching of size $2\ell + 2$ in G. Recall that all these edges are in E^{imp}. Since no two edges of an induced matching can be contained in the same biclique, each edge in M has to be covered by a distinct biclique. Let the biclique that covers $q_1^u q_1^v$ be B_1^g and the one that covers $q_2^u q_2^v$ be B_2^g. Let B_j and $\bar{B}_j$ be the bicliques covering the edges $s_{j1}^u s_{j1}^v$ and $s_{j3}^u s_{j3}^v$, respectively. Since we have already used our budget of $2\ell + 2$, all the edges in E^{imp} must be covered by at least one biclique in $\mathcal{B} = \left\{B_g^1, B_g^2\right\} \cup \{B_1, B_2, \cdots B_\ell\} \cup \left\{\bar{B}_1, \bar{B}_2, \cdots, \bar{B}_\ell\right\}$. The only possible bicliques in $\mathcal{B}$ that can contain s_{j2}^u or s_{j2}^v are B_j and $\bar{B}_j$. That means, edges between s_{j2}^u and H and edges between s_{j2}^v and H have to be covered by $\{B_j, \bar{B}_j\}$. Moreover, they have to be partitioned by $\left\{B_j, \bar{B}_j\right\}$ for the following reason: if the edge $s_{j2}^u h_i^v$ appears in both bicliques B_j and $\bar{B}_j$, then the edge $s_{j2}^v h_i^u$ cannot appear in any of the two bicliques as there is no edge between h_i^u and h_i^v; and symmetrically, if the edge

$s^v_{j2}h^u_i$ appears in both bicliques B_j and $\bar{B}_j$, then the edge $s^u_{j2}h^v_i$ cannot appear in any of the two bicliques. Combined with the fact that $N(\{s^u_{j2}, s^v_{j2}\}) = U(H) \cup V(H)$, we get that $\{B'_j, \bar{B}'_j\}$ is a duplex biclique, where B'_j and $\bar{B}'_j$ are the intersection of the bicliques B_j and $\bar{B}_j$, respectively, with H.

B^g_1 and B^g_2 can each cover at most 1 edge from each P_i. Hence, there is at least 1 edge of each P_i that must be covered by $\mathcal{B} \setminus \{B^1_g, B^2_g\}$. Let us fix one such edge for each P_i and call it e_i. Since, there are no edges from end points of e_i to any S_j for $j \geq 2$, we know that each e_i must be covered by B_1 or $\bar{B}_1$. But since end points of e_i are not adjacent to s^u_{13} and s^v_{13}, e_i cannot be covered by $\bar{B}_1$. Thus, each e_i has to be covered by B_1.

Now, we construct an assignment A according to B_1 as follows. For each $i \in [n]$, since $\{B_1, \bar{B}_1\}$ is a duplex biclique, B_1 contains exactly one among h^u_i and h^v_i. If $h^u_i \in U(B_1)$, then we assign $x_i = true$ in the assignment A. Otherwise, i.e, if $h^v_i \in V(B_1)$, then we assign $x_i = false$ in A. We claim that A must be a satisfying assignment for ψ, which can be observed as follows. Consider an arbitrary clause C_j. Let x_i be the variable corresponding to e_j. Suppose x_i occurs in positive form in C_j. From the construction of edges between H and P, we know that there cannot be an edge from h^v_i and end points of e_j. Hence, B_1 cannot contain h^v_i. But, since B_1 should contain one of h^u_i and h^v_i, it should contain h^u_i. This means that we assigned $x_i = true$ in A and hence A satisfies clause C_j. Symmetrically, we can argue that if x_i occurred in the negative form in C_j, then we would have assigned $x_i = false$ in A. Thus, A satisfies all the clauses of ψ. ◀

The statement of Theorem 6 follows from Lemmas 17, 18, and 19.

4 Approximation of BicliqueCover and BicliquePartition

In this section, we use the exponential kernel given in [6] to get a polynomial time approximation algorithm for the optimization versions of BicliqueCover and BicliquePartition, achieving an approximation ratio of $\frac{n}{\log_2 n}$. In the optimization versions of the problems, we are required to find the biclique cover/partition with the smallest size. First, we give a useful definition and then proceed towards describing the algorithm. We give the complete algorithm including the reduction rule used for kernelization in [6].

▶ **Definition 20** (Star). A star of a vertex w in a graph is the subgraph induced by $\{w\} \cup N(w)$.

Algorithm: Let G be the input graph. If there exist twin vertices w_1 and w_2 in G, we remove one of them (say, w_2) and the edges incident on it and then recurse by finding the biclique cover/partition of the remaining graph $G \setminus w_2$. After finding the biclique cover/partition of $G \setminus w_2$, we add w_2 to all the bicliques in the solution that contain w_1. If G contains no twins, we output the set of stars of all vertices in $U(G)$ as the biclique cover/partition. We prove in Theorem 21 that this algorithm achieves an approximation ratio of $n/\log_2 n$.

▶ **Theorem 21.** *The above algorithm correctly finds a biclique cover/partition of G whose size is at most $\frac{n}{\log_2 n} \cdot k$, where k is the size of the minimum biclique cover/partition and n is the number of vertices of G. In other words, the algorithm is a polynomial time approximation algorithm for the optimization versions of* BicliqueCover *and* BicliquePartition*, giving an approximation ratio of $\frac{n}{\log_2 n}$.*

Proof. First let us prove the correctness, i.e, we indeed output a biclique cover/partition of G. The correctness of reducing twins is clear and is already proven in [6]. In the case when G does not contain twins, the correctness follows because each star is a biclique, and

each edge of G is present in exactly one of the stars of the vertices in $U(G)$. Let k be the number of bicliques in the optimal biclique cover/partition. Since we do not add any extra bicliques while reducing twins, we need only to consider the case when G has no twins, in order to estimate the size of the biclique/cover partition output by our algorithm. In this case, we know that $|U(G)|, |V(G)| \leq 2^k$ from [6]. Hence, the number of bicliques in the cover/partition that we output is at most $\min\{n, 2^k\} = \frac{\min\{n,2^k\}}{k} \cdot k \leq \frac{n}{\log_2 n} \cdot k$. ◀

Acknowledgements. We are grateful to Erik Jan van Leeuwen for helpful discussions.

References

1. Parinya Chalermsook, Sandy Heydrich, Eugenia Holm, and Andreas Karrenbauer. Nearly Tight Approximability Results for Minimum Biclique Cover and Partition. In *Algorithms – ESA 2014*, volume 8737 of *LNCS*, pages 235–246. Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-662-44777-2_20.
2. Marek Cygan, Marcin Pilipczuk, and Michał Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM Journal on Computing*, 45(1):67–83, 2016. doi:10.1137/130947076.
3. Alina Ene, William Horne, Nikola Milosavljevic, Prasad Rao, Robert Schreiber, and Robert E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *SACMAT'08: Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 1–10, New York, NY, USA, 2008. ACM. doi:10.1145/1377836.1377838.
4. David Eppstein, Michael T. Goodrich, and Jeremy Yu Meng. Confluent layered drawings. *Algorithmica*, 47:439–452, 2007. doi:10.1007/s00453-006-0159-8.
5. Peter C. Fishburn and Peter L. Hammer. Bipartite dimensions and bipartite degrees of graphs. *Discrete Math.*, 160(1-3):127–148, 1996. doi:10.1016/0012-365X(95)00154-O.
6. Herbert Fleischner, Egbert Mujuni, Daniël Paulusma, and Stefan Szeider. Covering graphs with few complete bipartite subgraphs. *TCS*, 410(21-23):2045–2053, 2009. doi:10.1016/j.tcs.2008.12.059.
7. Floris Geerts, Bart Goethals, and Taneli Mielikäinen. Tiling databases. In *Discovery Science*, pages 278–289. Springer, 2004.
8. David A. Gregory, Norman J. Pullman, Kathryn F. Jones, and J. Richard Lundgren. Biclique coverings of regular bigraphs and minimum semiring ranks of regular matrices. *J. Comb. Theory, Ser. B*, 51(1):73–89, 1991. doi:10.1016/0095-8956(91)90006-6.
9. Hermann Gruber and Markus Holzer. Inapproximability of Nondeterministic State and Transition Complexity Assuming P≠NP. In Tero Harju, Juhani Karhumäki, and Arto Lepistö, editors, *Developments in Language Theory*, volume 4588 of *Lecture Notes in Computer Science*, pages 205–216. Springer, 2007. doi:10.1007/978-3-540-73208-2_21.
10. Tao Jiang and B. Ravikumar. Minimal NFA Problems are Hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993. doi:10.1137/0222067.
11. Dana S. Nau, George Markowsky, Max A. Woodbury, and D. Bernard Amos. A mathematical analysis of human leukocyte antigen serology. *Math. Biosciences*, 40(3-4):243–270, 1978. doi:10.1016/0025-5564(78)90088-3.
12. Igor Nor, Danny Hermelin, Sylvain Charlat, Jan Engelstadter, Max Reuter, Olivier Duron, and Marie-France Sagot. Mod/Resc parsimony inference: Theory and application. *Inf. Comput.*, 213:23–32, 2012. doi:10.1016/j.ic.2011.03.008.
13. James Orlin. Contentment in graph theory: Covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80(5):406–424, 1977. doi:10.1016/1385-7258(77)90055-5.

Exact Algorithms for List-Coloring of Intersecting Hypergraphs

Khaled Elbassioni

Masdar Institute of Science and Technology, Abu Dhabi, United Arab Emirates
kelbassioni@masdar.ac.ae

Abstract

We show that list-coloring for any intersecting hypergraph of m edges on n vertices, and lists drawn from a set of size at most k, can be checked in quasi-polynomial time $(mn)^{o(k^2 \log(mn))}$.

1998 ACM Subject Classification G.2.1 Combinatorial algorithms, G.2.2 Hypergraphs

Keywords and phrases Hypergraph coloring, monotone Boolean duality, list coloring, exact algorithms, quasi-polynomial time

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.12

1 Introduction

Hypergraph k-Coloring is the problem of checking whether the vertex-set of a given hypergraph (family of sets) can be colored with at most k colors such that every edge receives at least two *distinct* colors. It is a basic problem in theoretical computer science and discrete mathematics which has received considerable attention (see, e.g. [3, 4, 11, 12, 26, 29, 38]). The problem is NP-complete already for $k = 2$, and in fact, it is quasi-NP-hard[1] to decide if a 2-colorable hypergraph can be (properly) colored with $2^{(\log n)^{\Omega(1)}}$ colors [26]. On the positive side, there exist polynomial time algorithms that can color an $O(1)$-colorable hypergraph with $n^{O(1)}$ colors, where n is the number of vertices (see, e.g., [1, 9, 30]). Several generalizations of the problem have also been considered, for example, List-coloring where every vertex can take only colors from a given list of colors [20, 37].

Given the intrinsic difficulty of the problem, it is natural to consider special classes of hypergraphs for which the problem is easier. Some better results exist for special classes, e.g., better approximation algorithms for hypergraphs of low discrepancy and rainbow-colorable hypergraphs [5], polynomial time algorithms for bounded-degree linear hypergraphs [4, 8], for random 3-uniform 2-colorable hypergraphs [34], as well as for some special classes of graphs [14, 25, 27, 10].

In this paper, we consider the special class of *intersecting* hypergraphs, i.e., those in which every pair of edges have a non-empty intersection (also considered in [35]). While this may seem as a strong restriction at a first thought, the problem is still actually highly non-trivial. In fact, the case $k = 2$ is equivalent to the well-known Monotone Boolean Duality Testing, which is the problem of checking for a given pair of monotone CNF and DNF formulas if they represent the same monotone Boolean function [15, 35]. Determining the exact complexity of this duality testing problem is an outstanding open question, which has been referenced in a number of complexity theory retrospectives, e.g., [31, 32], and has been the subject of many papers, see, e.g., [6, 7, 13, 19, 15, 16, 17, 18, 21, 23, 22, 24, 28, 36].

[1] More precisely, there is no polynomial time algorithm unless NP$\subseteq$ DTIME($2^{\text{polylog}\, n}$).

Fredman and Khachiyan [21] gave an algorithm for solving this problem with running time $n^{o(\log n)}$, where n is the size of the input, thus providing strong evidence that this decision problem is unlikely to be NP-hard.

The reduction from Boolean Duality Testing to checking 2-colorability is essentially obtained by a construction from [35] which reduces the problem to checking if a monotone Boolean function given by its CNF representation is *self-dual*. However, almost all the known algorithms for solving Boolean duality testing cannot work directly with the self-duality (and hence the 2-colorabilty) problem, due to their recursive nature which results in subproblems that do not involve checking self-duality. The only algorithm we are aware of that works directly on the 2-colorability version is the one given in [22], but it yields weaker bounds $n^{O(\log n)}$ than those given in [21]. In this paper, we provide bounds that (almost) match those given in [21] and show that those can be in fact extended to the list-colorabilty version[2].

It is also worth mentioning that intersecting hypergraphs have been considered in [33, Section 2.4.1] (with a slight generalization), where it was shown that if such a hypergraph is 2-colorable then it is also list-colorable for any lists of size 2. It is not clear whether such result extends to the case $k > 2$.

2 Basic Notation and Main Result

Let $\mathcal{H} \subseteq 2^V$ be a hypegraph on a finite set V, $k \geq 2$ be a positive integer, and $\mathcal{L} : V \to 2^{[k]}$ be a mapping that assigns to each vertex $v \in V$ a non-empty list of *admissible colors* $\mathcal{L}(v) \subseteq [k] := \{1, \ldots, k\}$. An *$\mathcal{L}$-(list) coloring* of $\mathcal{H}$ is an assignment $\chi : V \to [k]$ of colors to the vertices of $\mathcal{H}$ such that $\chi(v) \in \mathcal{L}(v)$ for all $v \in V$. An $\mathcal{L}$-coloring is said to be *proper* if it results in no *monochromatic* edges, that is, if $|\chi(H)| \geq 2$, for all $H \in \mathcal{H}$, where $\chi(H) := \{\chi(v) : \ v \in H\}$.

A hypergraph $\mathcal{H}$ is said to be *intersecting* if

$$H \cap H' \neq \emptyset \text{ for all } H, \ H' \in \mathcal{H}. \tag{1}$$

In this paper, we are interested in the following problem:

> Proper-$\mathcal{L}$-Coloring: Given a hypergraph $\mathcal{H} \subseteq 2^V$ satisfying (1) and a mapping $\mathcal{L} : V \to 2^{[k]}$, either find a proper $\mathcal{L}$-coloring of $\mathcal{H}$, or declare that no such coloring exists.

We denote by $n := |V|$, $m := |\mathcal{H}|$, $\nu := \min_{v \in V} |\mathcal{L}(v)|$, $\rho := \max_{v \in V} |\mathcal{L}(v)|$, and $\kappa := \max_{u,v \in V, \ u \neq v} |\mathcal{L}(u) \cap \mathcal{L}(v)|$. We assume without loss of generality that $\nu \geq 2$.

For a set $S \subseteq V$, let $H_S := \{H \in \mathcal{H} : \ H \subseteq S\}$ be the subhypergraph of $\mathcal{H}$ induced by set S, $\mathcal{H}^S = \{H \cap S : \ H \in \mathcal{H}\}$ be the projection (or trace) of $\mathcal{H}$ into S, and $\mathcal{H}(S) := \{H \in \mathcal{H} : \ H \cap S \neq \emptyset\}$. For simplicity, we allow $\mathcal{H}^S$ to be a multi-hypergraph (some edges may be repeated). For $v \in V$, we define $\deg_{\mathcal{H}}(v) := |\{H \in \mathcal{H} : \ v \in H\}|$.

The main result of the paper is the following.

▶ **Theorem 1.** *Problem* Proper-$\mathcal{L}$-Coloring *can be solved in time* $(mn)^{o(k^2 \log(mn))}$.

[2] Note that is an intersecting hypergragh (with more than one edge) is trivially 3-colorable; so the generalization to k colors would only be interesting if we consider list-coloring.

In the following, we will consider partial $\mathcal{L}$-colorings $\chi : V \to [0:k] := \{0, 1, \ldots, k\}$ of $\mathcal{H}$, where $\chi(v) = 0$ is used to mean that the vertex v is not assigned any color yet; we say that such coloring is proper if no edge is monochromatic with this coloring. Given a proper partial $\mathcal{L}$-coloring χ of a hypergraph $\mathcal{H} \subseteq 2^V$, we will use the following notation: $V_0(\chi) := \{v \in V : \ \chi(v) = 0\}$ and $\mathcal{H}_i(\chi) := \{H \in \mathcal{H} : \ \chi(H) = \{0, i\}\}$ for $i \in [0:k]$, and shall simply write V_0 and $\mathcal{H}_i$ when χ is clear from the context. For $i \in [0:k]$, we write $\bar{\mathcal{H}}_i := \bigcup_{j \neq i} \mathcal{H}_j$. For a set $S \subseteq V$, we write $\bar{S} := V \setminus S$ and denote by $\chi[S]$ the restriction of χ on S. For two $\mathcal{L}$-colorings $\chi : S \to [k]$ and $\chi' : S' \to [k]$, where $S \cap S' = \emptyset$, we denote by $\chi'' := \chi \cup \chi' : S \cup S' \to [k]$ the k-coloring that assigns $\chi''(v) := \chi(v)$ for $v \in S$ and $\chi''(v) := \chi'(v)$ for $v \in S'$. If there is an $H \in \mathcal{H}$ such that $|H| \leq 1$, we shall assume that $\mathcal{H}$ is *not* properly $\mathcal{L}$-colorable for any $\mathcal{L} : V \to 2^{[k]}$. Also, by assumption, an empty hypergraph (that is, $\mathcal{H} = \emptyset$) is properly $\mathcal{L}$-colorable.

In the following two sections we give two algorithms for solving the problem. They are inspired by the two corresponding algorithms in [21] and can be thought of as generalizations. The first algorithm is simpler and exploits the idea of the existence of a high-degree vertex in any non-colorable instance. By considering all possible admissible colorings of such a vertex we can remove a large fraction of the edges and recurse on substantially smaller subproblems. Unfortunately, the degree of the high-degree vertex is only large enough to guarantee a bound of $m^{O(\log^2 m)}$ (assuming $k = O(1)$). The second algorithm is more complicated and considers both scenarios when there is a high-degree vertex and there are none (where now the threshold for "high" is actually higher). If there is no high-degree vertex, then we can find a "balanced-set" which contains a constant fraction of edges. Then a decomposition can be obtained based on this set.

3 Solving Proper-$\mathcal{L}$-Coloring in time $n^{O(k^3)} m^{O(k^2 \log^2 m)}$

We give two lemmas which show the existence of a high-degree vertex, unless the hypergraph is easily colorable.

▶ **Lemma 2.** *Let $\mathcal{H} \subseteq 2^V$ be a given hypergraph satisfying (1) of minimum edge-size 2, $\mathcal{L} : V \to 2^{[k]}$, and $\chi : V \to [0:k]$ be a proper partial $\mathcal{L}$-coloring of $\mathcal{H}$. Then either* (i) *there is a vertex $v \in V_0$ with $\deg_{\mathcal{H}_0}(v) \geq \frac{|\mathcal{H}_0|}{\log_\nu(m\kappa)}$, or* (ii) *an $\mathcal{L}$-coloring $\chi_0 : V_0 \to [k]$, such that $\chi[V \setminus V_0] \cup \chi_0$ is a proper $\mathcal{L}$-coloring of $\mathcal{H}$, can be found in $O(\rho|V_0|m)$ time.*

Proof. We use the probabilistic method [2]. Let $H_{\min}$ be an edge in $\bigcup_{i=0}^k \mathcal{H}_i^{V_0}$ of minimum size. Pick a random $\mathcal{L}$-coloring $\chi_0 : V_0 \to [k]$ by assigning, independently for each $v \in V_0$, $\chi_0(v) = i \in \mathcal{L}(v)$ with probability $\frac{1}{|\mathcal{L}(v)|}$. Then, for an edge $H \in \mathcal{H}_0$,

$$\Pr[H \text{ is monochromatic}] = |\bigcap_{v \in H} L(v)| \cdot \prod_{v \in H} \frac{1}{|\mathcal{L}(v)|} \leq \kappa \cdot \left(\frac{1}{\nu}\right)^{|H|},$$

and for $H \in \mathcal{H}_i$, $i \in [k]$,

$$\Pr[H \text{ is monochromatic}] \leq \prod_{v \in H \cap V_0} \frac{1}{|\mathcal{L}(v)|} \leq \left(\frac{1}{\nu}\right)^{|H \cap V_0|}.$$

It follows that

$$\begin{aligned} \mathbb{E}[\#\text{ monochromatic } H \in \mathcal{H}] &= \sum_{H \in \mathcal{H}} \Pr[H \text{ is monochromatic}] \\ &\leq \kappa \sum_{H \in \mathcal{H}_0} \left(\frac{1}{\nu}\right)^{|H|} + \sum_{i=1}^k \sum_{H \in \mathcal{H}_i} \left(\frac{1}{\nu}\right)^{|H \cap V_0|} \leq m\kappa \left(\frac{1}{\nu}\right)^{|H_{\min}|}. \end{aligned}$$

Thus if $m\kappa\left(\frac{1}{\nu}\right)^{|H_{\min}|} < 1$, then there is a proper $\mathcal{L}$-coloring $\chi' := \chi[V \setminus V_0] \cup \chi_0$ of $\mathcal{H}$, which can be found by the method of conditional expectations in time $O(\rho|V_0|m)$. Let us therefore assume for the rest of this proof that $|H_{\min}| \leq \log_\nu(m\kappa)$.

Let $v_{\max}$ be a vertex of maximizing $\deg_{\mathcal{H}_0}(v)$ over $v \in H_{\min}$. Then (1) implies that

$$|\mathcal{H}_0| = \left|\bigcup_{v\in H_{\min}} \{H \in \mathcal{H}_0 :\ v \in H\}\right| \leq \sum_{v\in H_{\min}} |\{H \in \mathcal{H}_0 :\ v \in H\}| = \sum_{v\in H_{\min}} \deg_{\mathcal{H}_0}(v)$$
$$\leq |H_{\min}| \deg_{\mathcal{H}_0}(v_{\max}).$$

Consequently, $\deg_{\mathcal{H}_0}(v_{\max}) \geq \frac{|\mathcal{H}_0|}{|H_{\min}|} \geq \frac{|\mathcal{H}_0|}{\log_\nu(m\kappa)}$. ◀

▶ **Lemma 3.** *Let $\mathcal{H} \subseteq 2^V$ be a given hypergraph satisfying (1) of minimum edge-size 2, $\mathcal{L} : V \to 2^{[k]}$ be a mapping, and $\chi : V \to [0 : k]$ be a proper partial $\mathcal{L}$-coloring of $\mathcal{H}$. Then either* (i) *there is a vertex $v \in V_0$ and $i, j \in [k]$, $j \neq i$, such that $\deg_{\mathcal{H}_i}(v) \geq \frac{|\mathcal{H}_i|}{\log_\nu m}$ and $\deg_{\mathcal{H}_j}(v) \geq 1$, or* (ii) *an $\mathcal{L}$-coloring $\chi_0 : V_0 \to [k]$, such that $\chi[V \setminus V_0] \cup \chi_0$ is a proper $\mathcal{L}$-coloring of $\mathcal{H}$, can be found in $O(\rho|V_0|m)$ time.*

Proof. Let $H_{\min}$ be an edge in $\bigcup_{i=1}^k \mathcal{H}_i^{V_0}$ of minimum size. Note that (1) implies:

$$\forall H \in \mathcal{H}_i :\ H \cap H' \cap V_0 \neq \emptyset \ \text{ for all } H' \in \bar{\mathcal{H}}_i, \tag{2}$$

since $\{i\} = \chi(H \setminus V_0) \neq \chi(H' \setminus V_0) = \{j\}$ for all $H \in \mathcal{H}_i$ and $H' \in \mathcal{H}_j$, for $i \neq j$.

If there is an $i \in [k]$ such that $\mathcal{H}_j = \emptyset$ for all $j \in [k] \setminus \{i\}$ then an $\mathcal{L}$-coloring satisfying (ii) can be found by choosing arbitrarily $\chi(v) \in \mathcal{L}(v) \setminus \{i\}$ for $v \in V_0$. Assume therefore that $\mathcal{H}_i \neq \emptyset$ for at least two distinct indices $i \in [k]$. Pick a random $\mathcal{L}$-coloring $\chi_0 : V_0 \to [k]$ by assigning, independently for each $v \in V_0$, $\chi(v) = i \in \mathcal{L}(v)$ with probability $\frac{1}{|\mathcal{L}(v)|}$. Then

$$\begin{aligned}\Pr[\exists i \in [k],\ H \in \mathcal{H}_i :\ \chi(H_i) = \{i\}] &\leq \sum_{i=1}^k \sum_{H\in\mathcal{H}_i} \Pr[\chi(H) = \{i\}] \\ &\leq \sum_{i=1}^k \sum_{H\in\mathcal{H}_i} \prod_{v\in H\cap V_0} \frac{1}{|\mathcal{L}(v)|} \leq m\left(\frac{1}{\nu}\right)^{|H_{\min}|}.\end{aligned}$$

Thus if $m\left(\frac{1}{\nu}\right)^{|H_{\min}|} < 1$, then there is an $\mathcal{L}$-coloring satisfying (ii), which can be found by the method of conditional expectations in time $O(\rho|V_0|m)$. Let us therefore assume for the rest of this proof that $|H_{\min}| \leq \log_\nu m$.

Let j be such that $H_{\min} \in \mathcal{H}_j^{V_0}$, and $v_{\max}$ be a vertex maximizing $\deg_{\bar{\mathcal{H}}_j}(v)$ over $v \in H_{\min}$. Then (2) implies that

$$|\bar{\mathcal{H}}_j| = \left|\bigcup_{v\in H_{\min}} \{H \in \bar{\mathcal{H}}_j :\ v \in H\}\right| \leq \sum_{v\in H_{\min}} |\{H \in \bar{\mathcal{H}}_j :\ v \in H\}| = \sum_{v\in H_{\min}} \deg_{\bar{\mathcal{H}}_j}(v)$$
$$\leq |H_{\min}| \deg_{\bar{\mathcal{H}}_j}(v_{\max}).$$

Consequently, $\sum_{i\neq j} \deg_{\mathcal{H}_i}(v_{\max}) = \deg_{\bar{\mathcal{H}}_j}(v_{\max}) \geq \frac{|\bar{\mathcal{H}}_j|}{|H_{\min}|} \geq \frac{|\bar{\mathcal{H}}_j|}{\log_\nu m} = \frac{\sum_{i\neq j}|\mathcal{H}_i|}{\log_\nu m}$, from which it follows that $\max_{i\neq j} \frac{\deg_{\mathcal{H}_i}(v_{\max})}{|\mathcal{H}_i|} \geq \frac{\sum_{i\neq j}\deg_{\mathcal{H}_i}(v_{\max})}{\sum_{i\neq j}|\mathcal{H}_i|} \geq \frac{1}{\log_\nu m}$. ◀

If the number of edges in each $\mathcal{H}_i$ is small, the problem is easily solvable in polynomial time.

▶ **Lemma 4.** *Given a hypergraph $\mathcal{H} \subseteq 2^V$ such that $\max_{i=0}^{k} |\mathcal{H}_i| \leq \delta$ or $|\{i : \mathcal{H}_i \neq \emptyset\}| = 1$, a mapping $\mathcal{L} : V \to 2^{[k]}$, and a proper partial $\mathcal{L}$-coloring $\chi : V \to [0 : k]$ of $\mathcal{H}$ such that $\mathcal{H}_0 = \emptyset$, there is a procedure* Proper-$\mathcal{L}$-Coloring-*simple$(\mathcal{H}, L, \chi)$ that checks if there is a proper $\mathcal{L}$-coloring of $\mathcal{H}$ extending χ, in time $O((|V_0|\rho)^{(k+1)\delta})$.*

Proof. If $\mathcal{H}_i \neq \emptyset$ for exactly one i, then assigning any color $j \neq i$ to the uncolored vertices yields a proper $\mathcal{L}$-coloring for $\mathcal{H}$. On the other hand, if $|\mathcal{H}_i| \leq \delta$ for all i, we can simply try all possibilities: for each edge $H \in \mathcal{H}_i$, for $i = 1, \ldots, k$ (resp., $H \in \mathcal{H}_0$), we choose a vertex $v \in H \cap V_0$ and a color for v among the colors in $\mathcal{L}(v) \setminus \{i\}$ (resp., two distinct vertices $v, v' \in H \cap V_0$ and two distinct colors $i \in \mathcal{L}(v)$ and $j \in \mathcal{L}(v')$). For each such choice, if the resulting coloring, combined with χ, is a proper partial $\mathcal{L}$-coloring for $\mathcal{H}$, then it can be extended to a proper $\mathcal{L}$-coloring by coloring any remaining uncolored vertices arbitrarily; otherwise, we conclude that no such coloring exists if we run out of choices. Since we have at most $(k+1)\delta$ edges in $\bigcup_{i=0}^{k} \mathcal{H}_i$, the total number of choices is at most $(|V_0|\rho)^{(k+1)\delta}$. ◀

The algorithm for solving Proper-$\mathcal{L}$-Coloring is given as Algorithm 1, which is called initially with $\chi \equiv 0$. The algorithm terminates either with a proper $\mathcal{L}$-coloring of $\mathcal{H}$, or with a partial $\mathcal{L}$-coloring with some unassigned vertices, in which case we conclude that no proper $\mathcal{L}$-coloring of $\mathcal{H}$ exists.

The algorithm proceeds in two phases. As long as there is an edge with no assigned colors, that is $|\mathcal{H}_0| \geq 1$, the algorithm is still in phase I; otherwise it proceeds to phase II. In a general step of phase I (resp. phase II), the algorithm picks a vertex v satisfying condition (i) of Lemma 2 (resp., Lemma 3) and iterates over all feasible assignments of colors to v, that result in no monochromatic edges; if no such v can be found, the algorithm concludes with a proper $\mathcal{L}$-coloring. In each iteration, any edge that becomes non-monochromatic is removed and the algorithm recurses on the updated sets of hypergraphs. If non of the recursive calls yields a feasible extension of the current proper partial $\mathcal{L}$-coloring χ, we unassign vertex v and return that there are no proper $\mathcal{L}$-colorings (line 11).

To analyze the running time of the algorithm, let us measure the "volume" of a subproblem with input $(\mathcal{H}, \mathcal{L}, \chi)$, in phase I by $\mu_1 = \mu_1(\mathcal{H}, \chi) := |\mathcal{H}_0(\chi)|$, and in phase II by

$$\mu_2 = \mu_2(\mathcal{H}, \chi) := |\{i \in [k] : |\mathcal{H}_i(\chi)| \geq 1\}| \cdot \prod_{i=1}^{k} \max\{|\mathcal{H}_i(\chi)|, 1\}. \tag{3}$$

The recursion stops when the volume $\mu_2(\mathcal{H})$ becomes sufficiently small, or an $\mathcal{L}$-coloring satisfying condition (ii) of Lemmas 2 or 3 is found.

Lemma 4 implies that problem Proper-$\mathcal{L}$-Coloring can be solved in time $O((\rho n)^{(k+1)\rho^2})$ if $m \leq \delta := \rho^2$. Algorithm Proper-$\mathcal{L}$-Coloring-A can be used to solve the problem in case $m > \delta$.

▶ **Lemma 5.** *Algorithm 1 solves problem* Proper-$\mathcal{L}$-Coloring *in time*

$$(\rho n)^{O(k\rho^2)} \rho^3 m^{O(k^2 \frac{\log^2 m}{\log \nu})}.$$

Proof. Let $\epsilon := \min\{\frac{1}{\log_\nu m}, \frac{1}{k}\}$, $\alpha = \frac{1}{1-\epsilon}$ and $\delta = \rho^2$. Note that $\delta \geq \alpha^2$ since $\epsilon \leq \frac{1}{k} \leq \frac{1}{2}$ and thus $\rho \geq 2 \geq \frac{1}{1-\epsilon} = \alpha$.

Consider the recursion tree $\mathbf{T}$ of the algorithm. Let $\mathbf{T}_1$ (reps., $\mathbf{T}_2$) be the subtree (resp., sub-forest) of $\mathbf{T}$ belonging to phase I (resp., phase II) of the algorithm. Note that $\mathbf{T}_2$ consists of maximal sub-trees of $\mathbf{T}$, each of which is rooted at a leaf in $\mathbf{T}_1$. Let $A_1(\mu)$ (resp., $A_2(\mu)$) be the total number of nodes in $\mathbf{T}_1$ (resp., $\mathbf{T}_2$) that result from a subproblem of volume μ.

Algorithm 1 PROPER-$\mathcal{L}$-COLORING-A$(\mathcal{H}, \mathcal{L}, \chi)$

Input: hypergraph $\mathcal{H} \subseteq 2^V$, a mapping $\mathcal{L} : V \to 2^{[k]}$, and a proper partial $\mathcal{L}$-coloring $\chi : V \to [0:k]$

Output: TRUE (resp,. FALSE) if a proper $\mathcal{L}$-coloring $\chi : V \to [k]$ of $\mathcal{H}$ is (resp., cannot be) found

1: $V_0 := V_0(\chi)$; $\mathcal{H}_i := \mathcal{H}_i(\chi)$ for $i \in [0:k]$
2: **if** $\mathcal{H}_0 \neq \emptyset$ **then** /* Phase I */
3: **if** there is $v \in V_0$ satisfying condition (i) of Lemma 2 **then**
4: **for** each $j \in \mathcal{L}(v)$ **do**
5: $\chi(v) := j$
6: **if** there is no $H \in \mathcal{H}$ such that $\chi(H) = \{j\}$ **then** /* if no edge becomes monochromatic */
7: $\mathcal{H}' := \mathcal{H} \setminus \bigcup_{i \in [k],\ i \neq j} \{H \in \mathcal{H}_i :\ j \in \chi(H)\}$ /* delete non-monochromatic edges */
8: **return** PROPER-$\mathcal{L}$-COLORING-A$(\mathcal{H}', \mathcal{L}, \chi)$
9: **end if**
10: **end for**
11: $\chi(v) := 0$; **return** FALSE
12: **else**
13: Let $\chi_0 : V_0 \to [k]$ be a coloring computed as in (ii) of Lemma 2
14: Set $\chi := \chi[V \setminus V_0] \cup \chi_0$ and stop /* A proper $\mathcal{L}$-coloring has been found */
15: **end if**
16: **else**/* Phase II */
17: **if** $\mu_2(\mathcal{H}, \chi) \leq \delta := \rho^2$ or $|\{i :\ \mathcal{H}_i \neq \emptyset\}| = 1$ **then**
18: **if** PROPER-$\mathcal{L}$-COLORING-simple$(\mathcal{H}, L, \chi)$ **then**
19: Stop /* A proper $\mathcal{L}$-coloring has been found */
20: **else**
21: **return** FALSE
22: **end if**
23: **end if**
24: **if** there is $v \in V$ satisfying condition (i) of Lemma 3 **then**
25: Same as in steps 4-11 of Phase I
26: **else**
27: Let $\chi_0 : V \to [k]$ be a coloring computed as in (ii) of Lemma 3
28: Set $\chi := \chi[V \setminus V_0] \cup \chi_0$ and stop /* A proper $\mathcal{L}$-coloring has been found */
29: **end if**
30: **end if**

▶ **Claim 6.** *The number of nodes in* $\mathbf{T}_1$ *is at most* $m^{\log \rho \cdot \log_\nu(m\kappa)+O(1)}$.

Proof. For a non-leaf node of $\mathbf{T}_1$, we have the recurrence:

$$A_1(\mu_1) \leq \rho \cdot A_1((1-\epsilon)\mu_1) + 1. \tag{4}$$

At leaves we have $\mu_1 = 0$. It follows that the depth $d(\mu_1)$ of the recursion subtree of a node (in $\mathbf{T}_1$) of volume μ_1 is at most $\log_{\frac{1}{1-\epsilon}} \mu_1 + 1$, and hence the total number of tree nodes N_1 is bounded by $\frac{\rho^{d(\mu_1)+1}-1}{\rho-1} \leq \mu_1^{\log_{\frac{1}{1-\epsilon}} \rho+2}$. Using $\mu_1 \leq m$, we get $N_1 = O(m^{\frac{\log \rho}{\log(1+1/\log_\nu(m\kappa))}}) = m^{\log \rho \cdot \log_\nu(m\kappa)+O(1)}$. ◀

▶ **Claim 7.** *The number of nodes in any sub-tree of* $\mathbf{T}_2$ *is at most* $m^{\log\rho\cdot\log_\nu(m\kappa)+O(1)}$.

Proof. Suppose that the algorithm proceeds to line 25 during the current recursive call corresponding to a subproblem of volume μ_2, and let $v \in V$ be the vertex chosen at step 24, and $i, j \in [k]$ be such that $i \neq j$, $\deg_{\mathcal{H}_i}(v) \geq \epsilon|\mathcal{H}_i|$ and $\deg_{\mathcal{H}_j}(v) \geq 1$. There are $|\mathcal{L}(v)|$ recursive calls that will be initiated from this point, corresponding to $\ell \in \mathcal{L}(v)$; consider the ℓth recursive call. If $\ell \neq i$ then setting $\chi(v) = \ell$ will result in deleting all the edges containing v from $\mathcal{H}_i$. Thus the new volume μ_2' will satisfy $\mu_2' \leq (1 - \frac{1}{\log_\nu m})\mu_2$ if $|\mathcal{H}_i| > 1$ and $\mu_2' \leq (1 - \frac{1}{k})\mu_2$ if $|\mathcal{H}_i| = 1$; in both cases, $\mu_2' \leq (1-\epsilon)\mu_2$. On the other hand, if $\ell = i$, then at least one edge in $\mathcal{H}_j$ will be deleted, yielding $\mu_2' \leq \mu_2 - 1$. Consequently we get the recurrence:

$$A_2(\mu_2) \leq (\rho - 1) \cdot A(\lfloor(1-\epsilon)\mu_2\rfloor) + A(\mu_2 - 1) + 1. \tag{5}$$

By the stopping criterion in line 17, we have $A_2(\mu_2) = 1$ for $\mu_2 \leq \delta$. We will prove by induction on $\mu_2 > \delta$ that $A_2(\mu_2) \leq C \cdot \mu_2^{\log_\alpha \mu_2}$, where $C := (2\delta + 1)$. We consider 3 cases:

Case 1. $\mu_2 - 1 \leq \delta$: Then $\lfloor(1-\epsilon)\mu_2\rfloor \leq \delta$ and (5) reduces to $A_2(\mu_2) \leq \rho + 1 < C$.

Case 2. $(1-\epsilon)\mu_2 \leq \delta$: Then (5) reduces to

$$A_2(\mu_2) \leq \rho + A_2(\mu_2 - 1).$$

Iterating we get $A_2(\mu_2) \leq r\rho + A_2(\mu_2 - r) \leq r\rho + 1$, for $r = \mu_2 - \delta \leq \frac{\epsilon}{1-\epsilon}\delta$. Thus, $A_2(\mu_2) \leq \frac{\epsilon}{1-\epsilon}\delta\rho + 1 \leq \frac{1}{k-1}\rho\delta + 1 \leq C$.

Case 3. $\mu_2 - 1 > \delta$ and $(1-\epsilon)\mu_2 > \delta$: We apply induction:

$$\begin{aligned}
A_2(\mu_2) &\leq C(\rho-1)\cdot((1-\epsilon)\mu_2)^{\log_\alpha((1-\epsilon)\mu_2)} + C(\mu_2-1)^{\log_\alpha(\mu_2-1)} + 1\\
&\leq C\frac{\rho-1}{(1-\epsilon)\mu_2}\cdot\frac{1}{\mu_2}\cdot\mu_2^{\log_\alpha\mu_2} + C(\mu_2-1)^{\log_\alpha\mu_2} + 1\\
&= C\cdot\mu_2^{\log_\alpha\mu_2}\left(\frac{\rho-1}{\delta\mu_2} + \left(1-\frac{1}{\mu_2}\right)^{\log_\alpha\mu_2} + \frac{1}{C\cdot\mu_2^{\log_\alpha\mu_2}}\right)\\
&\leq C\cdot\mu_2^{\log_\alpha\mu_2}\left(\frac{1}{\mu_2} + \left(1-\frac{1}{\mu_2}\right)^2 + \frac{1}{\mu_2^2}\right) \quad \text{(since } \mu_2 \geq \delta \geq \alpha^2 \text{ and hence } \log_\alpha\mu_2 \geq 2\text{)}\\
&\leq C\cdot\mu_2^{\log_\alpha\mu_2} \quad \text{(since } \mu_2 \geq \delta > 2\text{).}
\end{aligned}$$

Using the bound

$$\mu_2(\mathcal{H},\chi) \leq k\cdot\prod_{i=1}^k|\mathcal{H}_i| \leq k\cdot\left(\frac{\sum_{i=1}^k|\mathcal{H}_i|}{k}\right)^k \leq k\cdot\left(\frac{m}{k}\right)^k,$$

we get the claim. ◀

Putting these Claims 6 and 7 together, and noting that at internal nodes the running time is $O(nm\rho)$, and that the roots of the maximal sub-trees in $\mathbf{T}_2$ are the leaves of $\mathbf{T}_1$, the lemma follows. ◀

4 Solving Proper-$\mathcal{L}$-Coloring in time $(nm)^{o(k^2\log(nm))}$

For a hypergraph $\mathcal{H} \subseteq 2^V$ and a positive number $\epsilon \in (0,1)$, denote by $T(\mathcal{H},\epsilon)$ the subset $\{v \in V :\ \deg_{\mathcal{H}}(v) > \epsilon|\mathcal{H}|\}$ of "high" degree vertices in $\mathcal{H}$. Given $\epsilon', \epsilon'' \in (0,1)$, let us call an (ϵ',ϵ'')-*balanced set* with respect to $\mathcal{H}$, any set $S \subseteq V$ such that $\epsilon'|\mathcal{H}| \leq |\mathcal{H}_S| \leq \epsilon''|\mathcal{H}|$.

▶ **Lemma 8** ([19]). *Let $\epsilon_1, \epsilon_2 \in (0,1)$ be two given numbers such that, $\epsilon_1 < \epsilon_2$ and $T = T(\mathcal{H},\epsilon_1)$ satisfies $|\mathcal{H}_T| \leq (1-\epsilon_2)|\mathcal{H}|$. Then there exists a $(1-\epsilon_2, 1-(\epsilon_2-\epsilon_1))$-balanced set $S \supseteq T$ with respect to $\mathcal{H}$. Such a set S can be found in $O(nm)$ time.*

▶ **Lemma 9.** *Let $\mathcal{H} \subseteq 2^V$ be a hypergraph satisfying (1), $\mathcal{L} : V \to 2^{[k]}$ be a mapping, $\chi : V \to [0:k]$ be a proper partial $\mathcal{L}$-coloring of $\mathcal{H}$, and $S \subseteq V_0$ be a given set of vertices such that $(\mathcal{H}_0)_S \neq \emptyset$. Then, χ is extendable to a proper $\mathcal{L}$-coloring of $\mathcal{H}$ if and only if either*

$$\chi \text{ is extendable to a proper } \mathcal{L}\text{-coloring for } \bar{\mathcal{H}}_0 \cup \mathcal{H}_0^S, \text{ or} \tag{6}$$

$$\exists Y \in \mathcal{H}_0^S \setminus (\mathcal{H}_0)_S,\ j \in \bigcap_{v\in Y} \mathcal{L}(v) :\ \chi \text{ is extendable to a proper } \mathcal{L}\text{-coloring } \chi' \text{ for } \mathcal{H} \text{ such that } \chi'(Y) = \{j\}. \tag{7}$$

Proof. Suppose that χ is extendable to a proper $\mathcal{L}$-coloring χ' for $\mathcal{H}$. If (6) is not satisfied then (since $\emptyset \notin \mathcal{H}_0^S$ by (1)) there is a $Y \in \mathcal{H}_0^S \setminus (\mathcal{H}_0)_S$, such that (in any proper extension χ' of χ) $\chi'(Y) = \{j\}$, for some $j \in \bigcap_{v\in Y} \mathcal{L}(v)$, and hence (7) is satisfied.

Conversely, if either (6) or (7) holds then there is an $\mathcal{L}$-coloring extension χ' of χ that properly colors $\mathcal{H}$. ◀

▶ **Lemma 10.** *Let $\mathcal{H} \subseteq 2^V$ be a hypergraph satisfying (1), $\mathcal{L} : V \to 2^{[k]}$ be a mapping, $\chi : V \to [0:k]$ be a proper partial $\mathcal{L}$-coloring of $\mathcal{H}$, and $S \subset V_0$ be a given set of vertices such that, for some $i \in [k]$, $(\mathcal{H}_i)_{S\cup(V\setminus V_0)} \neq \emptyset$. Then χ is extendable to a proper $\mathcal{L}$-coloring of $\mathcal{H}$ if and only if either*

$$\chi \text{ is extendable to a proper } \mathcal{L}\text{-coloring for } \bar{\mathcal{H}}_i^{S\cup(V\setminus V_0)} \cup (\mathcal{H}_i)_{S\cup(V\setminus V_0)}, \text{ or} \tag{8}$$

$$\exists j \neq i,\ Y \in \mathcal{H}_j^{S\cup(V\setminus V_0)} \setminus (\mathcal{H}_j)_{S\cup(V\setminus V_0)} :\ j \in \bigcap_{v\in Y} \mathcal{L}(v),\ \chi \text{ is extendable to a proper } \mathcal{L}\text{-coloring } \chi' \text{ for } \mathcal{H} \text{ such that } \chi'(Y) = \{j\}. \tag{9}$$

Proof. Suppose that χ is extendable to a proper $\mathcal{L}$-coloring χ' for $\mathcal{H}$. If (8) is not satisfied then (since $\emptyset \notin \bar{\mathcal{H}}_i^{S\cup(V\setminus V_0)}$ by(1)) there is a $Y \in \mathcal{H}_j^{S\cup(V\setminus V_0)}$ for some $j \neq i$, such that $\chi'(Y) = \{j\}$ and $j \in \bigcap_{v\in Y} \mathcal{L}(v)$, and hence (9) is satisfied.

Conversely, suppose that either (8) or (9) holds. If (9) is satisfied then χ' is a proper $\mathcal{L}$-coloring of $\mathcal{H}$ which extends χ. On the other hand, if (8) holds then there is an $\mathcal{L}$-coloring $\chi' : S \cup (V \setminus V_0) \to [k]$ such that $|\chi(H)| \geq 2$ for all $H \in \bar{\mathcal{H}}_i^{S\cup(V\setminus V_0)} \cup (\mathcal{H}_i)_{S\cup(V\setminus V_0)}$. Then χ' can be extended to a proper $\mathcal{L}$-coloring for $\mathcal{H}$ by setting $\chi'(v) \in \mathcal{L}(v) \setminus \{i\}$ arbitrarily for $v \in V_0 \setminus S$ (as $H \cap (V_0 \setminus S) \neq \emptyset$ for all $H \in \mathcal{H}_i \setminus (\mathcal{H}_i)_{S\cup(V\setminus V_0)}$). ◀

▶ **Lemma 11.** *Let $\mathcal{H} \subseteq 2^V$ be a hypergraph satisfying (1), $\mathcal{L} : V \to 2^{[k]}$ be a mapping, $\chi : V \to [0:k]$ be a proper partial $\mathcal{L}$-coloring of $\mathcal{H}$ such that $\mathcal{H}_i \neq \emptyset$ for at least two i's, and $\epsilon_1, \epsilon_2 \in (0,1)$ be two given numbers such that, $\epsilon_1 < \epsilon_2$. Then either* (i) *there is $v \in V_0$ and $i \neq j$ such that $\deg_{\mathcal{H}_i}(v) \geq \epsilon_1|\mathcal{H}_i|$ and $\deg_{\mathcal{H}_j}(v) \geq \epsilon_1|\mathcal{H}_j|$, or* (ii) *there is a $(1-\epsilon_2, 1-(\epsilon_2-\epsilon_1))$-balanced set $S \subseteq V_0$ with respect to $\mathcal{H}_j^{V_0}$ for some $j \in [k]$.*

Proof. For any $i \neq j$ such that $\mathcal{H}_i \neq \emptyset$ and $\mathcal{H}_i \neq \emptyset$, let $T_i := T(\mathcal{H}_i^{V_0}, \epsilon_1)$ and $T_j := T(\mathcal{H}_j^{V_0}, \epsilon_1)$. If $T_i \cap T_j \neq \emptyset$ then any v in this intersection will satisfy (i). Otherwise, (1) implies that either $(\mathcal{H}_i^{V_0})_{T_i} = \emptyset$ or $(\mathcal{H}_j^{V_0})_{T_j} = \emptyset$, in which case a $(1-\epsilon_2, 1-(\epsilon_2-\epsilon_1))$-balanced set with respect to $\mathcal{H}_i^{V_0}$ or $\mathcal{H}_j^{V_0}$, respectively, can be obtained by Lemma 8. ◀

Algorithm 2 is more sophisticated than Algorithm 1 as it does not require the existence of a large-degree vertex, but uses more complicated decomposition rules, given by lemmas 9 and 10. As before, the algorithm proceeds in two phases. As long as there is a large "volume" of edges with no assigned colors, that is $|\mathcal{H}_0| \sum_{H \in \mathcal{H}_0} |H| \geq \delta$, the algorithm is still in phase I; otherwise it proceeds to phase II. In a general step of phase I (resp., phase II), the algorithm tries, in step 3 (resp., step 35), to find a vertex v of large degree in $\mathcal{H}_0$ (resp., in $\mathcal{H}_i$ and $\mathcal{H}_j$ for some $i \neq j$) and iterates over all feasible assignments of colors to v, that result in no monochromatic edges; if no such v can be found then Lemma 8 guarantees the existence of a $(1-\epsilon_2, 1-(\epsilon_2-\epsilon_1))$-balanced set with respect to $\mathcal{H}_0$ (resp., with respect to either $\mathcal{H}_i^{V_0}$ or $\mathcal{H}_j^{V_0}$ as in Lemma 11), which is found in step 13 (resp, 38). Lemma 9 (resp., Lemma 10) then reduces the problem in the latter case to checking (6) and (7) (resp., (8) and (9)), which is done in steps 14, and 17-23 (resp., in steps 39, and 43-45), respectively. If none of the recursive calls yields a feasible extension of the current proper partial $\mathcal{L}$-coloring χ, we return that there are no proper $\mathcal{L}$-colorings (lines 11, 24, 32 and 46).

To analyze the running time of the algorithm, let us measure the volume of a subproblem in phase I by $\mu_2 = \mu_2(\mathcal{H}, \chi) = |\mathcal{H}_0| \sum_{H \in \mathcal{H}_0} |H|$, and in phase II by $\mu_2 = \mu_2(\mathcal{H}, \chi)$ given by (3). Phase II (and hence the recursion) stops when $\mu_2(\mathcal{H}, \chi) \leq \delta$, or an $\mathcal{L}$-coloring has been found.

Given a subproblem of volume μ, let $\epsilon(\mu) := \frac{\ln(e\rho)}{\xi(\mu)}$, where $\xi(\mu)$ is the unique positive root of the equation:

$$\left(\frac{\xi(\mu)}{2\ln(e\rho)}\right)^{\xi(\mu)} = \mu^2. \tag{10}$$

Note that (for constant ρ) $\chi(\mu) \approx O\left(\frac{\log \mu}{\log\log \mu}\right)$. We set $\delta \geq 2\rho$ such that $\xi(\delta) \geq 2k\ln(e\rho)$. Note that $\xi(\mu) \geq 2$ and $\epsilon(\mu) \leq \frac{1}{2k}$, for $\mu \geq \delta$. We use in the algorithm: $\epsilon_1(\mu) := \epsilon(\mu)$ and $\epsilon_2(\mu) := 2\epsilon(\mu)$.

▶ **Lemma 12.** *Algorithm 2 solves problem* Proper-$\mathcal{L}$-Coloring *in time* $(mn)^{o(k^2\log(mn))}$.

Consider the recursion tree $\mathbf{T}$ of the algorithm. Let $\mathbf{T}_1$ (reps., $\mathbf{T}_2$) be the subtree (resp., sub-forest) of $\mathbf{T}$ belonging to phase I (resp., phase II) of the algorithm. Let $B_1(\mu)$ (resp., $B_2(\mu)$) be the total number of nodes in $\mathbf{T}_1$ (resp., $\mathbf{T}_2$) that result from a subproblem of volume μ. The lemma follows from the following two claims whose proofs are given in the appendix.

▶ **Claim 13.** *The number of nodes in* $\mathbf{T}_1$ *is at most* $(\delta(\rho\delta+1)+1)(m^2n)^{\xi(m^2n)}$.

Proof. If there is a vertex $v \in V_0$ such that $\deg_{\mathcal{H}_0}(v) \geq \epsilon_1|\mathcal{H}_0|$ then the algorithm proceeds with steps 4-11 and we get the recurrence:

$$B_1(\mu_1) \leq \rho \cdot B_1(\lfloor(1-\epsilon_1)\mu_1\rfloor) + 1, \tag{11}$$

since we recurse in step 8 on a hypergraph $\mathcal{H}'$ that excludes all the edges containing v from $\mathcal{H}'_0$. On the other hand, if no such v can be found then Lemma 8 implies that there is a $(1-\epsilon_2, 1-(\epsilon_2-\epsilon_1))$-balanced set S, with respect to some $\mathcal{H}_0$, which is found in

Algorithm 2 PROPER-$\mathcal{L}$-COLORING-B$(\mathcal{H}, \mathcal{L}, \chi)$

Input: hypergraph $\mathcal{H} \subseteq 2^V$, a mapping $\mathcal{L} : V \to 2^{[k]}$, and a proper partial $\mathcal{L}$-coloring $\chi : V \to [0:k]$
Output: TRUE (resp,. FALSE) if a proper $\mathcal{L}$-coloring $\chi : V \to [k]$ of $\mathcal{H}$ is (resp., cannot be) found

1: $V_0 := V_0(\chi)$; $\mathcal{H}_i := \mathcal{H}_i(\chi)$ for $i \in [0:k]$
2: **if** $\mu_1 := \mu_1(\mathcal{H}, \chi)| > \delta$ **then** /* Phase I */
3: **if** there is $v \in V_0$ such that $\deg_{\mathcal{H}_0}(v) \geq \epsilon_1(\mu_1)|\mathcal{H}_0|$ **then**
4: **for** each $j \in \mathcal{L}(v)$ **do**
5: $\chi(v) := j$
6: **if** there is no $H \in \mathcal{H}$ such that $\chi(H) = \{j\}$ **then** /* if no edge becomes monochromatic */
7: $\mathcal{H}' := \mathcal{H} \setminus \bigcup_{i\in[k],\ i\neq j}\{H \in \mathcal{H}_i :\ j \in \chi(H)\}$ /* delete non-monochromatic edges */
8: **return** PROPER-$\mathcal{L}$-COLORING-B$(\mathcal{H}', \mathcal{L}, \chi)$
9: **end if**
10: **end for**
11: $\chi(v) := 0$; **return** FALSE
12: **else**
13: Let S be a $(1-\epsilon_2, 1-(\epsilon_2(\mu_1)-\epsilon_1(\mu_1)))$-balanced set computed as in Lemma 8 w.r.t $\mathcal{H}_0$
14: **if** PROPER-$\mathcal{L}$-COLORING-B$(\bar{\mathcal{H}}_0 \cup \mathcal{H}_0^S, \mathcal{L}, \chi)$ **then**
15: stop /* A proper $\mathcal{L}$-coloring has been found */
16: **else**
17: **for** each $Y \in \mathcal{H}_0^S \setminus (\mathcal{H}_0)_S$ and $j \in \bigcap_{v\in Y} \mathcal{L}(v)$ **do**
18: $\chi(Y) := \{j\}$
19: **if** there is no $H \in \mathcal{H}$ such that $\chi(H) = \{j\}$ **then** /* if no edge becomes monochromatic */
20: $\mathcal{H}' := \mathcal{H} \setminus \bigcup_{i\in[k],\ i\neq j}\{H \in \mathcal{H}_i :\ j \in \chi(H)\}$ /* delete non-monochromatic edges */
21: **return** PROPER-$\mathcal{L}$-COLORING-B$(\mathcal{H}', \mathcal{L}, \chi)$
22: **end if**
23: **end for**
24: $\chi(Y) := \{0\}$; **return** FALSE
25: **end if**
26: **end if**
27: **else**/* Phase II */
28: **if** $\mu_2 := \mu_2(\mathcal{H}, \chi) \leq \delta$ or $|\{i :\ \mathcal{H}_i \neq \emptyset\}| = 1$ **then**
29: **if** PROPER-$\mathcal{L}$-COLORING-simple$(\mathcal{H}, L, \chi)$ **then**
30: Stop /* A proper $\mathcal{L}$-coloring has been found */
31: **else**
32: **return** FALSE
33: **end if**
34: **else**
35: **if** there is $v \in V_0$ and $i \neq j$ such that $\deg_{\mathcal{H}_i}(v) \geq \epsilon_1(\mu_2)|\mathcal{H}_i|$ and $\deg_{\mathcal{H}_j}(v) \geq \epsilon_1|\mathcal{H}_j|$ **then**
36: Same as in steps 4-11 of Phase I
37: **else**
38: Let S be a $(1-\epsilon_2, 1-(\epsilon_2(\mu_2)-\epsilon_1(\mu_2)))$-balanced set computed as in Lemma 8 w.r.t $\mathcal{H}_i^{V_0}$ for some $i \in [k]$
39: **if** PROPER-$\mathcal{L}$-COLORING-B$(\bar{\mathcal{H}}_i^{S\cup(V\setminus V_0)} \cup (\mathcal{H}_i)_{S\cup(V\setminus V_0)}, \mathcal{L}, \chi)$ **then**
40: Set $\chi(v) \in \mathcal{L}(v) \setminus \{i\}$ arbitrarily for $v \in V_0 \setminus S$
41: stop /* A proper $\mathcal{L}$-coloring has been found */
42: **else**
43: **for** each $j \neq i$ and $Y \in \mathcal{H}_j^{S\cup(V\setminus V_0)} \setminus (\mathcal{H}_j)_{S\cup(V\setminus V_0)}$ s.t. $j \in \bigcap_{v\in Y} \mathcal{L}(v)$ **do**
44: Same as in steps 18-22 of Phase I
45: **end for**
46: $\chi(Y) := \{0\}$; **return** FALSE
47: **end if**
48: **end if**
49: **end if**
50: **end if**

step 13. Then we apply Lemma 9 which reduces the problem to one recursive call on the hypergraph $\bar{\mathcal{H}}_0 \cup \mathcal{H}_0^S$ in step 14, and at most $|\mathcal{H}_0^S \setminus (\mathcal{H}_0)_S|$ recursive calls (in step 21) on the hypergraphs obtained by fixing the color of one set $Y \in \mathcal{H}_0^S \setminus (\mathcal{H}_0)_S$. Note that S satisfies: $(1-\epsilon_2)|\mathcal{H}_0| \leq |(\mathcal{H}_0)_S \leq |(1-(\epsilon_2-\epsilon_1))|\mathcal{H}_0|$. In particular, there is an $H \in \mathcal{H}_0$

such that $H \setminus S \neq \emptyset$. Hence, in step 14, we recurse on the hypergraph $\bar{\mathcal{H}}_0 \cup \mathcal{H}_0^S$ which excludes at least one vertex from H. Moreover, in step 21, we recurse on a hypergraph $\mathcal{H}'$ that has $\mathcal{H}'_0 \subseteq \mathcal{H}_0 \setminus (\mathcal{H}_0)_S$, as all edges in $(\mathcal{H}_0)_S$ have non-empty intersections with the set $Y \in \mathcal{H}_0^S \setminus (\mathcal{H}_0)_S$ in the current iteration of the loop in step 17, all vertices of which are assigned color j. Since $|\mathcal{H}_0 \setminus (\mathcal{H}_0)_S| \leq \epsilon_2 |\mathcal{H}_0|$, we get the recurrence:

$$B_1(\mu_1) \leq B_1(\mu_1 - 1) + \rho\epsilon_2\mu_1 \cdot B_1(\lfloor \epsilon_2\mu_1 \rfloor) + 1. \tag{12}$$

By the termination condition of phase I (in line 2), we have $B_1(\mu_1) = 1$ for $\mu_1 \leq \delta$. We will prove by induction on $\mu_1 \geq \delta$ that $B_1(\mu_1) \leq C \cdot \mu_1^{\xi(\mu_1)}$, for $C := \delta(\rho\delta + 1) + 1$.

Let us consider first recurrence (11). If $(1-\epsilon_1)\mu_1 \leq \delta$ then we get $B_1(\mu_1) \leq \rho + 1 < C$. Otherwise, we apply induction to get

$$\begin{aligned}
B_1(\mu_1) &\leq C \cdot \rho((1-\epsilon_1)\mu_1)^{\xi(\mu_1)} + 1 \leq C \cdot \mu_1^{\xi(\mu_1)} \left(\rho(1-\epsilon_1)^{\chi(\mu_1)} + \frac{1}{\mu_1^{\xi(\mu_1)}} \right) \\
&\leq C \cdot \mu_1^{\xi(\mu_1)} \left(\rho e^{-\epsilon_1 \xi(\mu_1)} + \frac{1}{\mu_1^{\xi(\mu_1)}} \right) \\
&\leq C \cdot \mu_1^{\xi(\mu_1)} \left(\frac{1}{e} + \frac{1}{4} \right) < C \cdot \mu_1^{\xi(\mu_1)}, \quad (\text{as } \epsilon_1 \xi(\mu_1) = \ln(e\rho) \text{ and } \xi(\mu_1) \geq 2 \text{ for } \mu_1 \geq \delta).
\end{aligned}$$

Let us consider next recurrence (12). We consider 3 cases:

Case 1. $\mu_1 - 1 \leq \delta$: Then $\lfloor \epsilon_2\mu_1 \rfloor \leq \delta$ and (12) reduces to $B_1(\mu_1) \leq \rho\epsilon_2\mu_1 + 2 \leq \rho(\delta+1)+2 < C$.

Case 2. $\epsilon_2\mu_1 \leq \delta$: Then (12) reduces to

$$B_1(\mu_1) \leq B_1(\mu_1 - 1) + \rho\epsilon_2\mu_1 + 1 \leq B_1(\mu_1 - 1) + \rho\delta + 1.$$

Iterating we get $B_1(\mu_1) \leq B_1(\mu_1 - r) + r(\rho\delta + 1) \leq r(\rho\delta + 1) + 1$, for $r = \mu_1 - \delta \leq \left(\frac{1}{\epsilon_2} - 1\right)\delta$. Thus, $B_1(\mu_1) \leq \left(\frac{1}{\epsilon_2} - 1\right)\delta(\rho\delta + 1) + 1$. As $\epsilon_2 = 2\epsilon(\mu_1) = \frac{2\ln(e\rho)}{\xi(\mu_1)}$, we get $B_1(\mu_1) \leq \left(\frac{\xi(\mu_1)}{2\ln(e\rho)} - 1\right)\delta(\rho\delta + 1) + 1 \leq (\mu_1 - 1)\,\delta(\rho\delta + 1) + 1 \leq C\mu_1^{\xi(\mu_1)}$, for $\mu_1 \geq \delta$.

Case 3. $\mu_1 - 1 > \delta$ and $\epsilon_2\mu_1 > \delta$: We apply induction:

$$\begin{aligned}
B_1(\mu_1) &\leq C(\mu_1 - 1)^{\xi(\mu_1)} + C \cdot \rho\epsilon_2\mu_1(\epsilon_2\mu_1)^{\xi(\mu_2)} + 1 \\
&\leq C \cdot \mu_1^{\xi(\mu_1)} \left(\left(1 - \frac{1}{\mu_1}\right)^{\xi(\mu_1)} + \rho\epsilon_2\mu_1\epsilon_2^{\xi(\mu_1)} + \frac{1}{\mu_1^{\xi(\mu_1)}} \right) \\
&\leq C \cdot \mu_1^{\xi(\mu_1)} \left(\left(1 - \frac{1}{\mu_1}\right)^{\xi(\mu_1)} + \frac{\rho\epsilon_2}{\mu_1} + \frac{1}{\mu_1^{\xi(\mu_1)}} \right) \quad (\text{since } \epsilon_2^{\xi(\mu_1)} = \frac{1}{\mu_1^2} \text{ by (10)}) \\
&\leq C \cdot \mu_1^{\xi(\mu_1)} \left(\left(1 - \frac{1}{\mu_1}\right)^{2} + \frac{1}{\mu_1} + \frac{1}{\mu_1^2} \right) \quad (\text{since } \epsilon_2 \leq \frac{1}{k} \text{ and } \xi(\mu_1) \geq 2 \text{ for } \mu_1 \geq \delta) \\
&\leq C \cdot \mu_1^{\xi(\mu_1)} \quad (\text{since } \mu_2 \geq \delta > 2).
\end{aligned}$$

Using $\mu_1(\mathcal{H}, \chi) \leq m^2 n$, we get the claim. ◀

▶ **Claim 14.** *The number of nodes in any sub-tree of* $\mathbf{T}_2$ *is at most*

$$(\delta(\rho\delta + 1) + 1)(m^k/k^{k-1})^{\xi(m^k/k^{k-1})}.$$

Proof. If $\mu_2(\mathcal{H},\chi) \leq \delta$ (and already $\mu_1(\mathcal{H},\chi) \leq \delta$), then Lemma 4 implies that problem Proper-$\mathcal{L}$-Coloring can be solved in time $O((\rho n)^{(k+1)\delta})$, as $m \leq \delta$. If there is $v \in V_0$ and $i \neq j$ such that $\deg_{\mathcal{H}_i}(v) \geq \epsilon_1|\mathcal{H}_i|$ and $\deg_{\mathcal{H}_j}(v) \geq \epsilon_1|\mathcal{H}_j|$ then the algorithm proceeds similar to steps 4-11 and we get the recurrence:

$$B_2(\mu_2) \leq \rho \cdot B_2(\lfloor(1-\epsilon_1)\mu_2\rfloor) + 1, \tag{13}$$

since we recurse (in the step similar to step 8) on a hypergraph $\mathcal{H}'$ that excludes either all the edges containing v from $\mathcal{H}'_i$, if we set the color of v to j, or all those containing v from $\mathcal{H}'_j$ if we set the color of v to i (or both, if we set the color of v to $\ell \notin \{i,j\}$).

On the other hand, if no such v can be found then Lemma 11 implies that there is a $(1-\epsilon_2, 1-(\epsilon_2-\epsilon_1))$-balanced set S, with respect to some $\mathcal{H}_i$, which is found in step 38. Then we apply Lemma 10 which reduces the problem to one recursive call on the hypergraph $\bar{\mathcal{H}}_i^{S\cup(V\setminus V_0)} \cup (\mathcal{H}_i)_{S\cup(V\setminus V_0)}$ in step 39, and at most $\sum_{j\neq i} |\mathcal{H}_j^{S\cup(V\setminus V_0)} \setminus (\mathcal{H}_j)_{S\cup(V\setminus V_0)}|$ recursive calls on the hypergraphs obtained by fixing the color of one set $Y \in \mathcal{H}_j^{S\cup(V\setminus V_0)} \setminus (\mathcal{H}_j)_{S\cup(V\setminus V_0)}$. As S satisfies: $(1-\epsilon_2)|\mathcal{H}_i| \leq |(\mathcal{H}_i)_{S\cup(V\setminus V_0)}| \leq (1-(\epsilon_2-\epsilon_1))|\mathcal{H}_i|$, in step 39 we recurse on the hypergraph $\mathcal{H}' := \bar{\mathcal{H}}_i^{S\cup(V\setminus V_0)} \cup (\mathcal{H}_i)_{S\cup(V\setminus V_0)}$ with $\mu_2(\mathcal{H}',\chi) \leq (1-(\epsilon_2-\epsilon_1))\mu_2(\mathcal{H},\chi)$. Moreover, for each Y satisfying the condition in step 43, we recurse on a hypergraph $\mathcal{H}'$ that has $\mathcal{H}'_i \subseteq \mathcal{H}_i \setminus (\mathcal{H}_i)_{S\cup(V\setminus V_0)}$, as all edges in $(\mathcal{H}_i)_{S\cup(V\setminus V_0)}$ have non-empty intersections with the set Y, all vertices of which are assigned color $j \neq i$. Since $|\mathcal{H}_i \setminus (\mathcal{H}_i)_{S\cup(V\setminus V_0)}| \leq \epsilon_2|\mathcal{H}_i|$, we get the recurrence:

$$B_2(\mu_2) \leq B_2(\lfloor(1-(\epsilon_2-\epsilon_1))\mu_2\rfloor) + \rho\mu_2 \cdot B_2(\lfloor\epsilon_2\mu_2\rfloor) + 1. \tag{14}$$

By the stopping criterion in line 28, we have $B_2(\mu_2) = 1$ for $\mu_2 \leq \delta$. We will prove by induction on $\mu_2 \geq \delta$ that $B_2(\mu_2) \leq C \cdot \mu_2^{\xi(\mu_2)}$, for $C := \delta(\rho\delta+1)+1$.

As recurrence (13) is the same as (11), we need only to consider recurrence (14). We consider 3 cases:

Case 1. $(1-(\epsilon_2-\epsilon_1))\mu_2 \leq \delta$: Then $\epsilon_2\mu_2 \leq \frac{2\epsilon(\mu_2)}{1-\epsilon(\mu_2)}\delta \leq \frac{2}{2k-1}\delta < \delta$ for $\mu_2 \geq \delta$ (recall that $\epsilon(\mu_2) \leq \frac{1}{2k}$ for $\mu_2 \geq \delta$), and hence (13) reduces to $B_2(\mu_2) \leq \rho\mu_2 + 2 \leq \frac{\rho\delta}{1-\epsilon(\mu_2)} + 2 \leq \frac{2k\rho\delta}{2k-1} + 2 < 2(\rho\delta+1) < C$.

Case 2. $\epsilon_2\mu_2 \leq \delta$: Then (12) reduces to

$$B_2(\mu_2) \leq B_2((1-\epsilon(\mu_2))\mu_2) + \rho\epsilon_2\mu_2 + 1 \leq B_2((1-\epsilon(\mu_2))\mu_2) + \rho\delta + 1.$$

Iterating we get $B_2(\mu_2) \leq B_2(\mu_2(1-\epsilon(\mu_2))^r) + r(\rho\delta+1) \leq r(\rho\delta+1)+1$, for $r = \frac{\ln(\mu_2/\delta)}{\epsilon(\mu_2)} \leq \frac{\ln(1/\epsilon(\mu_2))}{\epsilon(\mu_2)}$. Thus, $B_2(\mu_2) \leq \left(\frac{\ln(1/\epsilon(\mu_2))}{\epsilon(\mu_2)}\right)(\rho\delta+1)+1$. As $\frac{\ln(1/\epsilon(\mu_2))}{\epsilon(\mu_2)} \leq \delta\mu_2^{\xi(\mu_2)}$ for $\mu_2 \geq \delta \geq 2$ (since $\left(\frac{1}{2\epsilon(\mu_2)}\right)^{\xi(\mu_2)} = \mu_2^2$ by (10)), we get $B_2(\mu_2) \leq \delta(\rho\delta+1)+1 \leq C\mu_1^{\xi(\mu_2)}$.

Case 3. $(1-(\epsilon_2-\epsilon_1))\mu_2 > \delta$ and $\epsilon_2\mu_2 > \delta$: We apply induction:

$$\begin{aligned}
B_2(\mu_2) &\leq C((1-\epsilon(\mu_2))\mu_2)^{\xi(\mu_2)} + C\cdot\rho\mu_2(\epsilon_2(\mu_2)\mu_2)^{\xi(\mu_2)} + 1\\
&\leq C\cdot\mu_2^{\xi(\mu_2)}\left((1-\epsilon(\mu_2))^{\xi(\mu_2)} + \rho\mu_2(\epsilon_2(\mu_2))^{\xi(\mu_2)} + \frac{1}{\mu_2^{\xi(\mu_2)}}\right)\\
&\leq C\cdot\mu_2^{\xi(\mu_2)}\left((1-\epsilon(\mu_2))^{\xi(\mu_2)} + \frac{\rho}{\mu_2} + \frac{1}{\mu_2^{\xi(\mu_2)}}\right) \quad (\text{since } (\epsilon_2(\mu_2))^{\xi(\mu_2)} = \frac{1}{\mu_2^2} \text{ by (10)})\\
&\leq C\cdot\mu_2^{\xi(\mu_2)}\left(e^{-\epsilon(\mu_2)\xi(\mu_2)} + \frac{\rho}{\mu_2} + \frac{1}{\mu_2^{\xi(\mu_2)}}\right)\\
&\leq C\cdot\mu_2^{\xi(\mu_2)}\left(\frac{1}{e\rho} + \frac{\rho}{\mu_2} + \frac{1}{\mu_2^{\xi(\mu_2)}}\right) \quad (\text{since } \epsilon(\mu_2) = \frac{\ln(e\rho)}{\xi(\mu_2)})\\
&\leq C\cdot\mu_2^{\xi(\mu_2)}\left(\frac{1}{4}+\frac{1}{2}+\frac{1}{16}\right) < C\cdot\mu_2^{\xi(\mu_2)}. \quad (\text{since } \xi(\mu_2)\geq 2 \text{ for } \mu_2\geq\delta\geq 2\rho)
\end{aligned}$$

Using the bound $\mu_1(\mathcal{H},\chi) \leq k\cdot\left(\frac{m}{k}\right)^k$, we get the claim.

Putting these Claims 6 and 7 together, the lemma follows. ◀

▶ Remark. Theorem 1 can be extended to the case when the input hypergraph $\mathcal{H}$ is *almost intersecting*, that is, if for all $H\in\mathcal{H}$,

$$H\cap H' = \emptyset \text{ for at most } O(1) \text{ edges } H'\in\mathcal{H}.$$

Acknowledgements. I thank Endre Boros and Vladimir Gurvich for helpful discussions.

References

1. N. Alon, P. Kelsen, S. Mahajan, and R. Hariharan. Approximate hypergraph coloring. *Nord. J. Comput.*, 3(4):425–439, 1996.
2. N. Alon and J.H. Spencer. *The Probabilistic Method*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2004.
3. G. Bacsó, C. Bujtás, Z. Tuza, and V. Voloshin. New challenges in the theory of hypergraph coloring. In S. Arumugam and R. Balakrishnan, editors, *ICDM 2008. International conference on discrete mathematics. Mysore, 2008.*, pages 67–78, Mysore, 2008. Univ. of Mysore.
4. J. Beck and S. Lodha. Efficient proper 2-coloring of almost disjoint hypergraphs. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 598–605, 2002.
5. V. V. S. P. Bhattiprolu, V. Guruswami, and E. Lee. Approximate hypergraph coloring under low-discrepancy and related promises. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, pages 152–174, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.152.
6. J. C. Bioch and T. Ibaraki. Complexity of identification and dualization of positive boolean functions. *Information and Computation*, 123(1):50–63, 1995.
7. E. Boros and K. Makino. A fast and simple parallel algorithm for the monotone duality problem. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 183–194, 2009.

8 A. Chattopadhyay and B. A. Reed. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques: 10th International Workshop, APPROX 2007, and 11th International Workshop, RANDOM 2007, Princeton, NJ, USA, August 20-22, 2007. Proceedings*, chapter Properly 2-Colouring Linear Hypergraphs, pages 395–408. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

9 H. Chen and A. Frieze. *Integer Programming and Combinatorial Optimization: 5th International IPCO Conference Vancouver, British Columbia, Canada, June 3–5, 1996 Proceedings*, chapter Coloring bipartite hypergraphs, pages 345–358. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

10 D. de Werra. Restricted coloring models for timetabling. *Discrete Mathematics*, 165–166:161–170, 1997. Graphs and Combinatorics.

11 I. Dinur and V. Guruswami. PCPs via Low-Degree Long Code and Hardness for Constrained Hypergraph Coloring. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 340–349, 2013.

12 I. Dinur, O. Regev, and C. D. Smyth. The hardness of 3-uniform hypergraph coloring. *Combinatorica*, 25(5):519–535, 2005.

13 C. Domingo. Polynominal time algorithms for some self-duality problems. In *CIAC'97: Proceedings of the 3rd Italian Conference on Algorithms and Complexity, Rome, Italy*, pages 171–180, 1997.

14 M. Dror, G. Finke, S. Gravier, and W. Kubiak. On the complexity of a restricted list-coloring problem. *Discrete Mathematics*, 195(1–3):103–109, 1999.

15 T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995.

16 T. Eiter, G. Gottlob, and K. Makino. New results on monotone dualization and generating hypergraph transversals. In *STOC'02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 14–22, 2002.

17 T. Eiter, G. Gottlob, and K. Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM Journal on Computing*, 32(2):514–537, 2003.

18 T. Eiter, K. Makino, and G. Gottlob. Computational aspects of monotone dualization: A brief survey. KBS Research Report INFSYS RR-1843-06-01, Vienna University of Technology, 2006.

19 K. Elbassioni. On the complexity of monotone dualization and generating minimal hypergraph transversals. *Discrete Applied Mathematics*, 156(11):2109–2123, 2008.

20 P. Erdős, A. L. Rubin, and H. Taylor. Choosability in graphs. In *Proceedings of the West Coast Conference on Combinatorics, Graph Theory and Computing, Arcata, CA, Congr. Numer. XXVI*, pages 125–157, 1979.

21 M. L. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21:618–628, 1996.

22 D. R. Gaur and R. Krishnamurti. Average case self-duality of monotone boolean functions. In *Canadian AI'04: Proceedings of the 17th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence,*, pages 322–338, 2004.

23 G. Gottlob. Hypergraph transversals. In *FoIKS'04: Proceedings of the 3rd International Symposium on Foundations of Information and Knowledge Systems*, pages 1–5, 2004.

24 G. Gottlob and E. Malizia. Achieving new upper bounds for the hypergraph duality problem through logic. In *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), CSL-LICS'14, Vienna, Austria, July 14-18, 2014*, pages 43:1–43:10, 2014.

25 S. Gravier, D. Kobler, and W. Kubiak. Complexity of list coloring problems with a fixed total number of colors. *Discrete Applied Mathematics*, 117(1–3):65–79, 2002.

26 V. Guruswami, P. Harsha, J. Håstad, S. Srinivasan, and G. Varma. Super-polylogarithmic hypergraph coloring hardness via low-degree long codes. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 614–623, 2014.

27 K. Jansen and P. Scheffler. Generalized coloring for tree-like graphs. *Discrete Applied Mathematics*, 75(2):135–155, 1997.

28 D. J. Kavvadias and E. C. Stavropoulos. Monotone boolean dualization is in co-NP$[\log^2 n]$. *Information Processing Letters*, 85(1):1–6, 2003.

29 S. Khot and R. Saket. Hardness of coloring 2-colorable 12-uniform hypergraphs with $2^{(\log n)^{\Omega(1)}}$ colors. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 206–215, 2014.

30 M. Krivelevich, R. Nathaniel, and B. Sudakov. Approximating coloring and maximum independent sets in 3-uniform hypergraphs. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'01, pages 327–328, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

31 L. Lovász. Combinatorial optimization: some problems and trends. DIMACS Technical Report 92-53, Rutgers University, 2000.

32 C. H. Papadimitriou. NP-Completeness: A Retrospective. In *ICALP'97: Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, pages 2–6, London, UK, 1997. Springer-Verlag.

33 M. Pei. *List colouring hypergraphs and extremal results for acyclic graphs.* PhD thesis, University of Waterloo, Canada, 2008.

34 Y. Person and M. Schacht. An expected polynomial time algorithm for coloring 2-colorable 3-graphs. *Electronic Notes in Discrete Mathematics*, 34:465–469, 2009. European Conference on Combinatorics, Graph Theory and Applications (EuroComb 2009).

35 P. D. Seymor. An expected polynomial time algorithm for coloring 2-colorable 3-graphs. *The Quarterly Journal of Mathematics*, 25(1):303–311, 1974.

36 K. Takata. On the sequential method for listing minimal hitting sets. In *DM & DM 2002: Proceedings of Workshop on Discrete Mathematics and Data Mining, 2nd SIAM International Conference on Data Mining*, pages 109–120, 2002.

37 V. G. Vizing. Vertex colorings with given colors. *Metody Diskret. Analiz.*, 29:3–10, 1976.

38 V. I. Voloshin. *Coloring Mixed Hypergraphs: Theory, Algorithms, and Applications.* Fields Institute monographs. American Mathematical Society, 2002.

Turbocharging Treewidth Heuristics*

Serge Gaspers[†1], Joachim Gudmundsson[2], Mitchell Jones[3], Julián Mestre[4], and Stefan Rümmele[5]

1 **UNSW, Sydney, Australia; and**
Data61, CSIRO, Sydney, Australia
sergeg@cse.unsw.edu.au
2 **University of Sydney, Sydney, Australia**
joachim.gudmundsson@sydney.edu.au
3 **University of Sydney, Sydney, Australia**
mjon1572@uni.sydney.edu.au
4 **University of Sydney, Sydney, Australia**
julian.mestre@sydney.edu.au
5 **University of Sydney, Sydney, Australia; and**
UNSW, Sydney, Australia
stefan.rummele@sydney.edu.au

Abstract

A widely used class of algorithms for computing tree decompositions of graphs are heuristics that compute an elimination order, i.e., a permutation of the vertex set. In this paper, we propose to *turbocharge* these heuristics. For a *target treewidth* k, suppose the heuristic has already computed a partial elimination order of width at most k, but extending it by one more vertex exceeds the target width k. At this *moment of regret*, we solve a subproblem which is to recompute the last c positions of the partial elimination order such that it can be extended without exceeding width k. We show that this subproblem is fixed-parameter tractable when parameterized by k and c, but it is para-NP-hard and $W[1]$-hard when parameterized by only k or c, respectively. Our experimental evaluation of the FPT algorithm shows that we can trade a reasonable increase of the running time for quality of the solution.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases tree decomposition, heuristic, fixed-parameter tractability, local search

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.13

1 Introduction

The most widely used treewidth heuristics are simple algorithms that compute an elimination order, i.e., a permutation of the vertex set. For a given elimination order π of a graph G, we obtain a chordal completion G' by eliminating vertices according to this order: when we eliminate a vertex, we add edges to make its neighborhood into a clique, and then remove the vertex. Given this chordal completion G', we can obtain a tree decomposition by traversing π backwards: for each vertex v in π, let $L(v)$ denote the neighbors of v in G' that occur later than v in π; choose a bag that contains $L(v)$ and add a new neighboring bag that contains $\{v\} \cup L(v)$. Thus, for a given elimination order, the width of the tree decomposition we

* The authors acknowledge support under the ARC's Discovery Projects funding scheme (DP150101134).
† Serge Gaspers is the recipient of an Australian Research Council (ARC) Future Fellowship (FT140100048).

obtain is the largest degree of an eliminated vertex (i.e., the degree of the vertex when it is eliminated).[1]

The GreedyDegree heuristic selects a vertex of minimum degree as the next vertex of the elimination order, while the GreedyFillIn heuristic selects a vertex that has fewest non-edges in its neighborhood. These heuristics compare favorably to more involved heuristics; see [3], where a small additional improvement is achieved by turning the resulting chordal completion into a minimal chordal completion in a post-processing step.

In this paper, we propose to *turbocharge* the GreedyDegree and the GreedyFillIn heuristics. For a *target treewidth* k and a *change parameter* c, suppose the heuristic has already computed a partial elimination order π of width at most k and length l, but extending π by one more vertex exceeds the target width k. At this moment of regret, we solve a subproblem which is to compute a partial elimination order π' of width at most k and length $l+1$ which coincides with π in the first $l-c$ positions. Having solved this subproblem, we continue running the heuristic with the partial elimination order π'. In this paper, we formally study this subproblem, parameterized by combinations of l, k, and c, and experimentally evaluate what effect the turbocharging has on the width of the obtained tree decompositions.

The subproblem turns out to be para-NP-hard under Turing reductions for parameter k, $W[1]$-hard for parameters c or l, but fixed-parameter tractable for the combination of parameters k and c. Our implementation is based on this FPT algorithm, and the experiments show an improvement of the width of the obtained tree decompositions for reasonable values of the parameter c, which allows us to trade running time for quality of the solution.

Our turbocharging strategy solves a local-search subproblem when the heuristic gets stuck because all remaining vertices have degree at least $k+1$, similar to the turbocharging strategy for list coloring by Hartung and Niedermeier [7]. Another way to turbocharge the GreedyFillIn heuristic would be to select several vertices at a time and minimize the number of edges added when eliminating this set of vertices. However, we quickly run into limitations of this method, since this problem is $W[1]$-hard when parameterized by the number of vertices we would like to eliminate, and we did not pursue this strategy further.

2 Preliminaries

A graph $\mathcal{G} = (V, E)$ consists of a vertex set V and an edge set E. The neighborhood $N(v)$ of vertex v in G is defined as the set of all vertices adjacent to v, $N(v) = \{w \mid \{v, w\} \in E\}$.

A *tree decomposition* of a graph $\mathcal{G} = (V, E)$ is a pair $\mathcal{T} = (T, \chi)$, where T is a tree and χ maps each node t of T (we use $t \in T$ as a shorthand below) to a *bag* $\chi(t) \subseteq V$, such that

1. for each $v \in V$, there is a $t \in T$, s.t. $v \in \chi(t)$;
2. for each $\{v, w\} \in E$, there is a $t \in T$, s.t. $\{v, w\} \subseteq \chi(t)$;
3. for each $r, s, t \in T$, s.t. s lies on the path from r to t, $\chi(r) \cap \chi(t) \subseteq \chi(s)$.

The *width* of a tree decomposition is defined as the cardinality of its largest bag minus one. The *treewidth* of a graph $\mathcal{G}$, denoted by $tw(\mathcal{G})$, is the minimum width over all tree decompositions of $\mathcal{G}$. For a given graph $\mathcal{G}$ and integer k, deciding whether $\mathcal{G}$ has treewidth at most k is NP-complete [1]. For fixed k, one can decide in linear time whether a graph has treewidth $\leq k$ and, if so, compute a tree decomposition of width k [2].

[1] We refer the reader who is not familiar with some of these notions to the Preliminaries section.

Treewidth	
Instance:	Graph G and integer k.
Problem:	Decide whether $tw(\mathcal{G}) \leq k$ holds.

One of the alternative characterizations of treewidth is based on so called *elimination orders*. Let $\mathcal{G} = (V, E)$ be a graph and $v \in V$ a vertex of $\mathcal{G}$. Eliminating v from $\mathcal{G}$ refers to the process of forming a clique out of the neighbourhood of v and removing v and its incident edges, that is, we create a new graph $\mathcal{G}' = (V \setminus \{v\}, (E \cup E_1) \setminus E_2)$, where $E_1 = \{\{u, w\} \mid u, w \in N(v)\}$ and $E_2 = \{e \in E \mid v \in e\}$. An elimination order of a graph $\mathcal{G} = (V, E)$ with $|V| = n$ is a bijective function $\pi : V \to \{1, \ldots, n\}$. Starting with $\mathcal{G}$ and iteratively eliminating the vertices in V according to the order π results in a sequence of $n + 1$ graphs with the last one being the empty graph. The width of π is the maximum degree of any vertex $v \in V$ during its elimination according to the order π. This notion leads to the following alternative characterization of treewidth.

▶ **Theorem 1** (see for example [3]). *Let $\mathcal{G} = (V, E)$ be a graph and let $k \in \mathbb{N}$. $\mathcal{G}$ has treewidth at most k if and only if there exists an elimination order π of width at most k.*

We generalize the notion of elimination orders to partial elimination orders as follows. A partial elimination order of length $l \leq n$ of a graph $\mathcal{G} = (V, E)$ with $|V| = n$ is a bijective partial function $\pi : V \to \{1, \ldots, l\}$, that is, an enumeration of l vertices of the graph. The intended meaning of a partial elimination order is that it represents the first l positions of an elimination order. Analogously to elimination orders, the width of partial elimination order π is the maximum degree of any vertex $v \in V$ during its elimination according the order π.

The two heuristics which we mentioned earlier as well as our algorithm compute the treewidth based on elimination orders. The GreedyDegree heuristic as well as the GreedyFillIn heuristic construct an elimination order by iteratively selecting the next vertex in the elimination order and eliminating it from the graph. During the elimination step, the vertex is removed and all its neighbours are connected to a clique. The selection of the next vertex is based on a greedy criteria. GreedyDegree selects the vertex with the minimal degree and GreedyFillIn selects the vertex whose elimination results in the fewest new edges that need to be added to the graph in order to form a clique out of its neighbours. In both cases ties are broken arbitrarily.

3 Local Search Variants of the Treewidth Problem

We are interested in how the existing greedy heuristics for Treewidth can be improved via local search. We introduce the following problem, which we call the *incremental conservative (IC) treewidth problem*, since it follows the spirit of incremental conservative k-list coloring of graphs by Hartung and Niedermeier [7].

IC-Treewidth	
Instance:	Graph $\mathcal{G}$, integer k and c, and partial elimination order π of length l and width $\leq k$.
Problem:	Does there exist a partial elimination order π' of length $l+1$ and width $\leq k$ such that π and π' are identical on the first $l - c$ positions?

Since we want to use an algorithm for IC-Treewidth as a subroutine for improving an existing partial elimination order, we are interested in finding parameters for which the problem becomes fixed-parameter tractable. The following result shows, that the length l and hence also the change c are ineligible.

▶ **Theorem 2.** IC-Treewidth *is* W[1]*-hard when parameterized by* l*.*

Proof. By reduction from Independent Set. An instance of Independent Set is given by a graph $\mathcal{G} = (V, E)$ and integer k. The question is whether $\mathcal{G}$ has an independent set of size k. Independent Set is W[1]-complete when parameterized by k [4].

Let $(\mathcal{G} = (V, E), k)$ be an instance of Independent Set with $V = \{v_1, \ldots, v_n\}$. We denote the maximum degree of $\mathcal{G}$ by $\Delta(\mathcal{G}) = d$. Now construct an instance $(\mathcal{G}', \pi, c)$ of IC-Treewidth as follows. Let $U, W, X_1, \ldots, X_{k-1}, Y_1, \ldots, Y_n$ be sets of new vertices of cardinalities $|U| = n$, $|W| = n + 2d$, $|X_i| = 2d + 1$ for $1 \leq i \leq k - 1$, and $|Y_j| = d + 1$ for $1 \leq j \leq n$. Let K_X^i with $1 \leq i \leq k - 1$ denote the complete graph of the vertices on X_i. Analogously, we denote by K_Y^j with $1 \leq j \leq n$ and by K_W the complete graphs over Y_j and W, respectively. The new graph $\mathcal{G}' = (V \cup U \cup W \cup X_1 \cdots \cup X_{k-1} \cup Y_1 \cdots \cup Y_n, E')$ contains all edges of $\mathcal{G}$, and all edges of the complete graphs K_X^i, K_Y^j, and K_W. Additionally, we add the following edges. Let $U = \{u_1, \ldots, u_n\}$.

$$
\begin{aligned}
\{u_i, v_j\} &\qquad 1 \leq i, j \leq n \\
\{u_i, x\} &\qquad 1 \leq i \leq k-1 \text{ and } x \in X_i \\
\{u_i, w\} &\qquad k \leq i \leq n \text{ and } w \in W \\
\{v_j, y\} &\qquad 1 \leq j \leq n \text{ and } y \in Y_j \\
\{x, w\} &\qquad x \in X_i, 1 \leq i \leq k-1 \text{ and } w \in W \\
\{y, w\} &\qquad y \in Y_j, 1 \leq j \leq n \text{ and } w \in W
\end{aligned}
$$

To complete the construction of the instance for IC-Treewidth, we set $\pi = (u_1, \ldots, u_{k-1})$ and $c = k - 1$. The partial elimination order π has width $n + 2d + 1$ since each vertex u_i, $1 \leq i \leq k - 1$, has as neighbourhood the n vertices of V and $2d + 1$ vertices of X_i at the time of its elimination. Note that this instance can be constructed in polynomial time.

We will show that $\mathcal{G}$ has an independent set of size k if and only if there exists a partial elimination order π' of width $n + 2d + 1$ and length k for graph $\mathcal{G}'$. For the first direction, assume that $\mathcal{G}$ has an independent set S of size k. Let π' be an arbitrary order of the k vertices in S. We show that π' is a partial elimination order of width $n + 2d + 1$ for graph $\mathcal{G}'$. The neighbourhood of each vertex $v_j \in S$ in $\mathcal{G}'$ consists of its original neighbourhood in $\mathcal{G}$ together with all the vertices of U and Y_j. Hence its degree is bounded by $d(v_j) \leq n + 2d + 1$. Since S is an independent set, it holds for all pairs $v_i, v_j \in S$, that eliminating v_i from $\mathcal{G}'$ does not change the neighbourhood of v_j. Therefore, π' is a partial elimination order of length k and width $n + 2d + 1$.

For the second direction, assume that π' is a partial elimination order of length k and width $n + 2d + 1$ for $\mathcal{G}'$. We will show that the k vertices of π' form an independent set in $\mathcal{G}$. π' can not contain any vertex from W since they have degree $n(3 + d) + 2dk - 1$. Similar, π' can not contain any vertex from X_i, $1 \leq i \leq k - 1$, or from Y_j, $1 \leq j \leq n$, since they have degree $n + 4d + 1$ and $n + 3d + 1$, respectively. We can also exclude vertices $u_k, \ldots, u_n$ since they have degree $2n + 2d$. Starting by eliminating a vertex $u_i \in \{u_1, \ldots, u_{k-1}\}$ creates a clique of all vertices in $V \cup X_i$. This means, if π' starts with u_i, then it can not contain any vertex from V. Note that eliminating a vertex $v_j \in V$ forms a clique of all vertices in $U \cup Y_j$. Hence, if π' contains v_j, it can not be succeeded by any vertex in U. These two observations combined, say that π' either consists only of vertices of U or only of vertices of V. We can exclude the case U, since π' has length k and there are only $k - 1$ suitable vertices in U. Therefore, π' contains only vertices from V. It remains to show that these vertices form an independent set. Assume towards a contradiction, that π' contains two adjacent vertices, say v_i and v_j. W.l.o.g. we assume v_i is eliminated before v_j. Eliminating v_i introduces an edge

between v_j and all $d+1$ vertices of Y_i. Hence, v_j has now degree $d(v_j) \geq n+2d+2$. But this means v_j can not be contained in π', which contradicts our assumption and the vertices in π' form indeed an independent set. ◀

This reduction from Independent Set together with a straight-forward NP-membership via a guess and check algorithm, gives us NP-completeness of IC-Treewidth.

▶ **Corollary 3.** IC-Treewidth *is NP-complete.*

A problem that is closely related to IC-Treewidth is the following Length-l-Partial-Elimination-Order problem. To solve IC-Treewidth, we can eliminate the first $l-c$ vertices of the graph and then ask for a length $c+1$ partial elimination order.

Length-l-Partial-Elimination-Order	
Instance:	Graph $\mathcal{G}$, integer l and k.
Problem:	Does there exist a partial elimination order of $\mathcal{G}$ of length l and width $\leq k$?

▶ **Theorem 4.** Length-l-Partial-Elimination-Order *is fixed-parameter tractable when parameterized by l and k.*

Proof. Let $\mathcal{G}=(V,E)$, l and k be an instance of Length-l-Partial-Elimination-Order. Let S be set of vertices of degree at most k, i.e., $S=\{v \in V \mid d_{\mathcal{G}}(v) \leq k\}$. Let $\mathcal{G}[S]$ be the subgraph $\mathcal{G}$ induced by S. Let $\mathcal{A}$ be a greedy algorithm for Independent Set that iteratively selects a minimum degree vertex and remove its closed neighborhood from the graph, until it either finds an independent set I of size l or fails to do so.

In case $\mathcal{A}$ succeeds, we show that sequencing $(v_1, v_2, \ldots, v_l)$ of I is a partial elimination order of $\mathcal{G}$ of width $\leq k$. Note that each v_i belongs to S, so it has degree at most k in $\mathcal{G}$. Since I is an independent set, eliminating v_i does not add a new edge incident on I. Hence, the partial elimination order $(v_1, \ldots, v_l)$ has width at most k.

On the other hand, if $\mathcal{A}$ fails to find an independent set of size l, we know that $|S| \leq (l-1)(k+1)$. To see this, note that adding some vertex $v \in S$ to I can block at most k other vertices (v's neighbors) from being selected by the greedy algorithm. Since the maximum independent set that $\mathcal{A}$ can find in case of failure has size $l-1$, the bound on S follows.

We can exploit this insight to design a branching algorithm for Length-l-Partial-Elimination-Order. Each node of the branching process will have associated a partial elimination order π' and a graph $\mathcal{G}'$. On the first level we only have the root node, where π' is empty and $\mathcal{G}'$ is the input graph. Consider a node $(\pi', \mathcal{G}')$ at level i of the branching process and let $S'=\{v \in V \mid d_{\mathcal{G}'}(v) \leq k\}$. If $|S'| > (l-i)(k+1)$ then we can use $\mathcal{A}$ to extend the partial elimination order by $(l-i+1)$ additional nodes, we can do just that and stop the branching process. Otherwise, if $|S'| \leq (l-i)(k+1)$, we branch on every node $v \in S'$, by adding it to π' order and eliminating it from $\mathcal{G}'$, thus generating a new node $(\pi'', \mathcal{G}'')$ on level $i+1$.

Notice that the number of branches we need to follow from a node in level i is at most $(l-i)(k+1)$. Therefore, the total number of nodes we explore is at most $\prod_{i=1}^{l}(l-i)(k+1) = (l-1)!(k+1)^l$. Hence, we can decide Length-l-Partial-Elimination-Order in $\mathcal{O}^*\left((l-1)!(k+1)^l\right)$ time. ◀

Given a partial elimination order, we can backtrack the last c choices and use this FPT result to extend it again by $c+1$ vertices. This leads to the following corollary.

▶ **Corollary 5.** IC-Treewidth *is fixed-parameter tractable when parameterized by c and k.*

Similar, the W[1]-hardness of IC-Treewidth when parameterized by c carries over to Length-l-Partial-Elimination-Order parameterized by l. We show next, that the combination of parameters l and k is indeed necessary, that is, we show hardness for parameter k alone.

▶ **Theorem 6.** Length-l-Partial-Elimination-Order *is NP-hard even when* $k = 5$.

Proof. Our reduction is from the NP-hard problem Independent Set on Cubic Graphs, which takes as input a 3-regular graph $\mathcal{G} = (V, E)$ and an integer k, and the question is whether $\mathcal{G}$ has an independent set of size k, i.e., a set of k vertices that are pairwise non-adjacent [5]. We construct an instance $(\mathcal{G}', l = k, 5)$ for Length-l-Partial-Elimination-Order as follows. To obtain $\mathcal{G}'$, we start from $\mathcal{G}$ and add a disjoint clique W on 7 vertices. For every vertex v of $\mathcal{G}$, we add two vertices a_v and b_v to $\mathcal{G}'$ and make them adjacent to $W \cup \{v\}$. To see that a partial elimination order π of width at most 5 of $\mathcal{G}'$ corresponds to an independent set in $\mathcal{G}$, and vice-versa, first observe that π contains no vertex from W or $N(W)$; indeed, the first vertex from $W \cup N(W)$ occurring in π has more than 5 neighbors when it is eliminated. Secondly, assume the partial elimination order contains two adjacent vertices. Let v be the first vertex that is eliminated for which at least one neighbor u has already been eliminated. But then, v has degree at least 6 when it is eliminated because eliminating u added the edges $\{v, a_u\}$ and $\{v, b_u\}$. But, on the other hand, eliminating an independent set of size $l = k$ gives a partial elimination order of width 5 and length l. ◀

IC-Treewidth is defined as a decision problem. We call the problem of actually computing such a partial elimination order π', the function version of IC-Treewidth. As mentioned before, if it exists, computing a tree decomposition of width k can be done in linear time for fixed k [2]. This does not hold for our partial elimination orders.

▶ **Theorem 7.** *The function version* IC-Treewidth *is NP-hard under Turing reductions even when* $k = 5$.

Proof. By reduction from Length-l-Partial-Elimination-Order for the special case of $k = 5$. We can solve this problem by iteratively solving IC-Treewidth starting with an empty partial elimination order and ending with one of length $l - 1$. ◀

Another application of Theorem 4 is a greedy algorithm where iteratively the next l vertices are selected instead of a single next vertex. A natural question is, whether we can get an FPT result if we try to select these l vertices in such a way, that number of fill-in edges is minimal. The following result shows that this is unlikely.

Min-FillIn-Set	
Instance:	Graph $\mathcal{G} = (V, E)$, integer l and T.
Problem:	Does there exist a set $S \subseteq V$ of size l, such that eliminating the vertices in S from $\mathcal{G}$ adds at most T new edges to $\mathcal{G}$?

▶ **Theorem 8.** Min-FillIn-Set *is W[1]-hard when parameterized by* l.

Proof. By reduction from Clique. An instance of Clique is given by a bipartite graph $\mathcal{G}$ and integer k. The question is, whether $\mathcal{G}$ contains a clique of size k. Clique is W[1]-hard when parameterized by k.

Let $\mathcal{G} = (V, E)$ and k be an instance of Clique. We construct an instance $(\mathcal{G}' = (V', E'), l, T)$ of Min-FillIn-Set as follows. The vertices V' consist of three disjoint sets $V' = X \cup Y \cup Z$, defined as follows. The set X contains a vertex for each edge in the original

graph, i.e., $X = \{x_e \mid e \in E\}$. The set Y contains two copies of each vertex inthe original graph, i.e., $Y = \{y_v, y'_v \mid v \in V\}$. The set Z contains $4k$ new vertices $Z = \{z_1, \ldots, z_{4k}\}$. For each edge $e = \{v, w\} \in E$ we add the following 4 edges to E': $\{x_e, y_v\}$, $\{x_e, y'_v\}$, $\{x_e, y_w\}$, and $\{x_e, y'_w\}$. Additionally, E' contains all possible edges between vertex sets Y and Z, i.e., Y and Z form a complete bipartite subgraph. Finally, we set $l = \binom{k}{2}$ and $T = 4\binom{k}{2} + k$. Clearly, $(\mathcal{G}' = (V', E'), l, T)$ can be constructed in polynomial time.

For the correctness, assume first that $C \subseteq V$ is a solution to the Clique problem, i.e., C is a clique of size k. We will show that the $\binom{k}{2}$ edges between vertices of C witness a solution for our Min-FillIn-Set instance. Let $S \subseteq X$ be the $\binom{k}{2}$ vertices corresponding to these edges. By construction, the neighbourhood of S consists of $2k$ vertices in Y that correspond to the k vertices forming the clique C and their copies, say $Y_C = \{y_{i_1}, \ldots y_{i_k}\}$ and $Y'_C = \{y'_{i_1}, \ldots y'_{i_k}\}$. Eliminating the vertices of S from $\mathcal{G}'$ adds a new edge between every vertex in $y \in Y_C$ and its copy $y' \in Y'_C$, resulting in k new edges. Furthermore, the elimination of a vertex $x_{\{v,w\}} \in S$ forces the following 4 edges to be added to $\mathcal{G}'$: $\{y_v, y_w\}$, $\{y_v, y'_w\}$, $\{y'_v, y_w\}$, and $\{y'_v, y'_w\}$. This results in a total of $T = 4\binom{k}{2} + k$ new edges being added to $\mathcal{G}'$. Hence, S is a solution for the Min-FillIn-Set instance.

For the other direction, assume that $S \subseteq V'$ is a solution for Min-FillIn-Set. Observe that eliminating a vertex $y \in Y$ forces us to create a clique out of the $4k$ vertices Z, resulting in more than T new edges to be added to $\mathcal{G}'$. Similar, eliminating a vertex $z \in Z$ forces us to create a clique out of the vertices in Y. Note that we can assume that Y contains at least $4k$ vertices, since otherwise we could simply blow up the Clique instance by adding at most k new isolated vertices. Hence, S contains only vertices of X. As mentioned above, the elimination of a vertex $x_{\{v,w\}} \in X$ forces the following 4 edges to be added to $\mathcal{G}'$: $\{y_v, y_w\}$, $\{y_v, y'_w\}$, $\{y'_v, y_w\}$, and $\{y'_v, y'_w\}$. By construction, these 4 edges are unique for every vertex $x \in X$. Since S is a solution of size $\binom{k}{2}$, we know that these $4\binom{k}{2}$ new edges are added to $\mathcal{G}'$. Furthermore, for every vertex $v \in V$ that is incident to an edge e corresponding to one of the eliminated vertices $x_e \in S$, the edge $\{y_v, y'_v\}$ is added to $\mathcal{G}'$. Since S is a solution, we know that $\leq T = 4\binom{k}{2} + k$ are added to $\mathcal{G}'$. Hence, S corresponds to a set of $\binom{k}{2}$ edges that is incident to at most k vertices. But this is only true, if S corresponds to the set of edges of a k-clique in $\mathcal{G}$. ◀

4 Turbocharged Treewidth Heuristics

In the previous section we showed that IC-Treewidth is fixed-parameter tractable when parameterized by c and k. We use this FPT algorithm to extend an existing partial elimination order in case a greedy heuristic gets "stuck":

We use a standard greedy algorithm, like GreedyDegree or GreedyFillIn, with one modification. In each step of the heuristic, we check if the next vertex that is to be eliminated, will cause the partial elimination order to exceed our given target width. If this is not the case, we proceed with the heuristic. On the other hand, if we would exceed the target width (we call this a point of regret), instead we backtrack the last c eliminated vertices and use our FPT algorithm to extend this shortened partial elimination order by adding $c+1$ vertices. If the FPT algorithm is not able to produce such an extension, we abort, otherwise we switch again to the greedy heuristic and continue to the next point of regret.

Algorithm 1 explains this approach in more detail for the case of using the GreedyDegree heuristic. To change the used heuristic, only line 4 needs to be altered. The outer loop (line 3) is executed $|V|$ many times, as each iteration either extends the partial elimination order π one position or aborts the whole search. Lines 5–7 correspond to the case when the

Algorithm 1: TurbochargedMinDegree

Input : Graph $\mathcal{G} = (V, E)$, integer k, integer c.
Output : Elimination order of width $\leq k$ or *no* if none was found.

```
1  H ← G;
2  π ← ();
3  for i ← 1 to |V| do
4      choose vertex v with minimum degree;
5      if d(v) ≤ k then
6          π ← π + (v);
7          H ← eliminate(H, v);
8      else
9          G' ← eliminate(G, π[1], ..., π[i − c − 1]);
10         W ← {v ∈ V(G') | d(v) ≤ k};                    // W is bounded by c(k + 1)
11         (H, π') ← IC-Treewidth(G', W, k, c + 1);
12         if π' is empty then
13             return no;
14         else
15             π ← (π[1], ..., π[i − c − 1]) + π';
16 return π
```

Algorithm 2: IC-Treewidth

Input : Graph $\mathcal{G}$, vertex set W, integer k, integer c.
Output : Pair $(\mathcal{H}, \pi)$ where π is a partial elimination order of width k and length c and $\mathcal{H}$ is the remaining graph. $(null, \emptyset)$ in case of failure.

```
1  for v ∈ W do
2      H ← eliminate(G, v);
3      if c = 1 then
4          return (H, (v));
5      W' ← {v ∈ V(H) | d(v) ≤ k};
6      (H, π') ← IC-Treewidth(H, W', k, c − 1);
7      if π' is not empty then
8          return (H, (v) + π');
9  return (null, ∅);
```

heuristic does not run into a point of regret. Here we add the selected vertex v to π and eliminate it from the graph. In case there is a point of regret, we fix the first part of the elimination order (except the last c positions) and eliminate these vertices from the graph (line 9). Vertex set W at line 10 contains all vertices of degree $\leq k$. These are the vertices which can be eliminated next without exceeding the target treewidth. The FPT algorithm from Theorem 4 is implemented as a recursive procedure outlined in Algorithm 2.

5 Experimental Evaluation

To complement our theoretical analysis of the turbocharged approach, we performed a thorough experimental evaluation of the turbocharged versions of GreedyDegree and GreedyFillIn. The experiments were run on a quad-core Intel Core i7 processor running at 2.7 GHz with 16GB of RAM. The implementation was in Java 7. We implemented and

Table 1 Comparison of average quality and average running time on different classes of randomly generated partial k-trees.

			`min-degree`		`min-fill-in`		`turbo-min-degree`		`turbo-min-fill-in`	
n	k	p	quality	time	quality	time	quality	time	quality	time
250	10	0.20	10.44	**0.12**	11.42	0.18	10.44	0.22	**10.12**	0.43
250	10	0.40	10.16	**0.10**	11.34	0.15	10.16	0.20	**10.04**	0.36
250	15	0.20	15.60	**0.17**	16.64	0.27	15.60	0.28	**15.34**	0.63
250	15	0.40	15.20	**0.14**	16.38	0.22	15.20	0.26	**15.12**	0.51
250	20	0.20	20.64	**0.22**	21.96	0.37	20.64	0.35	**20.32**	0.86
250	20	0.40	20.22	**0.18**	21.60	0.30	20.22	0.34	**20.08**	0.69
500	10	0.20	10.72	**0.36**	11.72	0.59	10.72	0.51	**10.24**	1.55
500	10	0.40	10.32	**0.28**	11.64	0.44	10.32	0.49	**10.26**	1.23
500	15	0.20	15.94	**0.63**	16.86	1.09	15.94	0.83	**15.70**	2.71
500	15	0.40	15.32	**0.46**	17.04	0.78	15.32	0.79	**15.20**	1.96
500	20	0.20	20.88	**0.94**	22.18	1.67	20.88	1.21	**20.82**	4.04
500	20	0.40	**20.32**	**0.67**	22.08	1.17	**20.32**	1.16	20.38	2.84
1000	10	0.20	10.90	**1.75**	11.94	3.11	10.90	2.08	**10.64**	7.81
1000	10	0.40	10.56	**1.29**	11.98	2.18	10.56	1.93	**10.20**	5.83
1000	15	0.20	16.04	**3.46**	17.20	6.71	16.04	3.87	**15.94**	15.46
1000	15	0.40	15.58	**2.44**	17.26	4.40	15.58	3.70	**15.46**	10.78
1000	20	0.20	**21.16**	**5.34**	22.38	10.24	**21.16**	5.58	21.54	22.38
1000	20	0.40	20.50	**3.76**	22.56	6.90	20.50	5.77	**20.34**	15.84

tested the following four algorithms:

- `min-degree`: Iteratively eliminates a vertex with minimum degree.
- `min-fill-in`: Iteratively eliminates a vertex with minimum fill-in.
- `turbo-min-degree`: The turbocharged version of `min-degre`.
- `turbo-min-fill-in`: The turbocharged version of `min-fill-in`.

When generating the elimination order for each of the above algorithms ties between vertices need to be broken. To handle this we use a fixed seed to generate a random permutation on the vertices. This permutation is then used to break ties. Using the same seed across all algorithms allows for a fair comparison between the heuristic and its turbocharged version.

The turbocharged version of the `min-degree` heuristic was implemented using the pseudo-code given in Algorithms 1 and 2, with one minor enhancement. In the first line of Algorithm 2, no specific order is given on W. To better guide the search, we first sort the vertices in W by increasing degree. The idea is that while we are now sorting the set W according to the heuristic at every call, we hope to find an extended ordering quicker. The `min-fill-in` heuristic is implemented using the corresponding versions of Algorithms 1 and 2, with the same enhancement.

We tested our algorithms on two types of instances: randomly generated partial k-trees (Section 5.1), and benchmark instances (Section 5.2).

In the rest of this section we explain how these instances were generated/sourced and analyze the experimental performance of the different algorithms.

5.1 Random instances

The partial k-trees were generated using the method by Gogate and Dechter [6, Section 7.2]. The generator takes as input a triple of parameters (n, k, p). It generates a graph of treewidth at most k having n nodes and $(1-p)\left(kn - \binom{k+1}{2}\right)$ edges. In order to ensure that the graph has a tree decomposition of width *exactly* k, we apply the Maximum-Minimum Degree (MMD) lower bound proposed by Koster et al. [8] and only keep those that are guaranteed to have treewidth k. Fifty partial k-trees were generated for each triple (n, k, p),

for all combinations of the following parameters $n = \{250, 500, 1000\}$, $k = \{10, 15, 20\}$ and $p = \{0.2, 0.4\}$.

From Theorem 4 we know that the number of times the turbocharged heuristic has to backtrack might be exponential in the length of the partial elimination order (l). Therefore, to keep the computation tractable, c needs to be small. For the experiments we choose $c = 8$ as the default value.

Table 1 provides statistical summaries of the quality and running times of the different algorithms on the randomly generated instances. The running times of `turbo-min-degree` and `turbo-min-fill-in` contain the running times of `min-degree` and `min-fill-in`, respectively, since whenever the turbocharged version fails to find a decomposition of given target width, we return the result of the standard version instead. Hence, the quality of the turbocharged version will always be at least as good as the quality of the standard algorithm and the running time will always be slower. Our first observation is that `min-degree` outperforms `min-fill-in` in terms of time and quality, which is consistent with the results reported by Bodlaender and Koster [3].

For `turbo-min-degree`, we saw no improvement in the quality of the decomposition. This is probably because in most cases `min-degree` finds the optimal solution (47% of the instances) or a solution very close to the optimal. Even after setting $c = 12$ the turbocharged version failed to improve on any instances.

For `turbo-min-fill-in`, however, we observed a large improvement in quality. In this case the algorithm was able to find the optimal treewidth in 690 out of the 900 instances. In many of the smaller instances, the algorithm did not even backtrack the full $c = 8$ vertices; indeed, on average only six steps was required. This means that for `min-fill-in` we spend a few additional seconds to turbocharge the heuristic and get a considerable improvement. Note that for most of the random instances `turbo-min-fill-in` finds a treewidth that is better than the ones found by `min-degree` and `turbo-min-degree`.

5.2 Benchmark instances

Two data sets were used for the experiments: DIMACS Graph coloring networks instances,[2] and Bayesian networks repository instances.[3] In total, there are 73 instances out of which 63 are DIMACS Graph coloring networks instances and 10 are Bayesian networks repository instances. The purpose of these experiments is to test the turbocharged heuristics performance on larger instances.

Each heuristic, `min-degree` and `min-fill-in`, was executed three times on each instance. The best result (smallest treewidth) for each heuristic was selected. Finally, the heuristic producing the best result for each instance was turbocharged, using the same random seed for consistency.

For the turbocharged version the heuristic requires a target treewidth parameter (k), which is unknown. To get around this problem we chose to perform a biased binary search as follows. Let k' be the best treewidth found by either `min-degree` or `min-fill-in`. The experimental evaluation showed that the turbocharge heuristic typically improved the treewidth by 3-5%. As a result we chose to perform a binary search in the range $[0.94 \cdot k', k' - 1]$ which terminated after four iterations. In the case when this interval is non-existent (i.e. $(k' - 1)/k' \leq 0.94$), we run the turbocharged heuristic with $k' - 1$, $k' - 2$, and so on.

[2] `http://mat.gsia.cmu.edu/COLOR/instances.html`
[3] `http://www.cs.huji.ac.il/site/labs/compbio/Repository/`

Table 2 A subset of the experimental results on DIMACS Graph coloring networks. For instances DSJC1000.5 and DSJC500.9 we used $c = 6$, and for the other instances $c = 8$.

				`min-degree`		`min-fill-in`		`turbo`	
id	n	m	tw	quality	time	quality	time	quality	time
queen7_7	49	952	35	37	**0.056**	37	0.075	**36**	0.104
queen8_8	64	1456	46	50	**0.081**	48	0.099	**47**	0.543
queen9_9	81	2112	59	64	**0.100**	63	0.128	**62**	0.266
queen11_11	121	3960	89	97	**0.231**	95	0.283	**93**	12.49
queen13_13	169	6656	125	140	**0.610**	137	0.808	**135**	36.67
queen14_14	196	8372	143	164	**1.060**	160	1.372	**159**	95.08
myciel4	23	71	10	11	**0.011**	11	0.016	**10**	4.62
le450_5b	450	5734	309	316	**15.12**	318	19.42	**311**	500.3
le450_15c	450	16680	372	376	**21.35**	376	26.44	**372**	240.6
le450_25d	450	17425	349	367	**20.48**	363	25.18	**360**	584.4
DSJC1000.5	1000	499652	977	980	**642**	978	705	**977**	5429
DSJC125.1	125	1472	64	67	**0.144**	66	0.170	**65**	54.885
DSJC250.1	250	6436	176	180	**1.835**	177	2.300	**176**	264.46
DSJC500.1	500	24916	409	413	**31.086**	411	43.048	**410**	2089.77
DSJC500.5	500	125248	479	481	**41.024**	482	48.481	**479**	19467.95
DSJC500.9	500	224874	492	493	**45**	493	47	**492**	2662

Coloring

We ran our heuristics on 63 instances of the DIMACS Graph coloring networks. Some of the results are shown in Table 2. The fourth column shows the best known treewidth for each instance extracted from the papers by Koster et al. [8] and Gogate and Dechter [6]. Each row also lists the results obtained by the `min-degree`, the `min-fill-in` and the turbocharged version (`turbo`).

In the 31 cases where neither greedy heuristics found the best known solution, the turbocharge method was able to improve the result in 16 of the instances. These instances are listed in Table 2. Specifically, in six of the cases the turbocharged version was able to find a tree decomposition that has width equal to the best known solution.

In all but two cases the computation terminated within two hours. Note that due to the size of some of the large instances, the parameter $c = 6$ was used for four instances, $c = 4$ for one instance and, $c = 8$ for the remaining instances.

In Table 3 we list the same instances as in Table 2 to compare our results with the results reported by Koster et al. [8] and Gogate and Dechter [6]. However it should be noted that Koster et al. [8] implemented several approaches with varying quality performance and speed, but we only include the smallest treewidth result in the table. For more details see [8].

Bayesian Networks

The Bayesian network instances are directed graphs transformed into undirected graphs for the experiments. The set contains ten instances, most of these are quite small so in nine out of the ten instances either the `min-degree` or `min-fill` heuristic found the best known solution. Therefore turbocharging yielded no benefit. The only exception out of the ten instances was the Link instance, where the turbocharged algorithm was able to improve the min-fill heuristic from 15 to 13, which also improved the best known bound for this instance.

6 Conclusion and Future Work

We studied variants of the Treewidth problem that aim at modelling local search scenarios that arise in the context of tree decomposition heuristics. We have shown that IC-Treewidth,

Table 3 A comparison between turbocharged heuristics and the results reported by Gogate and Dechter [6] and Koster et al. [8]. Note that the algorithm by Gogate and Dechter [6] was terminated after 3 hours. Also note that Koster et al. [8] implemented several approaches with varying quality performance and speed, however, only the smallest treewidth result is listed in this table.

			Gogate and Dechter [6]		Koster et al. [8]		`turbo`	
id	n	m	quality	time	quality	time	quality	time
queen7_7	49	952	**35**	543	**35**	0.51	36	**0.10**
queen8_8	64	1456	**46**	10800	**46**	1.49	47	**0.54**
queen9_9	81	2112	**59**	10800	**59**	3.91	62	**0.27**
queen11_11	121	3960	**89**	10800	**89**	23.36	93	**12.5**
queen13_13	169	6656	**125**	10800	**125**	107.6	135	**36.7**
queen14_14	196	8372	**143**	10800	145	215.4	159	**95.1**
myciel4	23	71	**10**	10800	**10**	**0.01**	**10**	4.6
le450_5b	450	5734	**309**	10800	313	7909	311	**500**
le450_15c	450	16680	**372**	10800	376	12471	**372**	**241**
le450_25d	450	17425	**349**	10800	356	11376	360	**584**
DSJC1000.5	1000	499652	**977**	10800	*	*	**977**	**5429**
DSJC125.1	125	1472	**64**	10800	67	171.5	65	**54.9**
DSJC250.1	250	6436	**176**	10800	179	5507	**176**	**264**
DSJC500.1	500	24916	**409**	10800	*	*	410	**2089**
DSJC500.5	500	125248	**479**	**10800**	*	*	**479**	19468
DSJC500.9	500	224874	**492**	10800	*	*	**492**	**2662**

the problem of extending a given partial elimination order without increasing its width by recomputing at most the last c eliminated vertices, is hard when parameterized by either the length of the partial elimination order or its width. But the problem becomes fixed-parameter tractable when parameterized by the width and c combined. We used this FPT result to turbocharge existing greedy heuristics by performing this local search whenever the heuristic would exceed some given target width. This approach was implemented and evaluated, showing that we can improve the quality of the heuristics with a modest trade off in the running time.

In future work it would be interesting to study a permissive variant of IC-Treewidth, which, for a given graph $G = (V, E)$, integer k, and partial elimination order π of length l and width at most k, asks to either compute a partial elimination order of length $l + 1$ and width at most k, or to determine that G has no elimination order (of length $|V|$) that coincides with π on the first $l - c$ vertices.

Acknowledgments. We thank Michael R. Fellows for inspiring this line of research.

References

1 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

3 Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations I. Upper bounds. *Inf. Comput.*, 208(3):259–275, 2010. doi:10.1016/j.ic.2009.03.008.

4 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: on completeness for W[1]. *Theor. Comput. Sci.*, 141(1&2):109–131, 1995.

5 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

6 Vibhav Gogate and Rina Dechter. A complete anytime algorithm for treewidth. In *Proc. of UAI'04*, pages 201–208. AUAI Press, 2004.

7 Sepp Hartung and Rolf Niedermeier. Incremental list coloring of graphs, parameterized by conservation. *Theor. Comput. Sci.*, 494:86–98, 2013.
8 Arie M. C. A. Koster, Hans L. Bodlaender, and Stan P. M. van Hoesel. Treewidth: Computational experiments. *Electronic Notes in Discrete Mathematics*, 8:54–57, 2001.

On Satisfiability Problems with a Linear Structure*

Serge Gaspers†[1], Christos H. Papadimitriou[2], Sigve Hortemo Sæther[3], and Jan Arne Telle[4]

1 **UNSW, Sydney, Australia; and Data61, CSIRO, Sydney, Australia**
sergeg@cse.unsw.edu.au
2 **UC Berkeley, USA**
christos@berkeley.edu
3 **University of Bergen, Norway**
sigve.sether@ii.uib.no
4 **University of Bergen, Norway**
telle@ii.uib.no

Abstract

It was recently shown [19] that satisfiability is polynomially solvable when the incidence graph is an interval bipartite graph (an interval graph turned into a bipartite graph by omitting all edges within each partite set). Here we relax this condition in several directions: First, we show an FPT algorithm parameterized by k for k-interval bigraphs, bipartite graphs which can be converted to interval bipartite graphs by adding to each node of one side at most k edges; the same result holds for the counting and the weighted maximization version of satisfiability. Second, given two linear orders, one for the variables and one for the clauses, we show how to find, in polynomial time, the smallest k such that there is a k-interval bigraph compatible with these two orders. On the negative side we prove that, barring complexity collapses, no such extensions are possible for CSPs more general than satisfiability. We also show NP-hardness of recognizing 1-interval bigraphs.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Satisfiability, interval graphs, FPT algorithms

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.14

1 Introduction

Constraint satisfaction problems (CSPs) such as satisfiability are both ubiquitous and difficult to solve. It is therefore essential to identify and exploit any special structure of instances that make CSPs susceptible to algorithmic techniques. One large class of such structured instances comprises CSPs whose constraints can be arranged in a linear manner, presumably reflecting temporal or spatial ordering of the real-life problem being modeled. A well known example is the *car sequencing* class of CSPs proposed by the French automobile manufacturer Renault in 2005 and reviewed in [22].

But defining what it means for a CSP to have "a linear structure" is not straightforward. The linear structure should be somehow reflected in the *incidence graph*, which is a bipartite graph that has a vertex for each variable and each constraint, and a variable vertex is

* This work was partially supported by a grant from the Peder Sather Center at UC Berkeley.
† Serge Gaspers is the recipient of an Australian Research Council (ARC) Future Fellowship (FT140100048) and acknowledges support under the ARC's Discovery Projects funding scheme (DP150101134).

adjacent to a constraint vertex if the corresponding variable occurs in the corresponding constraint. Previous work has considered satisfiability instances with incidence graphs of bounded treewidth or bounded cliquewidth [6, 16, 20, 21, 23]. Instances that are in some sense close to efficiently solvable instances have been studied in terms of backdoors [8, 24], in particular for CNF formulas that have a small number of variables whose instantiations give formulas of bounded treewidth [9]. An important special case of bounded treewidth is bounded pathwidth, a measure of how path-like a graph is and a strong indication of linear structure. Bounded cliquewidth is a stronger notion, in which the graph's cliques have a linear structure.

Recently another direction for defining linear structure in CSPs was proposed, based in a time honored graph-theoretic conception of linear structure: *interval graphs,* the intersection graphs of intervals on the line. Interval graphs are a well-known class of graphs, going back to the 1950s, used to model temporal reasoning [10], e.g. in resource allocation and scheduling [1]. However, the incidence graphs we care about are bipartite, and the only connected interval graphs that are bipartite are trees. A bipartite version of interval graphs was introduced by Harary et al. in 1982 [12]: An *interval bigraph* is, informally, a bipartite graph[1] in which each vertex is associated with an interval, and there is an edge between two vertices *on different sides* if and only if the corresponding intervals intersect. Interval bigraphs form a natural and fairly rich class of bipartite graphs, containing, *e.g.,* all bipartite permutation graphs, which have been shown to have unbounded cliquewidth and thus also unbounded treewidth and pathwidth [2].

Interval bigraphs have been studied quite extensively, and several important facts are known about them. First, they can be recognized in polynomial time: In 1997 Müller gave an algorithm with running time $\mathcal{O}(|V||E|^6(|V|+|E|)\log|V|)$ [14], and a 2012 technical report [17] gives an algorithm with running time $\mathcal{O}(|V|(|E|+|V|))$. Importantly, Hell and Huang [13] gave in 2004 a useful alternative characterization of interval bigraphs as all bipartite graphs whose set of vertices can be ordered on the line so that the set of neighbors of each vertex coincides with an interval whose high end is the position of the vertex (see Lemma 2 below for the formal statement).

Interval bigraphs constitute a natural basis for identifying an important class of CSPs possessing a linear order: Define *an interval CSP* as a CSP whose variable-constraint incidence graph is an interval bigraph. In [19] a general dynamic programming approach to solving a class of CSPs was developed, and one consequence of that framework is that satisfiability – even weighted MAXSAT and #SAT – on interval CNF formulae with m clauses and n variables can be solved in time $\mathcal{O}(m^3(m+n))$ (stated as Theorem 3 below). See also the work of Brault-Baron et al [3] which sets out the wider context of the results of [19].

The present work is about extending Theorem 3 in several natural directions:

1. Many CSPs are not interval CSPs. Can the definition of interval CSPs be extended usefully, so that a limited number of "faults" in the interval structure of CSPs is tolerated by polynomial time algorithms?
2. In a variety of applications there is a natural linear ordering of both variables and constraints, but not of their union. One well-studied application is car sequencing [22], where sliding-window constraints [18] naturally come with an ordering of the variables and the constraints. In other temporal or scheduling settings, variables can be ordered according to the discrete timestep they are relevant to. Even when one is only given an ordering of the variables, one can greedily order the constraints; for example by their

[1] We use "bigraph" and "bipartite graph" interchangeably.

earliest variable in the variable-ordering or more domain-dependent methods. Under what circumstances is it possible to merge these two linear orders into one, so that the resulting bipartite graph is an interval bigraph?

3. If an order as in (2) above does not exist, can at least a merged order be found so the resulting bipartite graph is as close as possible (presumably in some algorithmically useful sense as in (1) above) to an interval bigraph?
4. Finally, to what extent can these algorithmic results be extended to CSPs beyond satisfiability?

In this paper we address and largely resolve these questions. In particular, our contributions are the following:

1. We define a useful measure of how much the incidence graph of a CSP instance differs from an interval bigraph: The smallest number k such that the incidence bigraph becomes interval if each constraint vertex of the bigraph has at most k edges added to it. Deciding if $k \leq 1$ is NP-hard (Theorem 6) but we show (Theorem 9) that given an ordering certifying a value of k such instances of satisfiability with m clauses and n variables can be solved in $\mathcal{O}(m^3 4^k (m+n))$ time. Ditto for MAXSAT and #SAT; the exponential dependence on k is, of course, expected.
2. We give a simple characterization of when two linear orders, one for constraints and one for variables, can be merged so that the resulting total order satisfies the Hell-Huang characterization of interval bigraphs.
3. We also show that, if no such merging is possible, we can find in polynomial time – through a greedy algorithm – the minimum k such that the incidence graph becomes interval with the addition of at most k edges to each constraint vertex. Hence, in the case of satisfiability, if this minimum is bounded then polynomial algorithms result.
4. Finally, we show that the approach in (1) above – which started us down this path – does not work for general CSPs, in that CSP satisfiability is intractable even when the incidence graph has the same favorable structure as in (1), with bounded k (Theorem 10).

Definitions and Background

Since we mostly deal with satisfiability, we denote our bipartite graphs as $G = (\texttt{cla}, \texttt{var}, E)$, where `cla` stands for clauses and `var` for variables.

▶ **Definition 1.** A bipartite graph $G = (\texttt{cla}, \texttt{var}, E)$ is an *interval bigraph* if every vertex can be assigned an interval on the real line such that for all $x \in \texttt{var}$ and $c \in \texttt{cla}$ we have $xc \in E$ if and only if the corresponding intervals intersect. A Boolean formula in conjunctive normal form (CNF) is called an *interval CNF formula* if the corresponding incidence graph (`cla` the clauses, `var` the variables, E the incidences) is an interval bigraph.

A most interesting alternative characterization of interval bigraphs by Hell and Huang [13] is stated here, expressed in terms of interval CNF formulas.

▶ **Lemma 2** ([13]). *A CNF formula is an interval CNF formula if and only if its variables and clauses can be totally ordered (indicated by $<$) such that for any variable x appearing in a clause C:*

1. *if x' is a variable and $x < x' < C$ then x' also appears in C, and*
2. *if C' is a clause and $C < C' < x$ then x also appears in C'.*

We call an ordering of the variables and clauses of an interval CNF formula satisfying the lemma an *interval ordering*. Interval bigraphs can be recognized in polynomial time [14], see also [17].

2 k-interval Bigraphs

Recent work has articulated efficient algorithms in the dynamic programming style for interval CNF formulae.

▶ **Theorem 3** ([19]). *Given an interval CNF formula on n variables and m clauses and an interval ordering of it, #SAT and weighted* MaxSAT *can be solved in time $\mathcal{O}(m^3(m+n))$.*

Combining Theorem 3 with the recognition algorithm of [14] gives the following:

▶ **Corollary 4.** *Given a CNF formula, it can be decided if it is an interval CNF formula, and if so #SAT and weighted* MaxSAT *can be solved in polynomial time.*

We want to generalize this result to a larger class of formulae. To this end we introduce the following graph classes and formula classes, parametrized by $k \geq 1$.

▶ **Definition 5.** A bipartite graph $G = (\texttt{cla}, \texttt{var}, E)$ is a *k-interval bigraph* if we can add at most k edges to each vertex in `cla` such that the resulting bipartite graph is an interval bigraph. A CNF formula is called a *k-interval CNF formula* if its incidence graph (with clause vertices being `cla`) is a k-interval bigraph.

Note that 0-interval bigraphs are the interval bigraphs, and 1-interval bigraphs allow as many exceptions (added edges) as there are clauses. Unfortunately, the recognition problem for k-interval bigraphs becomes hard, already when $k = 1$. The proof is by reduction from the strongly NP-hard 3-Partition problem and is given in Section 5.

▶ **Theorem 6.** *Given a bipartite graph G and an integer k, deciding if G is a k-interval bigraph is NP-hard, even when $k = 1$.*

The alternative characterization of Lemma 2 can be extended to k-interval bigraphs.

▶ **Lemma 7.** *A CNF formula is a k-interval CNF formula if and only if its variables and clauses can be totally ordered such that for any clause C there are at most k variables x not appearing in C where either*
1. *a variable x' appears in C with $x' < x < C$, or*
2. *x appears in a clause C' with $C' < C < x$.*

Proof. The lemma follows directly from Definition 5 and Lemma 2. ◀

▶ **Definition 8.** For a k-interval CNF formula we call a total ordering of the kind guaranteed by Lemma 7 a k-interval ordering.

Our first algorithmic result is that, given a k-interval ordering of a k-interval CNF formula, #SAT and MaxSAT can be solved via a fixed-parameter tractable (FPT, see [4]) algorithm parameterized by k.

▶ **Theorem 9.** *Given a CNF formula and a k-interval ordering of it, we solve #SAT and weighted* MaxSAT *in time $\mathcal{O}(m^3 4^k (m+n))$.*

Proof. The full proof for #SAT and weighted MaxSAT is given in Section 4; here we give a straightforward construction establishing a weaker result for satisfiability only.

The basic observation is that the satisfiability of a CNF formula is not affected if a clause C is replaced by a particular set of clauses, defined next. Take any set S_C of $\ell \geq 0$ variables not occurring in C, and replace C with the 2^ℓ clauses of the form $(C \vee D_j) : j = 1, \ldots, 2^\ell$, where D_j ranges over the 2^ℓ possible clauses containing the variables in S_C. It is easy to see

that a truth assignment satisfies the new formula if and only if it satisfied the original one. It is further clear that the satisfiability of the formula remains unaffected if all clauses are so replaced, for different sets of variables and $\ell \geq 0$. Finally, if a CNF formula is k-interval, then it has such an equivalent variant whose incidence graph is an interval bigraph. An FPT algorithm (albeit for SAT only and with running time $\mathcal{O}(m^3 8^k(m2^k + n))$ instead of $\mathcal{O}(m^3 4^k(m + n))$) follows by applying Theorem 3. ◀

We next show that the k-interval structure is not helpful for general CSPs:

▶ **Theorem 10.** *Given a CSP instance I with variable-constraint incidence graph G and an interval bigraph G' obtained from G by adding at most k edges to each constraint vertex, deciding the satisfiability of I is $W[1]$-hard parameterized by k.*

Proof. Gottlob and Szeider [11] observed that CSP is $W[1]$-hard parameterized by the number of variables, recasting a W[1]-hardness proof for conjunctive queries in databases [15] to CSPs. The reduction is from clique, where for a graph $G = (V, E)$ and an integer parameter k, the question is whether there is a clique of size k in G, i.e., a set of k pairwise adjacent vertices. In the language of CSPs, their reduction construct a CSP with k variables $x_1, \ldots, x_k$, each with domain V. Intuitively, variable x_i represents the ith vertex of the clique we are looking for. For each pair of variables x_i, x_j with $i \neq j$, add a constraint with scope (x_i, x_j) whose constraint relation contains (u, v) iff $uv \in E$. Now, the CSP has a solution iff G has a clique of size k. Given a CSP instance with k variables, we can turn its incidence graph into an interval bigraph by adding all possible edges between variables and constraints. This creates a complete bipartite graph and adds at most k edges to each constraint vertex. ◀

3 Merging Linear Orders

Theorem 10 tells us that our ambition for new algorithmic results based on the concept of k-interval bigraph should be limited to CSPs of the satisfiability kind, while Theorem 6 suggests that the new concept of k-interval bigraph can only extend the class of solvable problems either in special cases, or indirectly, in specific contexts. In this section we derive an algorithmic result of the latter type.

Suppose that the real life situation modelled by the CNF formula has linearly ordered clauses, and linearly ordered variables, but there is no readily available linear order for both. That is, we assume the input comes with two linear orderings, one for the variables and one for the clauses. We wish to find the minimum value of k such that there exists a k-interval ordering compatible with both.

Problem: Merging to minimum k-interval bigraph ordering
Input: Bipartite graph $G = (\texttt{cla}, \texttt{var}, E)$, a total order of $\texttt{cla}$, and a total order of $\texttt{var}$
Output: The minimum k such that we can merge the two orders into a k-interval ordering of $\texttt{cla} \cup \texttt{var}$.

Consider first the case $k = 0$.

▶ **Lemma 11.** *If a formula is given with variable ordering, clause ordering, and incidences containing one of the obstructions in Figure 1, then it cannot be merged into an interval bigraph ordering.*

Proof. Consider the left-hand obstruction in Figure 1. We cannot insert z after C, since we get $A < C < z$ violating Condition 2 in Lemma 2. On the other hand, we cannot insert z before C, since we get $x < z < C$ violating Condition 1 in Lemma 2.

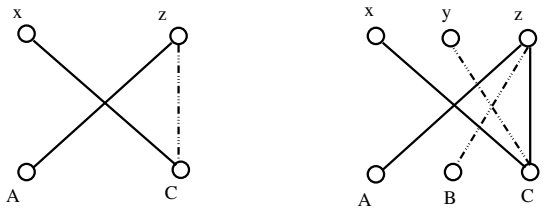


Figure 1 Obstructions to merging into an interval bigraph ordering: variables ordered $x < y < z$, clauses $A < B < C$, with solid lines indicating edges of the incidence graph and dotted lines indicating non-edges, with remaining possibilities being any combination of edges or non-edges.

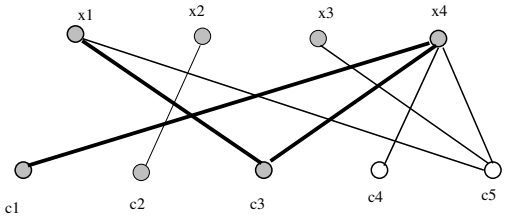


Figure 2 Consider the above input, with non-incidences indicated by non-edges. The bold edges and gray nodes show two overlapping obstructions as on the right side of Figure 1. Applying Lemma 2 these obstructions can be fixed in at least two ways: adding edge $c2x4$ by positioning $c3 < x2$; or adding edges $c3x2$ and $c3x3$ by positioning $c2 > x4$. In this last case a new obstruction appears, see Figure 3.

Consider the right-hand obstruction in Figure 1. We cannot insert z after B, since we get $A < B < z$ violating Condition 2 in Lemma 2. On the other hand, we cannot insert y before C, since we get $x < y < C$ violating Condition 1 in Lemma 2. Thus, since $B < C$ this leaves no place to insert z without violating Lemma 2. ◀

It turns out that, if there are no obstructions as in Figure 1 then MERGING TO MINIMUM k-INTERVAL BIGRAPH ORDERING has a solution with $k = 0$. Thus, for any instance where the solution has value $k > 0$ we can view the task as one of iteratively adding edges until the result has no obstruction as in Figure 1. On the face of it this is non-trivial, as there is more than one way of fixing an obstruction, with varying edge costs, and some ways may lead to a new obstruction appearing. For an example of this see Figures 2 and 3.

Nevertheless, a greedy approach will efficiently solve MERGING TO MINIMUM k-INTERVAL BIGRAPH ORDERING. Let us describe it. Assume the input ordering on variables and clauses is $x_1, ..., x_n$ and $c_1, ..., c_m$. All orderings we consider will be compatible with these input orderings. The greedy strategy works as follows. Start with $k = 0$ and consider clauses by decreasing index c_m, c_{m-1}, etc. Insert c_i among the variables in the highest possible position, below the position of c_{i+1}, that does not lead to more than k edges being added to c_i. If no such position exists then increase k and start all over again with c_m. The correctness of this strategy relies on the following observation.

▶ **Observation 12.** *For any fixed position of c_i among the variables the number of edges we must add to clause c_i does not depend on where the other clauses are inserted, as long as $c_1, ..., c_{i-1}$ end up below c_i and $c_{i+1}, ..., c_m$ above c_i.*

Proof. By Lemma 7 we must add to c_i exactly one edge for each variable x not appearing in c_i, where x satisfies one of the two conditions stated in Lemma 7. For the second condition

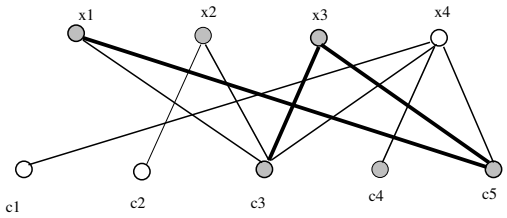


Figure 3 Assume we fixed the obstructions from Figure 2 by adding edges $c3x2$ and $c3x3$. We then get a new obstruction based in bold edges and gray nodes.

Greedy Algorithm for merging to minimum k-interval bigraph ordering

input: $G = (\texttt{cla}, \texttt{var}, E)$, orderings $\texttt{cla} = c_1, c_2, ..., c_m$ and $\texttt{var} = x_1, x_2, ..., x_n$
output: minimum k such that `cla` and `var` can be merged into a k-interval ordering

$q := -1$;
success := *false*;
while not *success*
  $q := q + 1$;
  start with the ordering $x_1, x_2, ..., x_n$;
  for $i = m$ **downto** 1
    insert c_i at the highest position, below c_{i+1}, where $EdgesAdded(c_i) \leq q$;
    if no such position exists for clause c_i then **break** out of the for loop;
  if all clauses have been inserted then *success* := *true*;
output q;

$EdgesAdded(C)$:= number of variables satisfying one of the conditions of Lemma 7

Figure 4 Greedy Algorithm for merging to minimum k-interval bigraph ordering.

note that C' can be any of $c_1, ..., c_{i-1}$ but no other clause. For the first condition note that it does not depend on any other clause, only on the position of c_i among the variables. ◀

In our greedy strategy, when deciding where to insert c_i the only restriction imposed on us by earlier decisions is that c_i must end up below the position of c_{i+1}. To allow the maximum degree of freedom we simply ensure that we have inserted c_{i+1} in the highest possible position. The pseudocode is in Figure 4.

▶ **Theorem 13.** *The Greedy Algorithm for* Merging to minimum k-interval bigraph ordering *is correct and can be implemented to run in time* $\mathcal{O}(|E| \log k)$.

Proof. Let us first argue for correctness. Consider an iteration of the inner loop that successfully found a position for clause c_i among the variables. For the current value of q it is not possible to insert c_i higher than this position without some c_j needing more than q edges added, for some $i \leq j \leq m$. This is in fact a loop invariant, as we inserted the clauses of higher index in the highest possible positions under exactly this constraint, and by Observation 12 their position does not influence the number of edges added to other clauses. Similarly, if for some c_i and current value of q we encounter 'no such position exists' then in any ordering of $\texttt{cla} \cup \texttt{var}$ compatible with the input orders there will be some $c_j, i \leq j \leq m$ which will need more than k edges added. Thus, when the algorithm successfully finds positions for all clauses then the current value of q is the correct answer.

Let us now argue for the running time. For the $\log k$ factor, rather than iterating on q until we succeed, we can search for the minimum k by what is known as galloping search, i.e. try q equal to 1, 2, 4, 8, etc until we succeed for an integer q, and then do binary search in

Deciding if we can merge to a q-interval ordering, for fixed q, in $\mathcal{O}(|E|)$ time

```
∀x ∈ var: live(x) := number of clauses x appears in
∀c ∈ cla: var(c) := the set of variables in c
          low(c) := i, lowest i with x_i ∈ var(c)
livevar := 0;
t := n;
start with the ordering x_1, x_2, ..., x_n;
for i := m downto 1
  inserted := false;
  while not inserted          /* try to insert c_i after x_t */
      if livevar + t − low(c_i) − |var(c_i)| ≤ q then
        insert c_i after x_t;
        inserted := true;
        ∀x_j ∈ var(c_i) : live(x_j) := live(x_j) − 1;
                          if live(x_j) = 0 and j > t then livevar := livevar − 1;
      else if t = 0 then halt: 'no for this value of q';
      else t := t − 1; if live(x_t) > 0 then livevar := livevar + 1;
'yes for this value of q';
```

Figure 5 Deciding if we can merge to a q-interval ordering, for fixed q, in $\mathcal{O}(|E|)$ time.

the interval $[q/2..q]$. To decide on positions for the clauses in time $\mathcal{O}(|E|)$, for a fixed q, we need several program variables. The pseudocode is in Figure 5.

We maintain for each $x \in \texttt{var}$ the value $live(x)$ as the number of live clauses x appears in, where a live clause is one whose position has not been decided yet. Also, we maintain $livevar$ as the number of variables indexed higher than the current x_t and appearing in a live clause. Finally, $var(c_i)$ are the variables in clause c_i and $low(c_i)$ the index of its lowest indexed variable. The number of edges needed for c_i if inserted immediately after x_t is then

$$EdgesAdded(c_i) = livevar + t - low(c_i) - |var(c_i)|\,.$$

This is so since we must add to c_i exactly one edge for each variable x not appearing in c_i, where x satisfies one of the two conditions stated in Lemma 7. The second condition counts the number of variables indexed higher than the current x_t and appearing in some clause indexed lower than c_i, i.e. $livevar$, minus the number of variables in c_i of index higher than t. The first condition counts the number of variables strictly between $x_{low(c_i)}$ and x_{t+1}, i.e. $t - low(c_i)$, minus the number of variables in c_i of index t or less. Summing these two counts we get the above. ◀

4 Proof of Theorem 9

In this section we prove Theorem 9, namely that if we are given a CNF formula and a k-interval ordering of it, we can solve #SAT and weighted MaxSAT in time $\mathcal{O}(m^3 4^k (m+n))$.

We first need to introduce the linear ps-width of a formula. We start with the related notion of ps-value of a CNF formula F on variables var and clauses cla. For an assignment τ of var, we denote by $\texttt{sat}(F,\tau)$ the inclusion maximal set $\mathcal{C} \subseteq \texttt{cla}$ so that each clause in $\mathcal{C}$ is satisfied by τ. Such a subset $\mathcal{C} \subseteq \texttt{cla}$ is called *projection satisfiable*. The ps-value of F is defined to be the number of projection satisfiable subsets of clauses, *i.e.* $|\texttt{PS}(F)|$, where

$$\texttt{PS}(F) = \{\texttt{sat}(F,\tau) : \tau \text{ is an assignment of } \texttt{var}\} \subseteq 2^{\texttt{cla}}\,.$$

Now, consider a linear ordering $e_1, e_2, ..., e_{n+m}$ of $\texttt{var} \cup \texttt{cla}$. For any $1 \leq i \leq n+m$ we define two disjoint subformulas $F_1(i)$ and $F_2(i)$ crossing the cut between $\{e_1, ..., e_i\}$ and $\{e_{i+1}, ..., e_{n+m}\}$. We define $F_1(i)$ to be the subformula we get by removing from F all clauses not in $\{e_1, ..., e_i\}$ followed by removing from the remaining clauses each literal of a variable not in $\{e_{i+1}, ..., e_{n+m}\}$, and we define $F_2(i)$ vice-versa, as the subformula we get by removing from F all clauses not in $\{e_{i+1}, ..., e_{n+m}\}$ followed by removing from the remaining clauses each literal of a variable not in $\{e_1, ..., e_i\}$.

The ps-width of this linear ordering is defined to be the maximum ps-value over all the $2(n+m)$ subformulas $F_1(1), F_2(1), F_1(2), ..., F_2(n+m)$ that cross a cut of the ordering. The linear ps-width of F is defined to be the minimum ps-width of all linear orderings of $\texttt{var} \cup \texttt{cla}$.

To prove Theorem 9, we will show that the input has linear ps-width at most $m2^k + 1$ and then we can apply the following result by [19].

▶ **Theorem 14** ([19]). *Given a CNF formula F with n variables* var *and m clauses* cla*, and a linear ordering of* $\texttt{cla} \cup \texttt{var}$ *showing that F has linear* ps*-width at most p, we solve* #SAT *and weighted* MaxSAT *in time $\mathcal{O}(p^2 m(m+n))$.*

Before giving the lemma that bounds the linear ps-width of the input formula, we state a useful result.

▶ **Lemma 15** ([19]). *Any interval ordering of an interval CNF formula has* ps*-width no more than the number of its clauses plus one.*

▶ **Lemma 16.** *Let F be a k-interval CNF formula on m clauses. Then any k-interval ordering of it has* ps*-width at most $m2^k + 1$.*

Proof. Starting from F on m clauses and its k-interval ordering π we first construct an interval CNF formula F' with at most $m2^k$ clauses. Any clause C of F for which Lemma 7 prescribes $k' \leq k$ added edges from C to some k' variables, will be replaced in F' by a set of $2^{k'}$ clauses consisting of the clause C extended by all linear combinations of these k' variables. Note that F' is then an interval CNF formula with an interval ordering π' we get from π by naturally expanding a clause C in π to the $2^{k'}$ clauses, in any order, that replace C in F'.

Applying Lemma 15 all we need to finalize our proof is to show that the ps-width of the k-interval ordering π of F is no larger than the ps-width of the interval ordering π' of F'. To do this we must consider cuts of π.

Consider subformulas $F_1(i)$ and $F_2(i)$ of F crossing a cut of π. We show that for the corresponding cut in π' (i.e. we cut π' in the corresponding place without splitting any of the expanded set of clauses) the ps-values of the subformulas F_1' and F_2' of F' associated with this cut has ps-value no smaller than the ps-values of $F_1(i)$ and $F_2(i)$. That is $|\texttt{PS}(F_1(i))| \leq |\texttt{PS}(F_1')|$ and $|\texttt{PS}(F_2(i))| \leq |\texttt{PS}(F_2')|$. Note that the variables of $F_1(i)$ and F_1' are the same, and similarly the variables of $F_2(i)$ and F_2' are the same. W.l.o.g., we focus on $F_1(i)$ and F_1', which we assume have variables $\texttt{var}_1$.

We need to show that if two assignments a, b of $\texttt{var}_1$ have $\texttt{sat}(F_1(i), a) \neq \texttt{sat}(F_1(i), b)$ then also $\texttt{sat}(F_1', a) \neq \texttt{sat}(F_1', b)$. W.l.o.g., assume some clause $C \in \texttt{sat}(F_1(i), a)$ but $C \notin \texttt{sat}(F_1(i), b)$. We show that we can find a clause C' that distinguishes a and b in F_1' as well. Clause C of $F_1(i)$ comes from an original clause (possibly larger, since C lives across a cut) in F. Assume this original clause was expanded in F' to $2^{k'}$ clauses, for some $k' \leq k$, by extending it with all linear combinations of the k' new variables. Depending on which variables are on the other side of the cut the clause C of $F_1(i)$ has been expanded to a set of $2^{k''}$, for some $k'' \leq k'$, clauses in F_1', still consisting of all linear combinations of the k''

variables not in C. Since a satisfies C and C is a part of all these expanded clauses we have assignment a satisfying all of them. Since b does not satisfy C there will be exactly one of these $2^{k''}$ clauses that are not satisfied by b, namely the one where the linear combination of the new variables is falsified by assignment b. This means that $\texttt{sat}(F'_1, a) \neq \texttt{sat}(F'_1, b)$.

Thus the ps-width of the k-interval ordering of F is no more than the ps-width of the interval ordering of F' and we are done. ◀

Combining Theorem 14 with Lemma 16 we arrive at Theorem 9. Combining with Theorem 13 we get the following.

▶ **Corollary 17.** *Given a CNF formula and two total orderings, one for its m clauses and one for its n variables, we can in polynomial time find the minimum k such that these two orders can be merged into a k-interval ordering and then solve #SAT and MaxSAT in time $\mathcal{O}(m^3 4^k (m+n))$.*

5 Proof of Theorem 6

In this section we prove Theorem 6, namely that it is NP-hard to recognize k-interval bigraphs, already for $k = 1$.

Proof. We give a polynomial time reduction from the 3-Partition problem, which is strongly NP-hard [7]. Given an integer b, a set A of $3n$ elements, and a positive integer $s(a)$ for each $a \in A$ such that $b/4 < s(a) < b/2$ for each $a \in A$ and $\sum_{a \in A} s(a) = n \cdot b$, the question is whether A can be partitioned into disjoint sets $A_1, \ldots, A_n$ such that $\sum_{a \in A_i} s(a) = b$ for each $i \in \{1, \ldots, n\}$.

For a 3-Partition instance (b, A, s), we construct an instance $G = (V, E)$ for the 1-interval bigraph recognition problem as follows. We assume, w.l.o.g., that $b \geq 4$, and therefore, $s(a) > 1$ for each $a \in A$.

We add a set of *slot* vertices $S = \bigcup_{i=1}^{n} S_i$ with $S_i = \{s_{i,1}, \ldots, s_{i,b+1}\}$. For all i, j with $1 \leq i \leq n$ and $1 \leq j \leq b$ we add a vertex $\ell_{i,j}$ that is adjacent to both $s_{i,j}$ and $s_{i,j+1}$, so that $(s_{i,1}, \ell_{i,1}, s_{i,2}, \ell_{i,2}, \ldots, s_{i,b}, \ell_{i,b}, s_{i,b+1})$ is a path for each $i \in \{1, \ldots, n\}$.

For each $i \in \{1, \ldots, n-1\}$ we add a *delimiter* vertex s_i^d, and two vertices $\ell_i^{d,1}$ and $\ell_i^{d,2}$. We make $\ell_i^{d,1}$ adjacent to $s_{i,b}$, $s_{i,b+1}$, s_i^d, and $s_{i+1,1}$ and we make $\ell_i^{d,2}$ adjacent to $s_{i,b+1}$, s_i^d, $s_{i+1,1}$, and $s_{i+1,2}$. The set of delimiter vertices is $D = \bigcup_{i=1}^{n-1} \{s_i^d\}$.

We add a *track* vertex t that is adjacent to each vertex in $S \cup D \setminus \{s_1^d\}$.

We add *left anchor* vertices a^l, $\ell^{a,l}$, and make $\ell^{a,l}$ adjacent to a^l, $s_{1,1}$, and $s_{1,2}$. Symmetrically, we add *right anchor* vertices a^r, $\ell^{a,r}$, and make $\ell^{a,r}$ adjacent to a^r, $s_{n,b+1}$, and $s_{n,b}$. See Figure 6 for an illustration of the graph constructed so far.

For each element $a \in A$, we add a *numeral* gadget which is obtained from a path on $2 \cdot s(a) + 1$ new vertices $(\ell^n_{a,0}, n_{a,1}, \ell^n_{a,1}, \ldots, n_{a,s(a)-1}, \ell^n_{a,s(a)-1}, n_{a,s(a)}, \ell^n_{a,s(a)})$ and the track vertex t is made adjacent to $n_{a,1}, \ldots, n_{a,s(a)}$. See Figure 7 for an illustration of a numeral gadget.

We will now show that (b, A, s) is a Yes-instance for 3-Partition if and only if G is a 1-interval bigraph. For the forward direction, consider a solution $A_1, \ldots, A_n$ to the 3-Partition instance. We construct an interval representation following the scheme outlined in Figure 6, which is missing the numeral gadgets. Now, for each $A_i = \{x, y, z\}$, we can intersperse the intervals $s_{i,1}, \ldots, s_{i,s(x)+1}$ with the intervals $n_{x,1}, \ldots, n_{x,s(x)}$, intersperse the intervals $s_{i,s(x)+1}, \ldots, s_{i,s(x)+s(y)+1}$ with the intervals $n_{y,1}, \ldots, n_{y,s(y)}$, and intersperse the intervals $s_{i,s(x)+s(y)+1}, \ldots, s_{i,b+1}$ with the intervals $n_{z,1}, \ldots, n_{z,s(z)}$. In this way, each vertex $\ell_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq b$, is non-adjacent to exactly one vertex (from $\{n_{a,1}, \ldots, n_{a,s(a)} : a \in A\}$)

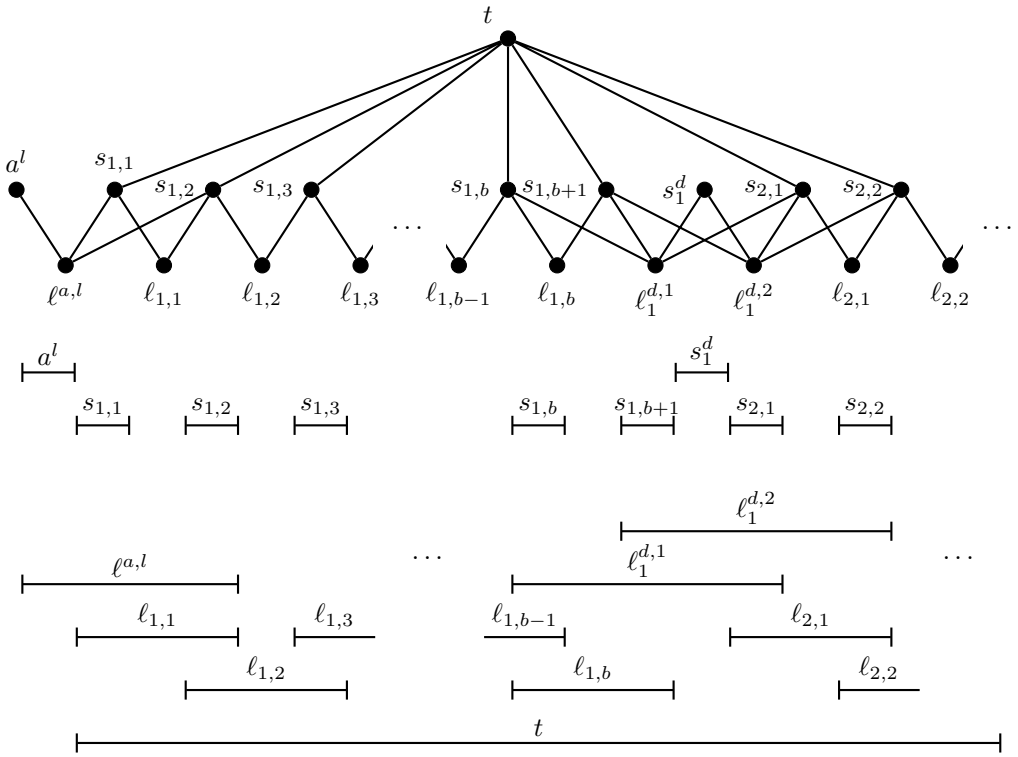


Figure 6 A part of the graph constructed by our reduction and a corresponding 1-interval representation formed by the all the vertices except the numeral gadgets. The top two rows of intervals correspond to the vertices in one partite set of the bipartition and the bottom rows to vertices in the other partite set.

whose corresponding intervals overlap, and each vertex $\ell^n_{a,j}$, $a \in A$, $1 \leq j \leq s(a) - 1$, is non-adjacent to exactly one vertex (from $\{s_{i,2}, \ldots, s_{i,b} : 1 \leq i \leq n\}$) whose corresponding intervals overlap. This certifies that G is a 1-interval bigraph.

For the backward direction, we observe that our construction enforces the rigid structure from Figure 6. Intuitively, for each $i \in \{1, \ldots, n\}$, the vertices $\ell_{i,j}$ enforce an ordering of the intervals corresponding to the vertices in S_i, and the delimiters glue the different sections of S_i vertices together in a linear fashion. Observe that between two vertices $s_{i,j}$ and $s_{i,j+1}$, we can still insert one vertex if it is adjacent to t, and we exploit this property to intersperse the numerals. The anchor vertices are used to stretch the structure of the slot vertices beyond the left and the right of the track t. This ensures then that the numerals need to be interspersed with the slots. Since there are no elements $a \in A$ with $s(a) = 1$, it is also not possible for a numeral gadget to intersperse a section S_i of slot vertices before $s_{i,1}$ or after $s_{i,b+1}$. In addition, the delimiters ensure that numerals do not straddle different S_i's. Therefore, we can obtain a solution to the 3-Partition instance by setting A_i to the elements from A that we used to construct the numeral gadgets that are interspersed with the slots in S_i. ◀

6 Discussion

The algorithmic challenge of CNF satisfiability and constraint satisfaction is central in both computational theory and practice, and new angles of attack to these age-old problems keep emerging. Here we focused on instances which possess a linear structure, and we

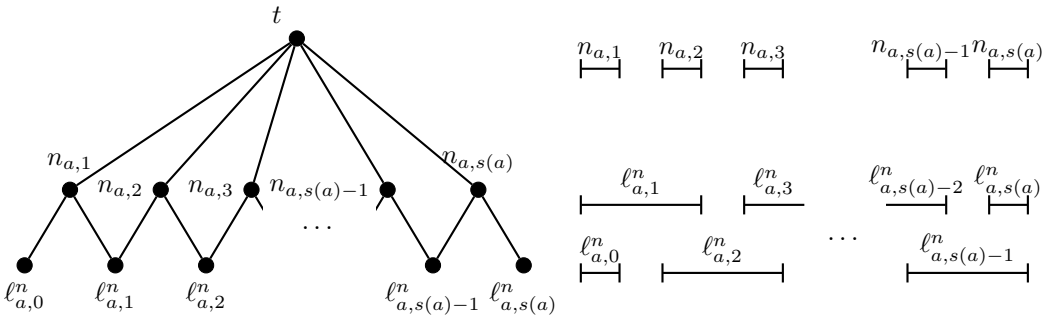


Figure 7 A numeral gadget for element $a \in A$.

proposed a new approach to dealing with local departures from such structure, as well as for deducing linear structure from partial evidence; we also identified complexity obstacles to fully exploiting and extending our approach. Our work raises several questions:

- What if only one side of the bipartite incidence graph is ordered? Say we are given an ordering of variables and asked if the clauses can be inserted so as to yield a k-interval ordering. For the case $k = 0$ we can use the obstructions in Figure 1 to guide us towards a linear ordering also of the clauses, e.g., for a pair of clauses A, C with two variables $x < z$ where xC, zA are edges and zC is a non-edge we must place C before A. We believe such an approach should solve the $k = 0$ case in polynomial time, but we are less optimistic about the general case of minimizing k.
- What if we are given a partial order, with some special properties, on variables and clauses? Note that already the approach for $k = 0$ hinted at above could yield a situation with a linear order on variables and a partial order on clauses.
- For which industrial CNF instances can we find k-interval orderings for low values of k? Our greedy algorithm for merging two linear orders to a minimum k-interval ordering is practical and can be applied to large instances in the SAT corpora. In light of the hardness result for recognizing 1-interval bigraphs, heuristics or domain expertise could be used to generate orders for clauses and variables, when they are not already given. For a preliminary experimental study using heuristics to compute linear orderings that produce k-interval bigraphs with small k, we refer to [5].
- Which other classes of interval bigraph CSP instances can be solved efficiently? Our hardness result is for general CSPs with large domains. For CSPs with Boolean domains we can show a similar hardness result albeit not for k-interval bigraph instances, instead for a different notion of "imperfection" where we are given k pairs of clause vertices in the incidence graph such that merging each such pair results in an interval bigraph.

References

1 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC'00, pages 735–744, New York, NY, USA, 2000. ACM. doi:10.1145/335305.335410.

2 Andreas Brandstädt and Vadim V. Lozin. On the linear structure and clique-width of bipartite permutation graphs. *Ars Comb.*, 67, 2003.

3 Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding model counting for beta-acyclic CNF-formulas. In *32nd International Symposium on Theoretical Aspects of*

Computer Science, STACS 2015, March 4-7, 2015, Germany, volume 30 of *LIPIcs*, pages 143–156. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.143.

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

5 Christian Egeland. Algorithms for linearly ordered boolean formulas. Master's thesis, University of Bergen, 2016. URL: http://hdl.handle.net/1956/12667.

6 Eldar Fischer, Johann A. Makowsky, and Elena V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008. doi:10.1016/j.dam.2006.06.020.

7 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

8 Serge Gaspers and Stefan Szeider. Backdoors to satisfaction. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond – Essays Dedicated to Mike Fellows on His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 287–317. Springer, 2012. doi:10.1007/978-3-642-30891-8_15.

9 Serge Gaspers and Stefan Szeider. Strong backdoors to bounded treewidth SAT. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 489–498. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.59.

10 Martin Charles Golumbic and Ron Shamir. Complexity and algorithms for reasoning about time: A graph-theoretic approach. *J. ACM*, 40(5):1108–1133, November 1993. doi:10.1145/174147.169675.

11 Georg Gottlob and Stefan Szeider. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction and database problems. *Comput. J.*, 51(3):303–325, 2008. doi:10.1093/comjnl/bxm056.

12 Frank Harary, Jerald A Kabell, and Frederick R McMorris. Bipartite intersection graphs. *Commentationes Mathematicae Universitatis Carolinae*, 23(4):739–745, 1982.

13 Pavol Hell and Jing Huang. Interval bigraphs and circular arc graphs. *Journal of Graph Theory*, 46(4):313–327, 2004. doi:10.1002/jgt.20006.

14 Haiko Müller. Recognizing interval digraphs and interval bigraphs in polynomial time. *Discrete Applied Mathematics*, 78(1-3):189–205, 1997. doi:10.1016/S0166-218X(97)00027-9.

15 Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999. doi:10.1006/jcss.1999.1626.

16 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for CNF formulas of bounded modular treewidth. In *STACS*, volume 20 of *LIPIcs*, pages 55–66. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.STACS.2013.55.

17 Arash Rafiey. Recognizing interval bigraphs by forbidden patterns. *CoRR*, abs/1211.2662, 2012.

18 Jean-Charles Régin and Jean-Francois Puget. A filtering algorithm for global sequencing constraints. In *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP 1997)*, volume 1330 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1997. doi:10.1007/BFb0017428.

19 Sigve Hortemo Sæther, Jan Arne Telle, and Martin Vatshelle. Solving #SAT and MAXSAT by dynamic programming. *J. Artif. Intell. Res. (JAIR)*, 54:59–82, 2015. doi:10.1613/jair.4831.

20 Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010. doi:10.1016/j.jda.2009.06.002.

21 Friedrich Slivovsky and Stefan Szeider. Model counting for formulas of bounded clique-width. In Leizhen Cai, Siu-Wing Cheng, and Tak Wah Lam, editors, *ISAAC*, volume 8283 of *Lecture Notes in Computer Science*, pages 677–687. Springer, 2013. doi:10.1007/978-3-642-45030-3_63.

22 Christine Solnon, Van-Dat Cung, Alain Nguyen, and Christian Artigues. The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. *European Journal of Operational Research*, 191(3):912–927, 2008. doi:10.1016/j.ejor.2007.04.033.

23 Stefan Szeider. On fixed-parameter tractable parameterizations of SAT. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT 2003*, volume 2919 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2003. doi:10.1007/978-3-540-24605-3_15.

24 Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In Georg Gottlob and Toby Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 1173–1178. Morgan Kaufmann, 2003.

Cutwidth: Obstructions and Algorithmic Aspects*†

Archontia C. Giannopoulou[1], Michał Pilipczuk[2], Jean-Florent Raymond[3], Dimitrios M. Thilikos[4], and Marcin Wrochna[5]

1 **Technische Universität Berlin, Berlin, Germany**
archontia.giannopoulou@tu-berlin.de

2 **Institute of Informatics, University of Warsaw, Poland**
michal.pilipczuk@mimuw.edu.pl

3 **Institute of Informatics, University of Warsaw, Poland; and AlGCo project team, CNRS, LIRMM, Montpellier, France**
jean-florent.raymond@mimuw.edu.pl

4 **AlGCo project team, CNRS, LIRMM, Montpellier, France; and Department of Mathematics, National and Kapodistrian University of Athens, Greece**
sedthilk@thilikos.info

5 **Institute of Informatics, University of Warsaw, Poland**
m.wrochna@mimuw.edu.pl

Abstract

Cutwidth is one of the classic layout parameters for graphs. It measures how well one can order the vertices of a graph in a linear manner, so that the maximum number of edges between any prefix and its complement suffix is minimized. As graphs of cutwidth at most k are closed under taking immersions, the results of Robertson and Seymour imply that there is a finite list of minimal immersion obstructions for admitting a cut layout of width at most k. We prove that every minimal immersion obstruction for cutwidth at most k has size at most $2^{O(k^3 \log k)}$.

As an interesting algorithmic byproduct, we design a new fixed-parameter algorithm for computing the cutwidth of a graph that runs in time $2^{O(k^2 \log k)} \cdot n$, where k is the optimum width and n is the number of vertices. While being slower by a $\log k$-factor in the exponent than the fastest known algorithm, due to Thilikos, Bodlaender, and Serna [17, 18], our algorithm has the advantage of being simpler and self-contained; arguably, it explains better the combinatorics of optimum-width layouts.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases cutwidth, obstructions, immersions, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.15

* A full version of the paper is available at http://arxiv.org/abs/1606.05975.

† This work was partially done while A. C. Giannopoulou was holding a post-doc position at Warsaw Center of Mathematics and Computer Science. The research of A. C. Giannopoulou has been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant DISTRUCT, agreement No 648527). The research of Mi. Pilipczuk and M. Wrochna is supported by the Polish National Science Center grant SONATA UMO-2013/11/D/ST6/03073. The research of J.-F. Raymond is supported by the Polish National Science Center grant PRELUDIUM UMO-2013/11/N/ST6/02706. Mi. Pilipczuk is supported by the Foundation for Polish Science (FNP) via the START stipend programme.

1 Introduction

The cutwidth of a graph is defined as the minimum possible *width* of a linear ordering of its vertices, where the width of an ordering σ is the maximum, among all the prefixes of σ, of the number of edges that have exactly one vertex in a prefix. Due to its natural definition, cutwidth has various applications in a range of practical fields of computer science: whenever data is expected to be roughly linearly ordered and dependencies or connections are local, one can expect the cutwidth of the corresponding graph to be small. These applications include circuit design, graph drawing, bioinformatics, and text information retrieval; we refer to the survey of layout parameters of Díaz, Petit, and Serna [5] for a broader discussion.

As finding a layout of optimum width is NP-hard [7], the algorithmic and combinatorial aspects of cutwidth were intensively studied. There is a broad range of polynomial-time algorithms for special graph classes [10, 11, 21], approximation algorithms [14], and fixed-parameter algorithms [17, 18]. In particular, Thilikos, Bodlaender, and Serna [17, 18] proposed a fixed-parameter algorithm for computing the cutwidth of a graph that runs[1] in time $2^{O(k^2)} \cdot n$, where k is the optimum width and n is the number of vertices. Their approach is to first compute the pathwidth of the input graph, which is never larger than the cutwidth. Then, the optimum layout can be constructed by an elaborate dynamic programming procedure on the obtained path decomposition. To upper bound the number of relevant states, the authors had to understand how an optimum layout can look in a given path decomposition. For this, they borrow the technique of *typical sequences* of Bodlaender and Kloks [3], which was introduced for a similar reason, but for pathwidth and treewidth instead of cutwidth.

Since the class of graphs of cutwidth at most k is closed under immersions, and the immersion order is a well-quasi ordering of graphs[2] [15], it follows that for each k there exists a *finite* obstruction set $\mathcal{L}_k$ of graphs such that a graph has cutwidth at most k if and only if it does not admit any graph from $\mathcal{L}_k$ as an immersion. However, this existential result does not give any hint on how to generate, or at least estimate the sizes of the obstructions. The sizes of obstructions are important for efficient treatment of graphs of small cutwidth; this applies also in practice, as indicated by Booth et al. [4] in the context of VLSI design.

The estimation of sizes of minimal obstructions for several graph parameters has been studied before. For minor-closed parameters pathwidth and treewidth, Lagergren [13] showed that any minimal minor obstruction to admitting a path decomposition of width k has size at most single-exponential in $O(k^4)$, whereas for treewidth he showed an upper bound double-exponential in $O(k^5)$. Less is known about immersion-closed parameters, like cutwidth. Govindan and Ramachandramurthi [9] showed that the number of minimal immersion obstructions for the class of graphs of cutwidth at most k is at least $3^{k-7}+1$. The construction in [9] exemplifies minimal obstructions for cutwidth at most k with $(3^{k-5}-1)/2$ vertices. To the best of our knowledge, nothing was known about upper bounds for the cutwidth case.

Results on obstructions. Our main result concerns the sizes of obstructions for cutwidth.

▶ **Theorem 1.** *Suppose a graph G has cutwidth larger than k, but every graph with fewer vertices or edges (strongly) immersed in G has cutwidth at most k. Then G has at most $2^{O(k^3 \log k)}$ vertices and edges.*

---

[1] Thilikos, Bodlaender, and Serna [17, 18] do not specify the parametric dependence of the running time of their algorithm. A careful analysis of their algorithm yields the above claimed running time bound.

[2] All graphs considered in this paper may have parallel edges, but no loops.

The above result immediately gives the same upper bound on the sizes of graphs from the minimal obstruction sets $\mathcal{L}_k$ as they satisfy the prerequisites of Theorem 1. This somewhat matches the $(3^{k-5}-1)/2$ lower bound of Govindan and Ramachandramurthi [9].

Our approach for Theorem 1 follows the technique used by Lagergren [13] to prove that minimal minor obstructions for pathwidth at most k have sizes single-exponential in $O(k^4)$. Intuitively, the idea of Lagergren is to take an optimum decomposition for a minimal obstruction, which must have width $k+1$, and to assign to each prefix of the decomposition one of finitely many "types", so that two prefixes with the same type "behave" in the same manner. If there were two prefixes, one being shorter than the other, with the same type, then one could replace one with the other, thus obtaining a smaller obstruction. Hence, the upper bound on the number of types, being double-exponential in $O(k^4)$, gives some upper bound on the size of a minimal obstruction. This upper bound can be further improved to single-exponential by observing that types are ordered by a natural domination relation, and the shorter a prefix is, the weaker is its type. An important detail is that one needs to make sure that the replacement can be modeled by minor operations. For this, Lagergren uses the notion of *linked path decompositions* (a weaker variant of *lean path decompositions*; cf. [19, 1]).

To prove Theorem 1, we perform a similar analysis of prefixes of an optimum ordering of a minimal obstruction. We show that prefixes can be categorized into a bounded number of types, depending on their "behavior". Provided two prefixes with equally strong type appear one after the other, we can "unpump" the part of the graph in their difference.

To make sure that unpumping is modeled by taking an immersion, we define *linked orderings* for cutwidth and reprove the analogue of the result of Thomas [19] (see [1] for simplified proofs): there is always an optimum-width ordering that is linked. We remark this already follows from more general results on submodular functions: the same is true for parameters like *linear rank-width*, as observed by Kanté and Kwon [12], which in turns follows from the proof of an analogous theorem of Geelen et al. [8] that applies to branch-decompositions, and thus, e.g., to parameters known as *branch-width* and *carving-width*.

The proof of the upper bound on the number of types essentially boils down to the following setting. We are given a graph G and a subset X of vertices, such that at most ℓ edges have exactly one endpoint in X. The question is how X can look like in an optimum-width ordering of G. We prove that there is always an ordering where X is split into at most $O(k\ell)$ blocks, where k is the optimum width. This allows us to store the relevant information on the whole X in one of a constant number of types (called *bucket interfaces*). The swapping argument used in this proof holds the essence of the typical sequences technique of Bodlaender and Kloks [3], while being, in our opinion, more natural and easier to understand.

As an interesting byproduct, we can also use our understanding to treat the problem of removing edges to get a graph of small cutwidth. More precisely, for parameters w, k, we consider the class of all graphs G, such that w edges can be removed from G to obtain a graph of cutwidth at most k. We prove that for every constant k, the minimal (strong) immersion obstructions for this class have sizes bounded *linearly* in w. Moreover we give an exponential lower bound to the number of these obstructions. Due to the auxiliary character of these results, we defer the precise statement and discussion to the full version of this paper.

Algorithmic results. Consider the following "compression" problem: given a graph G and its ordering σ of width ℓ, we would like to construct, if possible, a new ordering of the vertices of G of width at most k, where $k < \ell$. Then the types defined above essentially match states that would be associated with prefixes of σ in a dynamic programming algorithm solving this

problem. Alternatively, one can think of building an automaton that traverses the ordering σ of width ℓ while constructing an ordering of G of width at most k. Hence, our upper bound on the number of types can be directly used to limit the state space in such a dynamic programming procedure/automaton, yielding an FPT algorithm for the above problem.

With this result in hand, it is not hard to design of an exact FPT algorithm for cutwidth. One could introduce vertices one by one to the graph, while maintaining an ordering of optimum width. Each time a new vertex is introduced, we put it anywhere into the ordering, and it can be argued that the new ordering has width at most three times larger than the optimum. Then, the dynamic programming algorithm sketched above can be used to "compress" this approximate ordering to an optimum one in linear FPT time.

The above approach yields a quadratic algorithm. To match the optimum, linear running time, we use a similar trick as Bodlaender in his linear-time algorithm for computing the treewidth of the graph [2]. Namely, we show that instead of processing vertices one by one, we can proceed recursively by removing a significant fraction of all the edges at each step, so that their reintroduction increases the width by a factor of at most two. We then run the compression algorithm on the obtained 2-approximate ordering to get an optimum one. Since we remove a large portion of the graph at each step, the recursive equation on the running time solves to a linear function, instead of quadratic. This gives the following.

▶ **Theorem 2.** *There exists an algorithm that, given an n-vertex graph G and an integer k, runs in time $2^{O(k^2 \log k)} \cdot n$ and either correctly concludes that the cutwidth of G is larger than k, or outputs an ordering of G of width at most k.*

The algorithm of Theorem 2 has running time slightly larger than that of Thilikos, Bodlaender, and Serna [17, 18]. The difference is the $\log k$ factor in the exponent, the reason for which is that we use a simpler bucketing approach to bound the number of states, instead of the more entangled, but finer, machinery of typical sequences. We believe the main strength of our approach lies in its explanatory character. Instead of relying on algorithms for computing tree or path decompositions, which are already difficult by themselves, and then designing a dynamic programming algorithm on a path decomposition, we directly approach cutwidth "via cutwidth", and not "via pathwidth". That is, the dynamic programming procedure for computing the optimum cutwidth ordering on an approximate cutwidth ordering is technically far simpler and conceptually more insightful than performing the same on a general path decomposition. We also show that the reduction-by-a-large-fraction trick of Bodlaender [2] can be performed also in the cutwidth setting, yielding a self-contained, natural, and understandable algorithm.

2 Preliminaries

We denote the set of non-negative integers by $\mathbb{N}$ and the set of positive integers by $\mathbb{N}^+$. For $r, s \in \mathbb{N}$ with $r \leq s$, we denote $[r] = \{1, \ldots, r\}$ and $[r, s] = \{r, \ldots, s\}$. Notice that $[0] = \emptyset$.

Graphs. All graphs considered in this paper are undirected, without loops, and may have multiple edges. The vertex and edge sets of a graph G are denoted by $V(G)$ and $E(G)$, respectively. For disjoint $X, Y \subseteq V(G)$, by $E_G(X, Y)$ we denote the set of edges of G with one endpoint in X and one in Y. If $S \subseteq V(G)$, then we denote $\delta_G(S) = |E_G(S, V(G) \setminus S)|$. We drop the subscript if it is clear from the context. Every partition (A, B) of $V(G)$ is called a *cut of* G; the *size* of the cut (A, B) is $\delta(A)$.

Cutwidth. Let G be a graph and σ be an ordering of $V(G)$. For $u, v \in V(G)$, we write $u <_\sigma v$ if u appears before v in σ. Given two disjoint sequences $\sigma_1 = \langle x_1, \ldots, x_{r_1} \rangle$ and $\sigma_2 = \langle y_1, \ldots, y_{r_2} \rangle$ of vertices in $V(G)$, we define their *concatenation* as $\sigma_1 \circ \sigma_2 = \langle x_1, \ldots, x_{r_1}, y_1, \ldots, y_{r_2} \rangle$. For $X \subseteq V(G)$, let σ_X be the ordering of X induced by σ, i.e., the ordering obtained from σ if we remove the vertices that do not belong in X. For a vertex v we denote by V_v^σ the set $\{u \in V(G) \mid u \leq_\sigma v\}$. A *$\sigma$-cut* is any cut of the form $(V_v^\sigma, V(G) \setminus V_v^\sigma)$ for $v \in V(G)$. The *cutwidth of an ordering σ of G* is defined as $\mathbf{cw}_\sigma(G) = \max_{v \in V(G)} \delta(V_v^\sigma)$. The *cutwidth of G*, $\mathbf{cw}(G)$, is the minimum of $\mathbf{cw}_\sigma(G)$ over all possible orderings of $V(G)$.

Obstructions. Let $\leq$ be a partial order on graphs. We say that $G' \lneqq G$ if $G' \leq G$ and G' is not isomorphic to G. A graph class $\mathcal{G}$ is *closed under* $\leq$ if whenever $G' \leq G$ and $G \in \mathcal{G}$, we also have that $G' \in \mathcal{G}$. Given a partial order $\leq$ and a graph class $\mathcal{G}$ closed under $\leq$, we define the *(minimal) obstruction set* of $\mathcal{G}$ w.r.t. $\leq$, denoted by $\mathbf{obs}_\leq(\mathcal{G})$, as the set containing all graphs where the following two conditions hold: **O1**: $G \notin \mathcal{G}$, i.e., G is not a member of $\mathcal{G}$, and **O2**: for each G' with $G' \lneqq G$, we have that $G' \in \mathcal{G}$.

We say that a set of graphs $\mathcal{H}$ is a *$\leq$-antichain* if it does not contain any pair of comparable elements wrt. $\leq$. By definition, for any class $\mathcal{G}$ closed under $\leq$, the set $\mathbf{obs}_\leq(\mathcal{G})$ is an antichain.

Immersions. Let H and G be graphs. We say that G contains H as an *immersion* if there is a pair of functions (ϕ, ψ), called an *H-immersion model of G*, such that ϕ is an injection from $V(H)$ to $V(G)$ and ψ maps every edge uv of H to a path of G between $\phi(u)$ and $\phi(v)$ so that different edges are mapped to edge-disjoint paths. Every vertex in the image of ϕ is called a *branch vertex*. If we additionally demand that no internal vertex of a path in $\psi(E(H))$ is a branch vertex, then we say that (ϕ, ψ) is a *strong H-immersion model* and H is a *strong immersion* of G. We denote by $H \leq_{\mathrm{i}} G$ ($H \leq_{\mathrm{si}} G$) the fact that H is an immersion (strong immersion) of G; these are partial orders. Clearly, for any two graphs H and G, if $H \leq_{\mathrm{si}} G$ then $H \leq_{\mathrm{i}} G$. This implies the following observation:

▶ **Observation 3.** *If $\mathcal{G}$ is a graph class closed under $\leq_{\mathrm{i}}$, then $\mathbf{obs}_{\leq_{\mathrm{i}}}(\mathcal{G}) \subseteq \mathbf{obs}_{\leq_{\mathrm{si}}}(\mathcal{G})$.*

Robertson and Seymour proved in [15] that every $\leq_i$-antichain is finite and conjectured the same for $\leq_{\mathrm{si}}$. It is well-known that for every $k \in \mathbb{N}$, the class $\mathcal{C}_k$ of graphs of cutwidth at most k is closed under immersions. It follows from the results of [15] that $\mathbf{obs}_{\leq_{\mathrm{i}}}(\mathcal{C}_k)$ is finite; the goal of this paper is to provide good estimates on the sizes of graphs in $\mathbf{obs}_{\leq_{\mathrm{si}}}(\mathcal{C}_k)$. As the cutwidth of a graphs is the maximum cutwidth of its connected components, it follows that graphs in $\mathbf{obs}_{\leq_{\mathrm{si}}}(\mathcal{C}_k)$ are connected. Moreover, every graph in $\mathbf{obs}_{\leq_{\mathrm{si}}}(\mathcal{C}_k)$ has cutwidth exactly $k+1$, because the removal of any of its edges decreases its cutwidth to at most k.

3 Bucket interfaces

Let G be a graph and σ be an ordering of $V(G)$. For a set $X \subseteq V(G)$, the *X-blocks* in σ are the maximal subsequences of consecutive vertices of σ that belong to X. Suppose (A, B) is a cut of G. Then we can write $\sigma = b_1 \circ \ldots \circ b_p$, where $b_1, \ldots, b_p$ are the A- and B-blocks in σ; these will be called jointly *(A, B)-blocks*. The next lemma is the cornerstone of our approach: we prove that given a graph G and a cut (A, B) of G, there exists an optimum cutwidth ordering of G where number of blocks depends only on the cutwidth and the size of (A, B).

▶ **Lemma 4.** *Let $\ell \in \mathbb{N}^+$ and G be a graph. If (A, B) is a cut of G of size ℓ, then there is an optimum cutwidth ordering σ of $V(G)$ with at most $(2\ell+1) \cdot (2\mathbf{cw}(G)+3) + 2\ell$ (A, B)-blocks.*

Proof. Let σ be an optimum cutwidth ordering such that, subject to the width being minimum, the number of (A, B)-blocks it defines is also minimized. Let $\sigma = b_1 \circ b_2 \circ \cdots \circ b_r$, where $b_1, b_2, \ldots, b_r$ are the (A, B)-blocks of σ. If σ defines less than three blocks, then the claim already follows, so let us assume $r \geq 3$.

Consider any ordering σ' obtained by swapping two blocks, i.e., $\sigma' = b_1 \circ \cdots \circ b_{j-1} \circ b_{j+1} \circ b_j \circ b_{j+2} \ldots b_r$, for some $j \in [r-1]$. Observe that since the blocks $b_1, \ldots, b_r$ alternate as A-blocks and B-blocks, the ordering σ' has a strictly smaller number of blocks; indeed, either $j - 1 \geq 1$, in which case $b_{j-1} \circ b_{j+1}$ defines a single block of σ', or $j = 1$ and hence $j + 2 \leq r$, in which case $b_j \circ b_{j+2}$ does. Therefore, by choice of σ, for each $j \in [r-1]$, swapping b_j and b_{j+1} in σ must yield an ordering with strictly larger cutwidth.

We call a block *free* if it does not contain any endpoint of the cut edges $E_G(A, B)$. We now prove that any sequence of consecutive free blocks in σ has at most $2\mathbf{cw}(G) + 3$ blocks. Since the cut (A, B) has size ℓ, there are at most 2ℓ blocks that are not free. This implies the claimed bound on the total number of all blocks in σ.

Suppose, to the contrary, that there exists a sequence of $q > 2\mathbf{cw}(G) + 3$ consecutive free blocks in σ. Let these blocks be $b_r, b_{r+1}, \ldots, b_s$, where $s - r + 1 = q$. For $j \in [r, s-1]$, we define $\mu(j)$ to be the size of the cut between all vertices inside or preceding the vertices of block b_j and all vertices inside or following the vertices of block b_{j+1} in σ; see Figure 1.

▶ **Claim 5.** *For all $j \in [r+1, \ldots, s-2]$, we have that $\mu(j-1) > \mu(j)$ or $\mu(j) < \mu(j+1)$.*

Proof. Suppose that for some $j \in [r+1, s-2]$, $\mu(j) \geq \max(\mu(j-1), \mu(j+1))$. We will then show that the ordering σ' obtained by swapping the blocks b_j and b_{j+1} still has optimum cutwidth, a contradiction to the choice of σ. Notice that for every vertex v preceding all vertices of b_j or succeeding all vertices of b_{j+1}, $\delta(V_v^{\sigma'}) = \delta(V_v^{\sigma})$. Thus, it remains to show that for any vertex v belonging to the block b_j or to the block b_{j+1}, also $\delta(V_v^{\sigma'}) \leq \delta(V_v^{\sigma})$.

Let p_j be the number of edges of G with one endpoint in the block b_j and the other endpoint preceding (in σ) all vertices of b_j. Let also s_j be the number of edges of G with one endpoint in b_j and the other endpoint succeeding (in σ) all vertices of b_j (and hence succeeding all vertices of block b_{j+1}, since both b_j and b_{j+1} are free). Notice that $\mu(j) = \mu(j-1) - p_j + s_j$ and recall that $\mu(j) \geq \mu(j-1)$. This yields that $s_j \geq p_j$.

Similarly, let p_{j+1} be the number of edges of G with one endpoint in b_{j+1} and the other endpoint preceding all vertices of the block b_{j+1} (and, in particular, all vertices of block b_j). Let also s_{j+1} be the number of edges of G with one endpoint in b_{j+1} and the other endpoint succeeding all vertices of block b_{j+1}. Again, we have $\mu(j+1) = \mu(j) - p_{j+1} + s_{j+1}$ and $\mu(j) \geq \mu(j+1)$. This yields that $p_{j+1} \geq s_{j+1}$.

Let v be a vertex of the block b_j. Recall that the blocks b_j and b_{j+1} are free and thus, there are no edges between them. Observe then that $\delta(V_v^{\sigma'}) = \delta(V_v^{\sigma}) + s_{j+1} - p_{j+1} \leq \delta(V_v^{\sigma})$. Symmetrically, for any vertex v in b_{j+1}, observe that $\delta(V_v^{\sigma'}) = \delta(V_v^{\sigma}) + p_j - s_j \leq \delta(V_v^{\sigma})$. Thus, $\mathbf{cw}_{\sigma'}(G) \leq \mathbf{cw}_{\sigma}(G) = \mathbf{cw}(G)$, a contradiction. ◀

Claim 5 shows that for all $j \in [r+1, s-2]$, we have $\mu(j-1) > \mu(j)$ or $\mu(j) < \mu(j+1)$. It follows that any non-decreasing pair $\mu(j-1) \leq \mu(j)$ must be followed by an increasing pair $\mu(j) < \mu(j+1)$. Hence, if $j_{\min}$ is the minimum index such that $\mu(j_{\min}) \leq \mu(j_{\min}+1)$, then the sequence $\mu(j)$ has to be strictly decreasing up to $j_{\min}$ and strictly increasing from $j_{\min}+1$ onward. Since $\mu(j) \leq \mathbf{cw}(G)$ for all j, the length q of the sequence of consecutive free blocks cannot be longer than $2\mathbf{cw}(G) + 3$ in total, concluding the proof. ◀

We use the above lemma to bound the number of "types" of prefixes in graph orderings. To describe such a prefix, i.e., one side of a cut in a graph, we use the following definition.

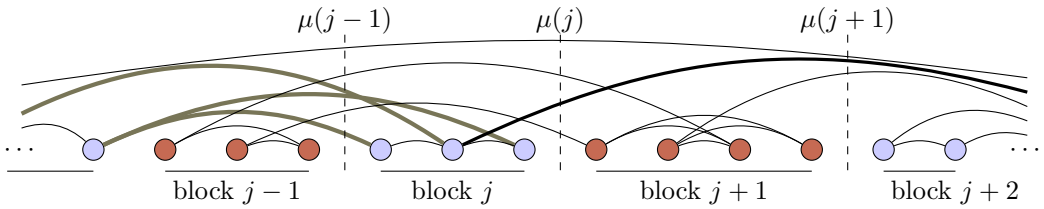


Figure 1 A cut (A, B) is highlighted (blue, red), with the corresponding blocks underlined and cuts between them marked with dashed lines. Edges counted as p_j and s_j are thickened.

▶ **Definition 6.** A *k-boundaried graph* is a pair $\mathbf{G} = (G, \bar{x})$ where G is a graph and $\bar{x} = (x_1, \ldots, x_k)$ is a k-tuple of the graph's *boundary vertices* (ordered, not necessarily distinct). The *extension* of $\mathbf{G}$ is the graph G^* obtained from G by adding k new vertices $x'_1, \ldots, x'_k$ and edges $x_1x'_1, \ldots, x_kx'_k$. The *join* $\mathbf{A} \oplus \mathbf{B}$ of two k-boundaried graphs $\mathbf{A} = (A, \bar{x}), \mathbf{B} = (B, \bar{y})$ is the graph obtained from the disjoint union of A and B by adding an edge x_iy_i for $i \in [k]$.

From Lemma 4 we derive that for any given cut (A, B) of size ℓ of a graph G with $\mathbf{cw}(G) \leq k$, there is an optimum cutwidth ordering in which the vertices of A occur in $O(k\ell)$ blocks. Our next goal is to show that the only information about A that can affect the cutwidth of G is: the placing of the endpoints of each cutedge (x_i and x'_i) into blocks, and the cutwidth of each block (as an induced subgraph of A or A^*). Recall that for an ordering σ of $V(G)$, *σ-cuts* are cuts of the form $(V^\sigma_v, V(G) \setminus V^\sigma_v)$, for $v \in V(G)$.

▶ **Definition 7.** Let G be a graph and σ be an ordering of its vertices. An *ℓ-bucketing* of σ is a function $T\colon V(G) \to [\ell]$ such that $T(u) \leq T(v)$ for any u appearing before v in σ. For every $i \in [\ell]$, the set $T^{-1}(i)$ will be called a *bucket*; a bucket is naturally ordered by σ. For every bucket $T^{-1}(i)$, $i \in [\ell]$, let $\mathtt{cuts}(G, \sigma, T, i)$ be the family of σ-cuts containing on one side all vertices of buckets appearing before i and a prefix (in σ) of the i-th bucket. For an ordering σ of the vertices of a graph G, define the *width* of the bucket i, $i \in [\ell]$, as the maximum width of any cut in the family $\mathtt{cuts}(G, \sigma, T, i)$. Formally,

$$\begin{aligned}
\mathtt{cuts}(G, \sigma, T, i) &= \big\{\big(T^{-1}([1, i-1]) \cup L, \quad R \cup T^{-1}([i+1, \ell])\big) : \\
&\qquad (L, R) \text{ is a } \sigma\text{-cut of } T^{-1}(i)\big\}, \\
\mathtt{width}(G, \sigma, T, i) &= \max\{\, |E_G(L, R)| \;:\; (L, R) \in \mathtt{cuts}(G, \sigma, T, i)\,\}.
\end{aligned}$$

Notice that every σ-cut of G is in $\mathtt{cuts}(G, \sigma, T, i)$ for at least one bucket $i \in [\ell]$; since $\mathbf{cw}_\sigma(G)$ is the maximum of $|E_G(L, R)|$ over σ-cuts (L, R), we have

$$\mathbf{cw}_\sigma(G) = \max_{i \in [\ell]} \mathtt{width}(G, \sigma, T, i). \tag{1}$$

For two k-boundaried graphs $\mathbf{A} = (A, \bar{x}), \mathbf{B} = (B, \bar{y})$, we slightly abuse notation and understand the edges $x_1x'_1, \ldots, x_kx'_k$ in A^* to be the same as $y'_1y_1, \ldots, y'_ky_k$ in B^* and as $x_1y_1, \ldots, x_ky_k$ in $\mathbf{A} \oplus \mathbf{B}$. That is, for an ordering σ of $\mathbf{A} \oplus \mathbf{B}$ with ℓ-bucketing T, we define $T|_{A^*}(v)$ to be $T(v)$ for $v \in V(A)$ and $T(y_i)$ for $v = x'_i$. We define $\sigma|_{A^*}$ as an ordering that orders x'_i just as σ orders y_i, with the order between x'_i and x'_j chosen arbitrarily when $y_i = y_j$. The following lemma shows that if an ℓ-bucketing respects the sides of a cut, then the width of any bucket can be computed as the sum of contributions of the sides.

▶ **Lemma 8 (♠[3]).** *Let k, ℓ be positive integers and $\mathbf{A} = (A, \bar{x}), \mathbf{B} = (B, \bar{y})$ be two k-boundaried graphs. Let also σ be a vertex ordering of $\mathbf{A} \oplus \mathbf{B}$ with ℓ-bucketing T. If $T^{-1}(i)$ does not contain any vertex of A, for some $i \in [\ell]$, that is, $T^{-1}(i) \cap V(A) = \emptyset$, then it holds that* $\texttt{width}(\mathbf{A} \oplus \mathbf{B}, \sigma, T, i) = \texttt{width}(A, \sigma|_A, T|_A, i) + \texttt{width}(B^*, \sigma|_{B^*}, T|_{B^*}, i)$.

Replacing the roles of $\mathbf{A}$ and $\mathbf{B}$ above, we obtain that if $T^{-1}(i)$ does not contain any vertex of B, then $\texttt{width}(\mathbf{A} \oplus \mathbf{B}, \sigma, T, i) = \texttt{width}(A^*, \sigma|_{A^*}, T|_{A^*}, i) + \texttt{width}(B, \sigma|_B, T|_B, i)$. Intuitively, this implies that the cutwidth of $\mathbf{A} \oplus \mathbf{B}$ depends on A only in the widths of each block relative to A and A^* (in any bucketing where buckets are either A-blocks or B-blocks). Therefore, replacing $\mathbf{A}$ with another boundaried graph whose extension has an ordering and bucketing with the same widths preserves cutwidth (as long as endpoints of the cut edges are placed in the same buckets too). This is formalized in the next definition.

▶ **Definition 9.** For $k, \ell \in \mathbb{N}$, a *(k,ℓ)-bucket interface* consists of functions:
- $b, b' : [k] \to [\ell]$ identifying the buckets which contain x_i and x'_i, respectively and
- $\mu, \mu^* : [\ell] \to [0, k]$ corresponding to the widths of buckets.

A k-boundaried graph $\mathbf{G}$ *conforms* with a (k, ℓ)-bucket interface if there exists an ordering σ of the vertices of G^* and an ℓ-bucketing T of G^* such that:
- $T(v)$ is odd for $v \in V(G)$ and even for $v \in \{x'_1, \ldots, x'_k\}$,
- $T(x_i) = b(i)$ and $T(x'_i) = b'(i)$, for each $i \in [k]$,
- $\texttt{width}(G, \sigma|_G, T|_G, j) \leq \mu(j)$, for each $j \in [\ell]$,
- $\texttt{width}(G^*, \sigma, T, j) \leq \mu^*(j)$, for each $j \in [\ell]$.

▶ **Observation 10.** *For all $k, \ell \in \mathbb{N}^+$ there are $\leq 2^{2(k \log \ell + \ell \log(k+1))}$ (k, ℓ)-bucket interfaces.*

We call two k-boundaried graphs $\mathbf{G}_1, \mathbf{G}_2$ *(k,ℓ)-similar* if the sets of (k, ℓ)-bucket interfaces they conform with are equal. The following lemma subsumes the above ideas. The proof follows easily from Lemma 8 and the fact that $\mathbf{cw}_\sigma(G) = \max_{i \in [\ell]} \texttt{width}(G, \sigma, T, i)$ (Eq. (1)).

▶ **Theorem 11 (♠).** *Let k, r be two positive integers. Let also $\mathbf{A}_1$ and $\mathbf{A}_2$ be two k-boundaried graphs that are (k, ℓ)-similar, where $\ell = (2k+1) \cdot (2r+4)$. Then for any k-boundaried graph $\mathbf{B}$ where $\mathbf{cw}(\mathbf{A}_1 \oplus \mathbf{B}) \leq r$, it holds that $\mathbf{cw}(\mathbf{A}_2 \oplus \mathbf{B}) = \mathbf{cw}(\mathbf{A}_1 \oplus \mathbf{B})$.*

4 Obstruction sizes and linked orderings

In this section we establish the main result on sizes of obstructions for cutwidth. We first define linked orderings and prove that there is always an optimum ordering that is linked.

▶ **Definition 12** (linked ordering)**.** An ordering σ of $V(G)$ is *linked* if for any two vertices $u \leq_\sigma u'$, there are $\min\{\delta(V_v^\sigma) \mid u \leq_\sigma v \leq_\sigma u'\}$ edge-disjoint paths between V_u^σ and $V(G) \setminus V_{u'}^\sigma$ in G.

▶ **Lemma 13** ([8, 12])**.** *For every graph G, there is a linked ordering σ of $V(G)$ with $\mathbf{cw}_\sigma(G) = \mathbf{cw}(G)$.*

Proof. Without loss of generality, we may assume that the graph is connected. Let σ be an optimum cutwidth ordering of $V = V(G)$. Subject to the optimality of σ, we choose σ so that $\sum_{v \in V} \delta(V_v^\sigma)$ is minimized. We prove that σ defined in this manner is in fact linked.

Assume the contrary. Then by Menger's theorem, there exist vertices $u <_\sigma u'$ in V and $i \in \mathbb{N}$ such that $\delta(V_v^\sigma) > i$ for every $u \leq_\sigma v \leq_\sigma u'$, but a minimum cut (A, B) of G with

[3] Proofs of statements marked with ♠ are ommited from this extended abstract. The full version of the paper is available in http://arxiv.org/abs/1606.05975.

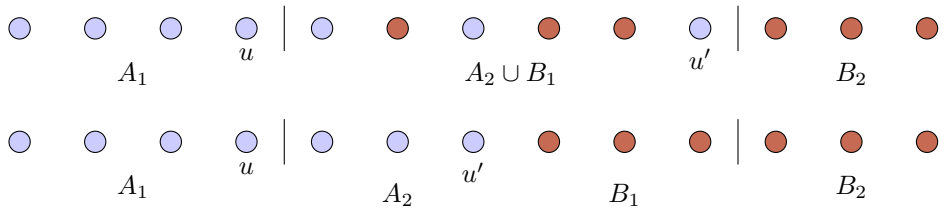


Figure 2 An ordering of vertices with the minimum cut (A, B) between A_1 and B_2 of size i highlighted in blue and red. Below, the modified ordering, with cutwidth bounded using submodularity.

$V_u^\sigma \subseteq A$ and $V \setminus V_{u'}^\sigma \subseteq B$ has size $\delta(A) \leq i$. We partition A into sets A_1 and A_2, where $A_1 = V_u^\sigma$ and $A_2 = A \setminus A_1$, and we partition B into sets B_1 and B_2, where $B_2 = V \setminus V_{u'}^\sigma$ and $B_1 = B \setminus B_2$ (see Figure 2). Notice that $A_2 = A \setminus V_u^\sigma = \{v \mid u <_\sigma v \leq_\sigma u'\} \cap A$ and that $B_1 = B \setminus (V \setminus V_{u'}^\sigma) = \{v \mid u <_\sigma v \leq_\sigma u'\} \cap B$. Let σ' be the ordering of V obtained by concatenating $\sigma|_{A_1}$, $\sigma|_{A_2}$, $\sigma|_{B_1}$, and $\sigma|_{B_2}$.

We prove that $\delta(V_v^{\sigma'}) \leq \delta(V_v^\sigma)$, for every $v \in V$. Observe first that for every vertex $v \in A_1 \cup B_2$ it holds that $V_v^{\sigma'} = V_v^\sigma$ and thus, $\delta(V_v^{\sigma'}) = \delta(V_v^\sigma)$. Let now $v \in A_2$. Then $V_v^{\sigma'} = V_v^\sigma \cap A$. By the submodularity of cuts it follows that $\delta(V_v^\sigma \cup A) + \delta(V_v^\sigma \cap A) \leq \delta(A) + \delta(V_v^\sigma)$. Notice that $(V_v^\sigma \cup A, V \setminus (V_v^\sigma \cup A))$ is also a cut separating $A_1 = V_u^\sigma$ and $B_2 = V \setminus V_{u'}^\sigma$. From the minimality of (A, B) it follows that $\delta(A) \leq \delta(V_v^\sigma \cup A)$. Therefore, $\delta(V_v^\sigma \cap A) \leq \delta(V_v^\sigma)$. As $V_v^{\sigma'} = V_v^\sigma \cap A$, we obtain that $\delta(V_v^{\sigma'}) \leq \delta(V_v^\sigma)$.

Symmetrically, let now $v \in B_1$. Then $V_v^{\sigma'} = V_v^\sigma \cup A$. By the submodularity of cuts we have $\delta(V_v^\sigma \cup A) + \delta(V_v^\sigma \cap A) \leq \delta(A) + \delta(V_v^\sigma)$. Notice that $(V_v^\sigma \cap A, V \setminus (V_v^\sigma \cap A))$ is a cut separating A_1 and B_2. From the minimality of (A, B) it follows that $\delta(A) \leq \delta(V_v^\sigma \cap A)$. Therefore, $\delta(V_v^\sigma \cup A) \leq \delta(V_v^\sigma)$. As $V_v^{\sigma'} = V_v^\sigma \cup A$, we obtain that $\delta(V_v^{\sigma'}) \leq \delta(V_v^\sigma)$.

Thus, $\delta(V_v^{\sigma'}) \leq \delta(V_v^\sigma) \leq \mathbf{cw}(G)$ for every $v \in V$, and hence $\mathbf{cw}_{\sigma'}(G) = \mathbf{cw}(G)$. Finally, note that $\delta(V_v^{\sigma'}) = \delta(A) \leq i < \delta(V_v^\sigma)$ for the last vertex v in A. Thus $\sum_v \delta(V_v^{\sigma'}) < \sum_v \delta(V_v^\sigma)$, contradicting the choice of σ. Therefore, σ is a linked ordering of V with $\mathbf{cw}_\sigma(G) = \mathbf{cw}(G)$. ◀

The following theorem is the technical counterpart of Theorem 1. Its proof is based on Theorem 11, Lemma 13, Observation 10 and the idea of "unpumping" repeating types, presented in the introduction. The linkedness is used to make sure that within the unpumped segment of the ordering, one can find the maximum possible number of edge-disjoint paths between the parts of the graph on the left side and on the right side of the segment. This ensures that the graph obtained from unpumping can be immersed in the original one.

▶ **Theorem 14 (♠).** *Let k be a positive integer. If $G \in \mathbf{obs}_{\leq_{\mathrm{si}}}(\mathcal{C}_k)$, then $|V(G)| \leq N^{k+1}$, where $N = 2^{2((k+1)\log \ell + \ell \log(k+2))} + 2$ and $\ell = (2k+3) \cdot (2k+6)$.*

Theorem 14 provides an upper bound on the number of vertices of a graph in $\mathbf{obs}_{\leq_{si}}(\mathcal{C}_k)$. Observe that since such a graph has cutwidth $k+1$, each of its vertices has degree at most $2(k+1)$. It follows that any graph from $\mathbf{obs}_{\leq_{si}}(\mathcal{C}_k)$ has $2^{O(k^3 \log k)}$ vertices and edges. Finally, by Observation 3 we have $\mathbf{obs}_{\leq_i}(\mathcal{C}_q) \subseteq \mathbf{obs}_{\leq_{si}}(\mathcal{C}_q)$, so the same bound holds also for immersions instead of strong immersions. This concludes the proof of Theorem 1.

5 An algorithm for computing cutwidth

In this section we present an exact FPT algorithm for computing the cutwidth of the graph. First, we need to give a dynamic programming algorithm that given an approximate ordering σ of width r, finds, if possible, an ordering of width at most k, where $k \leq r$ is given.

▶ **Lemma 15 (♠).** *Let $r \in \mathbb{N}^+$. Given a graph G and an ordering σ of its vertices with $\mathbf{cw}_\sigma(G) \leq r$, an ordering τ of the vertices of G with $\mathbf{cw}_\tau(G) = \mathbf{cw}(G)$ can be computed in time $2^{O(r^2 \log r)} \cdot |V(G)|$.*

The main ingredient in the proof of Lemma 15 is the insight given by Lemma 4: any set X with $\delta(X) \leq r$, in particular any prefix of σ, can be assumed to be split into at most $O(kr)$ blocks in some optimum ordering τ. A closer look into the proof of Lemma 4 shows that a stronger statement is true: there is some optimum ordering τ such that for *every* prefix X of σ, there are at most $O(kr)$ X-blocks in σ. This shows that an optimum ordering τ can be constructed by a dynamic programming procedure that scans σ from left to right while constructing an optimum ordering, maintained as a mapping of the already scanned vertices into at most ℓ blocks. The description of such a partial ordering is an enrichment of a (k, ℓ)-bucket interface that we call a (k, ℓ)-*bucket profile*. In such a profile, we additionally store some information about the sizes of cuts which is needed to make sure that the constructed ordering has width at most k. With this understanding, the construction of the dynamic programming algorithm is a routine, though technical task.

Having the algorithm of Lemma 15, a standard application of the iterative compression technique immediately yields a $2^{O(k^2 \log k)} \cdot n^2$ time algorithm for computing cutwidth, as sketched in Section 1. Simply add the vertices of G one by one, and apply the algorithm of Lemma 15 at each step. However, we can make the dependence on n linear by adapting the approach of Bodlaender [2]; more precisely, we make bigger steps. Such a big step consists of finding a graph H that can be immersed in the input graph G, which is smaller by a constant fraction, but whose cutwidth is not much smaller. This is formalized in the next lemma.

▶ **Lemma 16 (♠).** *There is an algorithm that given a positive integer k and a graph G, works in time $O(k^2 \cdot |V(G)|)$ and either concludes that $\mathbf{cw}(G) > k$, or finds a graph H immersed in G such that $|E(H)| \leq |E(G)| \cdot (1 - 1/(2k+1)^{4(k+1)+3})$ and $\mathbf{cw}(G) \leq 2\mathbf{cw}(H)$. Furthermore, in the latter case, given an ordering σ of the vertices of H, an ordering τ of the vertices of G with $\mathbf{cw}_\tau(G) \leq 2\mathbf{cw}_\sigma(H)$ can be computed in $O(|V(G)|)$ time.*

The proof starts by iteratively dissolving vertices of degree two with both incident edges going to different neighbors; this operation preserves the cutwidth of the graph. Then, if a constant fraction (depending on k) of vertices has degree equal to one, we can find a large matching of edges incident to degree-1 vertices. If we remove them, the size of the graph drops by a constant fraction, but reintroducing them increases the cutwidth by at most one.

Otherwise, we arrive at the situation when there are no vertices of degree 2, apart from the ones with only one neighbor, and there is only a very small fraction of vertices of degree one. Our goal now is to find a large (constant fraction of $|E(G)|$, where the constant depends on k) packing of edge-disjoint cycles in G. If we succeed in this, then an easy charging argument shows that removing one edge from each of these cycles yields a graph H with $\mathbf{cw}(H) \geq \mathbf{cw}(G)/2$, whereas the size of H is smaller than G by a constant fraction. Now, if there are many vertices of degree 2 with both incident edges going to the same neighbor, then we have a large packing of 2-cycles and we are done. So we can assume that the total number of vertices of degree 1 and 2 is only a small fraction of the total size of the graph.

At this point, we greedily find a large family of disjoint balls of radius $2(k+1)$ (sets of vertices at distance at most $2(k+1)$ from some central vertex), each containing no vertex of degree 1 or 2. If any of these balls induced a tree in G, it would contain a full binary tree of height $2(k+1)$; but such a binary tree is known to have cutwidth larger than k by itself, allowing us to conclude that $\mathbf{cw}(G) > k$. Finally, if every ball contains a cycle, then we have found a large packing of vertex-disjoint, hence also edge-disjoint cycles.

We are now ready to put all the pieces together and prove Theorem 2: given an n-vertex graph G and an integer k, one can in time $2^{O(k^2 \log k)} \cdot n$ either conclude that $\mathbf{cw}(G) > k$, or output an ordering of G of width at most k. The proof follows the same recursive Reduction&Compression scheme as the algorithm of Bodlaender [2]. By applying Lemma 16, we obtain a significantly smaller immersion H, and we recurse on H. This recursive call either concludes that $\mathbf{cw}(H) > k$, which implies $\mathbf{cw}(G) > k$, or it produces an ordering of H of optimum width $\mathbf{cw}(H) \leq k$. This ordering can be lifted, using Lemma 16 again, to an ordering of G of width $\leq 2k$. Given this ordering, we apply the dynamic programming procedure of Lemma 15 to construct an optimum ordering of G in time $2^{O(k^2 \log k)} \cdot |V(G)|$.

Since at each recursion step the number of edges of the graph drops by a multiplicative factor of at least $1/(2k+1)^{4(k+1)+3}$, we see that the graph G_i at level i of the recursion will have at most $(1 - 1/(2k+1)^{4(k+1)+3})^i \cdot |E(G)|$ edges. Hence, the total work used by the algorithm is bounded by the sum of a geometric series:

$$\begin{aligned}\sum_{i=0}^{\infty} 2^{O(k^2 \log k)} \cdot |E(G_i)| &\leq 2^{O(k^2 \log k)} \cdot |E(G)| \cdot \sum_{i=0}^{\infty} (1 - 1/(2k+1)^{4k+7})^i \\ &= 2^{O(k^2 \log k)} \cdot |E(G)| \cdot (2k+1)^{4k+7} = 2^{O(k^2 \log k)} \cdot |E(G)|.\end{aligned}$$

6 Conclusions

In this paper we have proved that the immersion obstructions for admitting a layout of cutwidth at most k have sizes single-exponential in $O(k^3 \log k)$. The core of the proof can be interpreted as bounding the number of different behavior types for a part of the graph that has only a small number of edges connecting it to the rest. This, in turn, gives an upper bound on the number of states for a dynamic programming algorithm that computes the optimum cutwidth ordering on an approximate one. This last result, complemented with an adaptation of the reduction scheme of Bodlaender [2] to the setting of cutwidth, yields a direct and self-contained FPT algorithm for computing the cutwidth of a graph. In fact, we believe that our algorithm can be thought of "Bodlaender's algorithm for treewidth in a nutshell". It consists of the same two components, namely a recursive reduction scheme and dynamic programming on an approximate decomposition, but the less challenging setting of cutwidth makes both components simpler, thus making the key ideas easier to understand. For an alternative attempt of simplification of the algorithm of Bodlaender and Kloks [3], applied for the case of pathwidth, see [6].

In our proof of the upper bound on the number of types/states, we used a somewhat new bucketing approach. This approach holds the essence of the typical sequences of Bodlaender and Kloks [3], but we find it more natural and conceptually simpler. The drawback is that we lose a $\log k$ factor in the exponent. It is conceivable that we could refine our results by removing this factor provided we applied typical sequences directly, but this is a price that we are willing to pay for the sake of simplicity and being self-contained.

An important ingredient of our approach is the observation that there is always an optimum cutwidth ordering that is linked: the cutsizes along the ordering precisely govern the edge connectivity between prefixes and suffixes. Recently, there is a growing interest in parameters that are tree-like analogues of cutwidth: tree-cut width [20] and carving-width [16]. In future work, we aim to explore and use linkedness for tree-cut decompositions and carving decompositions in a similar manner as presented here.

Acknowledgements. The second author thanks Mikołaj Bojańczyk for the common work on understanding and reinterpreting the Bodlaender-Kloks dynamic programming algorithm [3], which influenced the bucketing approach presented in this paper. We also thank O-joung Kwon for pointing us to [8, 12], as well as an anonymous referee for noting that the running time in Lemma 16 can be reduced to polynomial by amortization.

References

1 Patrick Bellenbaum and Reinhard Diestel. Two short proofs concerning tree-decompositions. *Combinatorics, Probability & Computing*, 11(6):541–547, 2002.

2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

3 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

4 Heather Booth, Rajeev Govindan, Michael A. Langston, and Siddharthan Ramachandramurthi. Cutwidth approximation in linear time. In *Proceedings of the Second Great Lakes Symposium on VLSI*, pages 70–73. IEEE, 1992.

5 Josep Díaz, Jordi Petit, and Maria J. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.

6 Martin Fürer. Faster computation of path-width. In Veli Mäkinen, J. Simon Puglisi, and Leena Salmela, editors, *Combinatorial Algorithms: 27th International Workshop, IWOCA 2016, Helsinki, Finland, August 17-19, 2016, Proceedings*, pages 385–396, Cham, 2016. Springer International Publishing.

7 Michael R. Garey and David S. Johnson. *Computers and intractability*, volume 174. Freeman New York, 1979.

8 James F. Geelen, A. M. H. Gerards, and Geoff Whittle. Branch-width and well-quasi-ordering in matroids and graphs. *J. Comb. Theory, Ser. B*, 84(2):270–290, 2002. A correction is available at `http://www.math.uwaterloo.ca/~jfgeelen/Publications/bn-corr.pdf`.

9 Rajeev Govindan and Siddharthan Ramachandramurthi. A weak immersion relation on graphs and its applications. *Discrete Mathematics*, 230(1):189–206, 2001.

10 Pinar Heggernes, Daniel Lokshtanov, Rodica Mihai, and Charis Papadopoulos. Cutwidth of split graphs and threshold graphs. *SIAM J. Discrete Math.*, 25(3):1418–1437, 2011.

11 Pinar Heggernes, Pim van 't Hof, Daniel Lokshtanov, and Jesper Nederlof. Computing the cutwidth of bipartite permutation graphs in linear time. *SIAM J. Discrete Math.*, 26(3):1008–1021, 2012.

12 Mamadou Moustapha Kanté and O-joung Kwon. An upper bound on the size of obstructions for bounded linear rank-width. *CoRR*, arXiv:1412.6201, 2014.

13 Jens Lagergren. Upper bounds on the size of obstructions and intertwines. *J. Comb. Theory, Ser. B*, 73(1):7–40, 1998.

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

15 Neil Robertson and Paul D. Seymour. Graph minors XXIII. Nash-Williams' immersion conjecture. *J. Comb. Theory, Ser. B*, 100(2):181–205, 2010.

16 Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

17 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. *J. Algorithms*, 56(1):1–24, 2005.

18 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Cutwidth II: Algorithms for partial w-trees of bounded degree. *J. Algorithms*, 56(1):25–49, 2005.

19 Robin Thomas. A Menger-like property of tree-width: The finite case. *J. Comb. Theory, Ser. B*, 48(1):67–76, 1990.

20 Paul Wollan. The structure of graphs not admitting a fixed immersion. *J. Comb. Theory, Ser. B*, 110:47–66, 2015.

21 Mihalis Yannakakis. A polynomial algorithm for the min-cut linear arrangement of trees. *J. ACM*, 32(4):950–988, 1985.

Computing Graph Distances Parameterized by Treewidth and Diameter*

Thore Husfeldt

Lund University and IT University of Copenhagen, Lund, Sweden
thore.husfeldt@cs.lth.se

Abstract

We show that the eccentricity of every vertex in an undirected graph on n vertices can be computed in time $n \exp O(k \log d)$, where k is the treewidth of the graph and d is the diameter. This means that the diameter and the radius of the graph can be computed in the same time. In particular, if the diameter is constant, it can be determined in time $n \exp O(k)$. This result matches a recent hardness result by Abboud, Vassilevska Williams, and Wang [SODA 2016] that shows that under the Strong Exponential Time Hypothesis of Impagliazzo, Paturi, and Zane [J. Comp. Syst. Sc., 2001], for any $\epsilon > 0$, no algorithm with running time $n^{2-\epsilon} \exp o(k)$ can distinguish between graphs with diameter 2 and 3.

Our algorithm is elementary and self-contained.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Graph algorithms, diameter, treewidth, Strong Exponential Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.16

1 Introduction

In an undirected graph, the *eccentricity* of a vertex is its largest distance to another vertex. The graph's *diameter*, denoted $\operatorname{diam} G$, is the largest eccentricity of any of its vertices. The graph's *radius*, denote $\operatorname{rad} G$, is the smallest eccentricity of any its vertices. The *treewidth* is a well-studied sparseness measure of graphs. These are fundamental parameters that permeate both the design of graph algorithms and the analysis of networks in many scientific domains.

We show the following result:

▶ **Theorem 1.** *Given an undirected n-vertex graph G with integer weights. If G has diameter $\operatorname{diam} G$ and treewidth k, then we can compute the eccentricity of every vertex, and compute $\operatorname{diam} G$ and $\operatorname{rad} G$, in time $n \exp O(k \log \operatorname{diam} G)$.*

For constant $\operatorname{diam} G$ this matches a recent lower bound by Abboud, Vassilevska Williams, and Wang [1] under the Strong Exponential Time Hypothesis of Impagliazzo, Paturi, and Zane [6]. In particular, it settles the complexity of the very simple question of deciding if a given undirected, unweighted graph has diameter 2 or 3.

1.1 Related work

It is easy to see that the diameter of an unweighted graph can be computed in time $O(nm)$ by computing the eccentricity of each node using breadth first search. For sparse graphs with

* This work was done at the Simons Institute for the Theory of Computation at UC Berkeley, partially supported by Grant 2012–4730 from the Swedish Research Council.

$m = O(n)$ this running time becomes $O(n^2)$, a bound that seems difficult to improve. Recent work by Roditty and Vassilevska Williams [8] has provided an illuminating explanation for this phenomenon: An algorithm for computing the diameter in time $O(m^{2-\epsilon})$ would violate the Strong Exponential Time Hypothesis.

However, it is also clear that for some *very* sparse graphs, this bound can be broken. In particularly, the diameter of a tree can be computed in linear time as follows. From any node find a remotest node u using breadth first search. Then find a remotest node v from u using breadth first search again. Elementary arguments show that $\operatorname{diam} G = \operatorname{dist}(u, v)$.

A useful parameter for studying this phenomenon is *treewidth*. The treewidth of a graph is a well-studied measure of its sparsity, properly defined in Section 2.4. In the extremes, every tree has treewidth 1 and the n-clique has treewidth $n - 1$.

In this framework, we can ask how the complexity of computing the diameter deteriorates with treewidth. For example, can the problem be solved in time $O(n \log n)$ on graphs of logarithmic treewidth? Very surprisingly, Abboud, Vassilevska Williams, and Wang [1] showed that not only can the complexity of the parameter be disentangled from the number of vertices in the sense of parameterised complexity, but this dependency is at least *exponential*: They show that under the Strong Exponential Time Hypothesis, it takes time

$$n^{2-\epsilon} \exp \Omega(k) \qquad \text{for any } \epsilon > 0 \tag{1}$$

to compute the diameter of an n-vertex graph with treewidth k. The same bound holds for computing the radius, but under a stronger hypothesis called the Hitting Set Conjecture. See [1] for a thorough presentation and discussion of these results and their underlying hypotheses and a rich overview of related work.

This result is surprising because the innocuous diameter problem exhibits a similar dependency on treewidth as several NP-hard problems. For instance, under the same Strong Exponential Time Hypothesis, it is known that the NP-hard Independent Set problem cannot be solved in time $(2 - \epsilon)^k \operatorname{poly} n$ for any $\epsilon > 0$ [7]. Of course, the lower bound (1) does not imply that the diameter problem is an exponential-time problem. This is because, unlike for Independent Set, the exponential dependency does not persist throughout all dependencies of k on n. In this case, it becomes vacuous for $k = \Omega(\log n)$, where the quadratic-time algorithm takes over.

The lower bound holds even for very restricted diameter problems, such as deciding if the diameter is 2 or 3, and (consequently) for approximating the solution. The same authors provide an algorithm for computing the diameter with running time

$$O(k^2 n \log^k n) = n^{1+o(1)} \exp O(k \log k)\,. \tag{2}$$

That algorithm follows a method introduced by Cabello and Knauer [3] for computing the Wiener index, based on a reduction to a k-dimensional orthogonal range query problem and its solution by Willard [9]. The authors explain how to extend this idea to provide algorithms for radius and eccentricities within the same time bound.

Closing the gap between (1) and (2) is viewed as a very interesting open problem [1].

Our contribution is to offer yet another parameter to this analysis, by introducing a dependency on the diameter to the running time. In particular, we match the lower bound of Abboud *et al.* for the regime of constant diameters: If $\operatorname{diam} G = O(1)$ then the complexity of these problems is (under various hypotheses) $n \exp \Theta(k)$.

Planar graphs of constant diameter have constant treewidth, so for that class of graphs the diameter can be found in linear time, which is a known result due to Eppstein [5].

Our algorithm is elementary, in particular compared to data structure results leveraged to establish (2). Instead of following Cabello and Knauer, we demonstrate that the necessary information can be built by traversing the tree decomposition a number of times in different directions. Once the dependency of the problem on the diameter has been realised, the construction is unsurprising. However, the argument is quite fragile and ultimately relies on a careful (but entirely combinatorial) analysis of shortest paths.

2 Algorithm

2.1 Preliminaries

A *walk* is an alternating sequence of vertices and edges, say $v_0, e_1, \ldots, e_l, v_r$, where $e_i = v_{i-1}v_i$ for $1 \leq i \leq l$. A walk with endvertices u and v is called a u, v-*walk*. A walk with no repeated vertices is a *path*. We denote a u, v-path by symbols like uPv, making the endvertices explicit for readability. The vertices on uPv except for u and v are called *internal*. The *length* of a walk uPv is the number of edges (with repetitions) and denoted $l(uPv)$. If x is a vertex on the u, v-path uPv then we write uPx for the u, x-subpath of uPv, and xPv for the x, v-subpath of uPv. Two paths uPv and vQw can be concatenated into the walk $uPvQw$ with the obvious interpretation.

Let $\operatorname{dist}(u, v)$ denote the *distance* between u and v in a connected graph G, which is the length of the shortest u, v-path; with the understanding that $\operatorname{dist}(u, u) = 0$. The *eccentricity* of vertex u, denoted $e(u)$, is $\max\{ \operatorname{dist}(u, v) \mid v \in V(G) \}$. The *diameter* of G, denoted $\operatorname{diam} G$, is $\max\{ e(u) \mid u \in V(G) \}$. The *radius* of G, denoted $\operatorname{rad} G$, is $\min\{ e(u) \mid u \in V(G) \}$.

The treewidth of a graph is the width of an optimal *tree decomposition*, an auxiliary structure that maintains the structure of G in a sparse fashion. (See Section 2.4 for a full definition.) Our algorithm requires a tree decomposition as input, and we note that this is provided within the time bounds of our own algorithm by a recent result by Bodlaender *et al.* [2] that we can state as follows:

▶ **Theorem 2** (Bodlaender et al.). *Given a graph G with n vertices and treewidth k, a nice tree decomposition of G of width $O(k)$ and with $O(nk)$ nodes can be computed in time $n \exp O(k)$.*

We also need the fact that given such a tree decomposition, we can compute pairwise distances quickly for all vertices in the same piece. For the purposes of exposition, we abstract this to a more general result due to Chaudhuri and Zariolagis [4] that falls slightly short of the ambitions on Theorem 1, by a factor $\log n$. We show in Section 3 how to replace this result with an explicit and self-contained construction with a better bound so as to establish the bound in Theorem 1.

▶ **Theorem 3** (Chaudhuri and Zariolagis). *Let G denote an n-vertex graph given with a tree decomposition of width k. In time $O(k^3 n \log n)$ we can compute a data structure such that for any $u, v \in V(G)$ we can compute $\operatorname{dist}(u, v)$ in time $O(k^3)$.*

Chaudhuri and Zariolagis [4] present various trade-offs between construction and query time that need not interest us here; the important part is the subquadratic dependency on n in the construction time and polynomial dependency on the treewidth, so that the time to compute the distance between a pair of vertices becomes negligible. We note that in itself, the above structure brings us no closer to our goal: To compute the diameter, we still need to evaluate $\operatorname{dist}(u, v)$ for all pairs of vertices, in time $O(n^2 k^3)$.

2.2 Distance profiles

Let R denote a fixed integer; think of R as the *range* of distances we want to keep track of, ideally $\operatorname{diam} G \leq R$.

We denote by π a partition (U, V, W) of $V(G)$, where $V = \{v_1, \ldots, v_r\}$ is a separator in G separating U and W.

(We will arrive at these partitions as induced by neighbouring nodes in a tree decomposition. We need the next two lemmas when we traverse the decomposition in both directions, which is why we avoid the terms 'above' and 'below' in favour of 'left' and 'right'. However, the reader is encouraged to think of V as a piece in the tree decomposition, with U denoting the vertices 'below' it, and W those 'above.' We give a more general framework here in order to avoid the proliferation of ultimately similar arguments and tedious case analyses.)

We will pay special attention to paths whose internal vertices belong to U. In particular, for $u \in U \cup V$ and $v \in V$ a u, v-path is *U-internal* if all its internal vertices belong to U. (As a possible source of confusion, the one-edge path uv is trivially U-internal. In fact, both u and v may belong to V, so a U-internal path might completely avoid U.) We then let $\operatorname{dist}_\pi(u, v) \in \{0, \ldots, R\} \cup \{\infty\}$ denote the length of the shortest U-internal u, v-path of length at most R, or ∞ if no such path exists.

For each vertex $u \in U$ define the *distance profile* $p_\pi(u)$ as the vector of its distances to V:

$$p_\pi(u) = (\operatorname{dist}_\pi(u, v_1), \ldots, \operatorname{dist}_\pi(u, v_r)) .$$

We let D_π denote the set of all such distance profiles,

$$D_\pi = \bigcup_{u \in U} p_\pi(u) .$$

This set contains the information that we will maintain while traversing the tree decomposition of the input graph; a vector of distances $(d_1, \ldots, d_r)$ is contained in D_π exactly if there exists a vertex $u \in U$ with that distance profile, but we forget the identity of that vertex, and how many there are. In particular, $|D_\pi| \leq (R+2)^r$, whereas U itself may be much larger.

2.3 Maintaining distance profiles over neighbouring cuts

Consider two partitions π and π' that differ only in a single vertex.

We consider two different ways in which π and π' can differ by moving a single vertex v_r 'one part to the left' in the partition. Thus, either the vertex v_r is moved from W to V, 'introducing' it to V; or the vertex v_r is moved from V to U, 'forgetting' it from V.

The next two lemmas handle each case separately.

▶ **Lemma 4** (Introduce). *Let $\pi' = (U, V - \{v_r\}, W \cup \{v_r\})$ be a partition with $V = \{v_1, \ldots, v_r\}$ and consider the partition π given by (U, V, W).*

1. *$D_\pi = D_{\pi'} \times \{\infty\}$.*
2. *For $v_i, v_j \in V$ with $1 \leq i < j \leq r$, we have*

$$\operatorname{dist}_\pi(v_i, v_j) = \begin{cases} \operatorname{dist}_{\pi'}(v_i, v_j) , & \text{if } j < r ; \\ l(v_i v_r) , & \text{if } j = r \text{ and } v_i v_r \in E ; \\ \infty , & \text{otherwise} . \end{cases}$$

Proof. Consider the partition π.

Since $V - \{v_r\}$ is a separator, there are no edges from v_r to any vertex in U. In particular, no path that includes any vertex in U can include v_r.

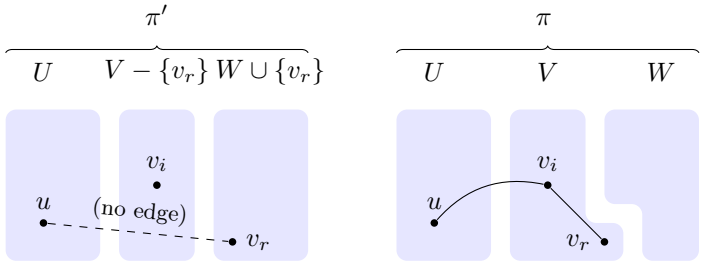


Figure 1 Vertex v_r is introduced to V.

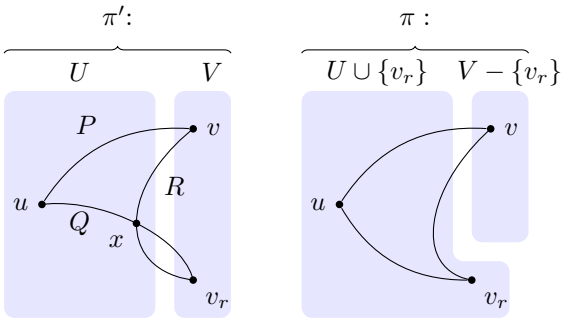


Figure 2 Vertex v_r is forgotten from V. The part W is not shown.

For the first part, consider the distance vector $(\mathrm{dist}_\pi(u, v_1), \ldots, \mathrm{dist}_\pi(u, v_r)) \in D_\pi$. The last coordinate must be ∞ because there is no U-internal path from $u \in U$ to v_r. Moreover, there is no path from u to any other $v_i \in V$ passing through v_r either, so the remaining distances are unchanged.

For the second part, consider a pair of vertices $v_i, v_j \in V$. If $j \neq r$ then their U-internal distance does not change. If $v_j = v_r$ then the only possible U-internal path is the edge $v_i v_r$ if it exists. ◀

The other case is more interesting:

▶ **Lemma 5** (Forget). *Let $\pi' = (U, V, W)$ be a partition with $V = \{1, \ldots, v_r\}$ and consider the partition π given by $(U \cup \{v_r\}, V - \{v_r\}, W)$.*

1. *D_π consists of the vector $(\mathrm{dist}_{\pi'}(v_1, v_r), \ldots, \mathrm{dist}_{\pi'}(v_{r-1}, v_r))$ and for each $(d'_1, \ldots, d'_r) \in D_{\pi'}$ the vectors $(d_1, \ldots, d_{r-1})$ given by*

$$d_j = \min\{d'_j, d'_r + \mathrm{dist}_{\pi'}(v_j, v_r)\}\,. \tag{3}$$

2. *For each $v_i, v_j \in V$ with $1 \leq i < j < r$, we have*

$$\mathrm{dist}_\pi(v_i, v_j) = \min\{\mathrm{dist}_{\pi'}(v_i, v_j), \mathrm{dist}_{\pi'}(v_i, v_r) + \mathrm{dist}_{\pi'}(v_j, v_r)\}\,.$$

Proof. Let $u \in U \cup V$ and $v \in V - \{v_r\}$. Let uSv be a shortest $U \cup \{v_r\}$-internal path. Let uPv, uQv_r, and $v_r Rv$ be shortest U-internal paths, see Fig. 2.

We will show that

$$l(uSv) = \min\{l(uPv), l(uQv_r) + l(v_r Rv)\}\,. \tag{4}$$

There are two cases. If the path uSv does not use v_r then it is U-internal. In particular, it has same length as the shortest U-internal path uPv. Let x be the earliest vertex on uQv_r

also appearing on v_rRv, possibly $x = v_r$. If $x \neq v_r$ then $uQxRv$ is a U-internal u, v-path, so that

$$l(uSv) = l(uPv) \leq l(uQxRv) \leq l(uQv_rRv) = l(uQv_r) + l(v_rRv)\,,$$

establishing (4) in this case. If $x = v_r$ then uQv_rRv is a path, and therefore no shorter than uSv. Thus,

$$l(uQv_r) + l(v_rRv) = l(uQv_rRv) \geq l(uSv) = l(uPv).$$

establishing (4) in this case.

If the path uSv does contain v_r then it decomposes into uSv_r and v_rSv, both of which are shortest U-internal paths. Thus we can write

$$l(uPv) \geq l(uSv) = l(uSv_rSv) = l(uSv_r) + l(v_rSv) = l(uQv_r) + l(v_rRv)\,,$$

where the first inequality merely observes that uPv is a shortest path in a smaller set of internal vertices than uSv. We have established (4).

To establish the lemma, set $v = v_j$. Provided all lengths are bounded by R, we can give (4) as

$$\mathrm{dist}_\pi(u, v_i) = \min\{\mathrm{dist}_{\pi'}(u, v_i), \mathrm{dist}_{\pi'}(u, v_r) + \mathrm{dist}_{\pi'}(v_r, v_i)\}\,. \tag{5}$$

For $u \in U \cup \{v_r\}$, write $p_\pi(u) = (d_1, \ldots, d_{r-1})$. If $u = v_r$ then the distances are simply given by $d_i = \mathrm{dist}_{\pi'}(v_i, v_r)$, because no U-internal v_i, v_r-path can use v_r as an internal node. For every other $u \in U$ let $p_{\pi'}(u) = (d'_1, \ldots, d'_r)$. Then (5) gives the first part of the lemma with $d_i = \mathrm{dist}_\pi(u, v_i)$ and $d'_i = \mathrm{dist}_{\pi'}(u, v_i)$. Finally, if $u \in V$ with $u = v_i$ for some $i \in \{1, \ldots, v_{r-1}\}$ then (5) gives the second part of the lemma.

It remains to verify that (5) holds also if some of the lengths in (4) exceed R. If $l(uSv) > R$ then $\mathrm{dist}_\pi(u, v_i) = \infty$. We already observed that $l(uSv)$ is at most $l(uPv)$ (the length of a shortest u, v-path internal in a subset of vertices) and also at most $l(uQv_rRv)$ (the length of a u, v-walk internal in the same set vertices). Thus, both values on the right hand side of (5) are also ∞. Conversely, if $l(uSv) \leq R$ then $l(uPv) \leq R$, in which case $\mathrm{dist}_{\pi'}(u, v_i) = l(uPv)$, or $l(uQv_r) + l(v_rRr) \leq R$, in which case both $\mathrm{dist}_{\pi'}(u, v_r) = l(uQv_r)$ and $\mathrm{dist}_{\pi'}(v_r, v_i) = l(v_rRv_i)$, because lengths are nonnegative. In any case, the minimum operation will pick the correct value. ◀

2.4 Tree decompositions

We consider the standard notion of a (nice) tree decomposition. To fix notation, consider a rooted, binary tree T and associate with each node $t \in T$ a set V_t of vertices, called a *piece*. Such a tree T is *nice* if it satisfies the following conditions:

1. if t is a leaf or the root then V_t is a singleton,
2. if t has a single child t' then there exists a vertex $v \in V$ such that either $v \notin V_{t'}$ and $V_t = V_{t'} \cup \{v\}$, in which case we say that t *introduces* v, or $v \in V_t t'$ and $V_t = V_{t'} - \{v\}$, in which case we say that t *forgets* v.
3. if t has two children t' and t'' then $V_t = V_{t'} = V_{t''}$.

The tree T forms a *tree decomposition of* G if the following conditions hold:

1. $V(G) = \bigcup_{t \in T} V_t$.
2. for each $uv \in E(G)$ there exists $t \in T$ such that $u, v \in V_t$.
3. for each $u \in V(G)$, the set of nodes $t \in T$ such that $u \in V_t$ are connected.

The *width* of a tree decomposition is $\max_{t\in T} |V_t|-1$. The *treewidth* of a graph is the minimum width of any tree decomposition of G.

For each node t, we define a tripartition $\pi(t)$ as the disjoint partition (U, V, W) of $V(G)$ given as follows:

1. V is the piece V_t associated with t in the tree decomposition.
2. Intuitively, U are the vertices 'below' t. Formally, let T' denote the successors of t in T. Then
$$U = \{\, V_{t'} \mid t' \in T' \,\} - V_t\,.$$
3. The remaining vertices belong to W, so $W = V(G) - (V_t \cup U)$.

We think of W as the vertices 'above' the node t, but remember that W includes vertices that are associated with siblings of t, so 'above' is a slightly misleading term.

We are finally ready to define our distance measures. For $u \in U \cup V$ and $v \in V$ consider $\operatorname{dist}_{\pi(t)}(u, v)$. Intuitively, this is the 'below'-internal distance in the sense that U contains the vertices 'below' the current node in the tree decomposition. The corresponding set of distance vectors is $D_{\pi(t)}$.

Symmetrically, from $\pi(t) = (U, V, W)$ we define the 'reverted' partition $\rho(t)$ as (W, V, U). Then, for $w \in W \cup V$ and $v \in V$ we consider $\operatorname{dist}_{\rho(t)}(w, v)$. Intuitively, this is the 'above'-internal distance in the sense that W are the vertices 'above' the current node in the tree decomposition. The corresponding set of distance profiles is $D_{\rho(t)}$.

We proceed to establish that the two sets $D_{\pi(t)}$ and $D_{\rho(t)}$ can be computed for each node $t \in T$ of the tree decomposition. The 'below'-values are established bottom-up, after which the 'above'-values are established top-down.

▶ **Algorithm D** (Distance profiles). *Given a graph G, its tree decomposition T, and an integer R such that $R \geq \operatorname{diam} G$, this algorithm computes the sets $D_{\pi(t)}$ and $D_{\rho(t)}$ for each $t \in T$.*

The algorithm works by traversing the tree decomposition twice, also computing for each $t \in T$ and each pair of vertices $u, v \in V_t$, the distances $\operatorname{dist}_{\pi(t)}(u, v)$ and $\operatorname{dist}_{\rho(t)}(u, v)$.

D1 – Traverse T bottom-up. For each leaf t of T, set $D_{\pi(t)} = \varnothing$. Traverse T bottom-up using Steps D2–D4. Then go to Step D5.

D2 – Introduce. If node t with child t' introduces vertex v_r to $V_{t'} = \{v_1, \dots, v_{r-1}\}$ then compute $\operatorname{dist}_{\pi(t)}(d_i, d_j)$ for $i \leq i < j \leq r$ and $D_{\pi(t)}$ from $\operatorname{dist}_{\pi(t')}(d_i, d_j)$ and $D_{\pi(t')}$ using Lemma 4 with $\pi = \pi(t)$ and $\pi' = \pi(t')$.

D3 – Forget. If node t with child t' forgets vertex v_r from $V_{t'} = \{v_1, \dots, v_r\}$ then compute $\operatorname{dist}_{\pi(t)}(d_i, d_j)$ for $1 \leq i < j < r$ and $D_{\pi(t)}$ from $\operatorname{dist}_{\pi(t')}(v_i, v_j)$ and $D_{\pi(t')}$ using Lemma 5 with $\pi = \pi(t)$ and $\pi' = \pi(t')$.

D4 – Join. If t joins t' and t'', with $V_t = V_{t'} = V_{t''} = \{1, \dots, v_r\}$ then set
$$D_{\pi(t)} = D_{\pi(t')} \cup D_{\pi(t'')}\,,$$
and for each $u, v \in V_t$ set
$$\operatorname{dist}_{\pi(t)}(u, v) = \min\{\operatorname{dist}_{\pi(t')}(u, v), \operatorname{dist}_{\pi(t'')}(u, v)\}\,.$$

D5 – Traverse T top-down. At the root t of T, set $D_{\rho(t)} = \varnothing$. Traverse T top-down using Steps D6–D8. Then return.

D6 – Child of introduce. If t is the child of a node t' introducing v_r to $V_t = \{v_1, \dots, v_{r-1}\}$ then compute $\operatorname{dist}_{\rho(t)}(v_i, v_j)$ for $1 \leq i < j < r$ and $D_{\rho(t)}$ from $\operatorname{dist}_{\rho(t')}(v_i, v_j)$ and $D_{\rho(t')}$ using Lemma 5 with $\pi = \rho(t)$ and $\pi' = \rho(t')$.

D7 – Child of forget. If t is the child of a node t' forgetting v_r from $V_t = \{v_1, \ldots, v_r\}$ then compute $\mathrm{dist}_{\rho(t)}(v_i, v_j)$ for $1 \leq i < i \leq r$ and $D_{\rho(t)}$ from $\mathrm{dist}_{\rho(t')}(v_i, v_j)$ and $D_{\rho(t')}$ using Lemma 4 with $\pi = \rho(t)$ and $\pi' = \rho(t')$.

D8 – Child of join. If t is the child of a join node t' and the sibling of t'' then set

$$D_{\rho(t)} = D_{\rho(t')} \cup D_{\pi(t'')}\,,$$

and for each $u, v \in V_t$, set

$$\mathrm{dist}_{\rho(t)}(u, v) = \min\{\mathrm{dist}_{\rho(t')}(u, v), \mathrm{dist}_{\pi(t'')}(u, v)\}\,.$$

Note the asymmetry in Steps D4 and D8. On the way up, we join information from 'below' the child nodes; on the way down we join information from 'above' the parent node and 'below' the other sibling. The correctness of the simple minimum operation in those two steps crucially rests on the fact that $\mathrm{dist}_{\pi''}$ records only the distances that are internal to the first part U of π'' (and similarly for π' or ρ'). In particular, no new paths internal to the first parts of π or ρ (both of which contain many more vertices than U) are introduced at these join nodes.

We can now compute the eccentricity of each vertex from the following lemma:

▶ **Lemma 6** (Eccentricities). *Let v be a vertex in G with $e(v) \leq r$. For any piece $V_t = \{v_1, \ldots, v_r\}$ such that $v \in V_t$, we have*

$$e(v) = \max_{(d_1,\ldots,d_r)} \min_{1\leq i\leq r} d_i + \mathrm{dist}(v, v_i)\,, \tag{6}$$

where the maximum is taken over all distance profiles $(d_1, \ldots, d_r) \in D_{\pi(t)} \cup D_{\rho(t)}$.

Proof. Consider a shortest path uPv. Assume $u \in U$, write π for $\pi(t)$ and let $(d_1, \ldots, d_r) = p_\pi(u)$. Let v_i denote the first vertex on uPv that belongs to V_t, possibly $v_i = u$. Because uPv_i is a shortest paths and it is U-internal we have $l(uPv_i) = d_i$. Because v_iPv is a shortest path we have $l(v_iPv) = \mathrm{dist}(v_i, v)$. Thus, $l(uPv) = l(uPv_iPv) = d_i + \mathrm{dist}(v, v_i)$. To see that uPv attains the minimum over all i on the right and side of the expression, assume for a moment that there exits j such that $d_j + \mathrm{dist}(v, v_j) < d_i + \mathrm{dist}(v, v_i)$. Choose a shortest U-avoiding path uQv_j of length d_j and a shortest path v_jRv of length $\mathrm{dist}(v, v_j)$. Then the walk uQv_jRv would be shorter than the shortest path uPv, which is absurd. Thus,

$$\begin{aligned} l(uPv) &= \min_{1\leq i\leq r} d_i + \mathrm{dist}(v, v_i) \\ &= \min_{1\leq i\leq r} \left[p_{\pi(u)}\right]_i + \mathrm{dist}(v, v_i)\,, \end{aligned}$$

where $\left[p_{\pi(u)}\right]_i$ is the ith coordinate of the vector $p_{\pi(u)}$. Maximising over all $u \in U$ we arrive at

$$\begin{aligned} \max_{u\in U} l(uPv) &= \max_{u\in U} \min_{1\leq i\leq r} \left[p_\pi(u)\right]_i + \mathrm{dist}(v, v_i) \\ &= \max_{p_\pi(u)\in D_\pi} \min_{1\leq i\leq r} \left[p_\pi(u)\right]_i + \mathrm{dist}(v, v_i) \\ &= \max_{(d_1,\ldots,d_r)\in D_\pi} \min_{1\leq i\leq r} \{d_i + \mathrm{dist}(v, v_i)\}\,, \end{aligned}$$

by definition of D_π. Repeating this argument to W and the corresponding distance profile vectors indexed by ρ we see that the length of the longest shortest path uPv is indeed expressed by (6). ◀

Proof of Theorem 1, vertex-superlinear. Assume without loss of generality that G is connected.

First assume that we know a bound R with $R \geq \operatorname{diam} G$ but $R = O(\operatorname{diam} G)$. We compute a tree decomposition T of width $O(k)$ using Theorem 2. We then run Algorithm D on input G and T. From the resulting D_π and D_ρ, we use Theorem 3 and Lemma 6 to compute all eccentricities. From these values we can easily compute $\operatorname{diam} G = \max_v e(v)$, $\operatorname{rad}(G) = \min_v e(v)$, and list the vertices on the perimeter and in the center.

It remains to analyse the running time. Algorithm D performs two passes through T, which as $O(kn)$ nodes. At each node, the computation is dominated by the applications of Lemmas 4 and 5. This entails processing D_π and $D_{\pi'}$, both of which are bounded by $(R+2)^{O(k)}$. The total running time of algorithm D therefore bounded by $|T|(R+2)^{O(k)} = n \exp O(k \log \operatorname{diam} G)$. When we apply Lemma 6, we need to compute the pairwise distances $\operatorname{dist}(v, v_i)$ for $v, v_i \in V_t$ for each tree node $t \in T$. This entails $|T|\binom{O(k)}{2}$ computations, each requiring time $O(|T|k^6)$ after $O(k^3 n \log n)$ preprocessing according to Theorem 3. Since we have $|T| = O(kn)$, this step takes time $n^{1+o(1)} \operatorname{poly}(k)$.

If we do not know R, we can search for it iteratively $R = 2, 3, \ldots$, until no infinite eccentricities appear. This increases the running time by a factor of at most $\operatorname{diam} G$, which is absorbed in our time bounds. ◀

3 Vertex-linear time

We finish the proof of Theorem 1 with the desired time bound by showing how the distances between vertices within the same piece can be computed in time linear in n. The idea is to construct a graph whose edge lengths model those paths that have all their internal nodes outside of the current piece. A standard all-pairs shortest paths computation among those vertices then suffices.

▶ **Algorithm L** (Linear time distances). *Given a graph and a tree decomposition T, this algorithm computes* $\operatorname{dist}(u, v)$ *for each pair of vertices u, v belonging to V_t.*

L1 – Internal distances. Compute $\operatorname{dist}_{\pi(t)}$ and $\operatorname{dist}_{\rho(t)}$ as in algorithm D, with $R = n$.
L2 – Traverse T top-down. Process T top-down. At each node t perform Steps L3 and L4.
L3 – Construct H. Let H be the complete graph on vertex set V_t. If t has a single child then set $l(u, v) = \min\{\operatorname{dist}_{\pi(t)}(u, v), \operatorname{dist}_{\rho(t)}(u, v)\}$. If t has two children t' and t'' then set $l(u, v) = \min\{\operatorname{dist}_{\pi(t')}(u, v), \operatorname{dist}_{\pi(t'')}(u, v), \operatorname{dist}_{\rho(t)}(u, v)\}$.
L4 – Find distances between all pairs in H. Run the Bellman–Ford algorithm to compute the distances $\operatorname{dist}_H(u, v)$ in H between each pair of vertices $u, v \in V_t$. Let $\operatorname{dist}(u, v) = \operatorname{dist}_H(u, v)$.

▶ **Theorem 7.** *Algorithm L is correct. If G has n vertices and T has width k then the algorithm runs in time $O(nk^3)$.*

Proof. In the first step, the algorithm mimics algorithm D but avoids the computation of D_ϕ and D_ρ. This requires, at each node $t \in T$, the constant-time computations described in Lemmas 4 and 5 for each pair $u, v \in V_t$. Thus, the running time is dominated by running the all-pairs shortest paths computation in Step L4, which runs in time $O(|V(H)|^3)$. ◀

We note that the performance of the algorithm is dwarfed by the requirements of actually computing a tree decomposition, so for the polynomial dependency on k it is crucial that T be provided as part of the input.

This finishes the proof of Theorem 1. Note that the computations of algorithm L can be performed during the top-down traversal in Steps D6–D8, just after $\mathrm{dist}_{\rho(t)}$ is found. Thus, a unified presentation of the algorithm could be given in only two traversals.

4 Conclusion

Our constructions do extend readily to *directed* graphs, and parameters like directed eccentricity, source radius, etc. can be computed within the same time bound as their undirected counterparts. We choose to claim this here without proof, since a more general presentation that encompasses directed graphs incurs considerable expository overhead without providing much insight outside of what is already present in the appendix of [1]. The most notable changes arise in the computation of *round-trip distance* from u to v, which is the minimum of the sums of the lengths of directed paths uPv and vQu. To compute these values, we need to change the definition of distance profile vectors to matrices: For partition $\pi = (U, \{v_1, \ldots, v_r\}, W)$ consider the $r \times r$ matrix M with $m_{ij} = d$ if there is a vertex $u \in U$ such that there exist U-internal shortest directed paths v_iPu and uPv_j with $d = l(v_iPu) + l(uPv_j)$. The set D_π then contains all matrices with elements bounded by R that are realised by some $u \in U$. The total size of this set is bounded by $(R+2)^{r^2}$. When v_r is forgotten in the tree decomposition, the entries m_{ir} and m_{rj} account for directed U-internal paths of length $m_{ir} + m_{ij}$. The resulting time running time, suppressing several details, becomes $n \exp O(k^2 \log \operatorname{diam} G)$, much like the bound of $O(k^2 n \log^{k^2-1} n)$ of [1]. In both constructions, we notice an exponential dependency on the square of the treewidth.

Our bounds rely on the fact that we store merely the *existence* of vertex pairs at certain distances, not the *number* of such pairs. Thus, our constructions have nothing to contribute to various graph distance measures that involve counting, such as the Wiener index, the median, or closeness centrality.

The main question opened up by our algorithms is the complexity of computing superconstant diameter. In particular, an algorithm with running time $n \exp O(k)$ could exist even for diameter $\omega(1)$. However, we conjecture that the diameter of a graph with treewidth and diameter k cannot be computed in time $n^{2-\epsilon} \exp o(k \log k)$.

Acknowledgements. I am happy that Virginia Vassilevska Williams told me about this question.

References

1 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. 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, SODA 2016, Arlington, Va, USA, January 10–12, 2016*, pages 377–391. SIAM, 2016.

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

3 Sergio Cabello and Christian Knauer. Algorithms for bounded treewidth with orthogonal range searching. *Comput. Geom.*, 42(9):815–824, 2009.

4 Shiva Chaudhuri and Christos D. Zaroliagis. Shortest path queries in digraphs of small treewidth. *Algorithmica*, 27(3):212–226, 2000.

5 David Eppstein. Subgraph isomorphism in planar graphs and related problems. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1995. San Francisco, California*, pages 632–640. ACM/SIAM, 1995.

6 Russel Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

7 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms for graphs of bounded treewidth are probably optimal. In *Proceedings of the Twenty-Second Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 777–789. SIAM, 2011.

8 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1–4, 2013*, pages 515–524. ACM, 2013.

9 Dan E. Willard. New data structures for orthogonal range queries. *SIAM J. Comput.*, 14(1):232–253, 1985.

Lower Bounds for Protrusion Replacement by Counting Equivalence Classes*†

Bart M. P. Jansen[1] **and Jules J. H. M. Wulms**[2]

1 **Eindhoven University of Technology, Eindhoven, The Netherlands**
b.m.p.jansen@tue.nl
2 **Eindhoven University of Technology, Eindhoven, The Netherlands**
j.j.h.m.wulms@tue.nl

Abstract

Garnero et al. [SIAM J. Discrete Math. 2015, 29(4):1864–1894] recently introduced a framework based on dynamic programming to make applications of the *protrusion replacement* technique constructive and to obtain explicit upper bounds on the involved constants. They show that for several graph problems, for every boundary size t one can find an explicit set $\mathcal{R}_t$ of *representatives*. Any subgraph H with a boundary of size t can be replaced with a representative $H' \in \mathcal{R}_t$ such that the effect of this replacement on the optimum can be deduced from H and H' alone. Their upper bounds on the size of the graphs in $\mathcal{R}_t$ grow triple-exponentially with t. In this paper we complement their results by lower bounds on the sizes of representatives, in terms of the boundary size t. For example, we show that each set of planar representatives $\mathcal{R}_t$ for the Independent Set problem contains a graph with $\Omega(2^t/\sqrt{4t})$ vertices. This lower bound even holds for sets that only represent the planar subgraphs of bounded pathwidth. To obtain our results we provide a lower bound on the number of equivalence classes of the canonical equivalence relation for Independent Set on t-boundaried graphs. We also find an elegant characterization of the number of equivalence classes in general graphs, in terms of the number of monotone functions of a certain kind. Our results show that the number of equivalence classes is at most 2^{2^t}, improving on earlier bounds of the form $(t+1)^{2^t}$.

1998 ACM Subject Classification G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases protrusions, boundaried graphs, independent set, equivalence classes, finite integer index

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.17

1 Introduction

Protrusion replacement is a versatile tool for attacking optimization problems on graphs. When applied to solve an optimization problem on a graph G, the main idea is the following: repeatedly replace a *protrusion* subgraph $H \subseteq G$ that interacts with the rest of G through a small boundary, by a smaller *representative* subgraph H'. Suppose that we can ensure that (i) the change Δ in the optimum caused by this replacement only depends on H and H', and that (ii) we can efficiently analyze H to find a suitable replacement H' and the corresponding Δ. Then we can solve the problem on G by solving it on the smaller graph and adding Δ to the final result. In recent years, protrusion replacement has been applied to obtain approximation

* A full version of the paper is available at https://arxiv.org/abs/1609.09304.
† This work was supported by NWO Veni grant "Frontiers in Parameterized Preprocessing" and NWO Gravitation grant "Networks".

algorithms [7, 8], kernelization algorithms [1, 7, 8, 10, 15], and fixed-parameter tractable algorithms [8, 15]. The generality of protrusion replacement comes at a price: it often results in proofs that efficient algorithms of a certain type *exist*, without showing explicitly how such algorithms can be *constructed* and without giving any explicit bounds on the constant factors involved in the analysis. This non-constructivity stems from the use of a property called *finite integer index* (FII, defined below). It is used to argue that for every constant boundary size t, there is a finite set of representatives $\mathcal{R}_t$ such that any t-boundaried subgraph H can safely be replaced by some representative $H' \in \mathcal{R}_t$, as described above. The key issue is that FII only guarantees that a finite set of representatives exist, without showing how to find it, how large the set is, or how many vertices the representative subgraphs have.

To deal with the issue of non-constructivity, Garnero et al. [12] introduced a framework based on dynamic programming. They showed that explicit bounds for the sizes of representatives can be obtained by analyzing the number of states required to solve the problem on graphs of bounded treewidth. By presenting explicit dynamic programming algorithms for problems such as r-Independent Set and r-Dominating Set, they were able to derive upper bounds on the size of representatives in terms of the boundary size t. These upper bounds grow very quickly with t, in some cases triple-exponentially. Garnero et al. [12, §7] suggest to examine to what extent this exponential dependance is unavoidable. We pursue this direction by presenting lower bounds.

Boundaried graphs and equivalence. To state our results we have to introduce some terminology.[1] We only consider undirected, finite, simple graphs. Let t be a positive integer. A t-boundaried graph G consists of a vertex set $V(G)$, an edge set $E(G) \subseteq \binom{V(G)}{2}$, and an injective labeling $\lambda_G \colon \{1, \ldots, t\} \to V(G)$ that identifies t distinct *boundary vertices* in the graph. The *boundary* of the graph is the set $B_G := \{\lambda_G(1), \ldots, \lambda_G(t)\}$. Two t-boundaried graphs G and H can be *glued* together on their boundary, resulting in the boundaried graph $G \oplus H$ that is obtained from the disjoint union of G and H by identifying corresponding boundary vertices and removing any parallel edges that are introduced. That is, we merge $\lambda_G(i)$ with $\lambda_H(i)$ for each $i \in [t]$. An *optimization problem* Π on graphs assigns to every (unboundaried) graph G an optimal solution value $\Pi(G) \in \mathbb{Z}$. We will also write $\Pi(G)$ for a boundaried graph G to denote the optimum of the underlying unboundaried graph. Two t-boundaried graphs G and H are *equivalent* with respect to Π, denoted $G \equiv_{\Pi,t} H$, if there exists a *transposition constant* $\Delta \in \mathbb{Z}$ such that for every t-boundaried graph F:

$$\Pi(G \oplus F) = \Pi(H \oplus F) + \Delta. \tag{1}$$

It is easy to see that $\equiv_{\Pi,t}$ is an equivalence relation. Problem Π has *finite integer index* if $\equiv_{\Pi,t}$ has a finite number of equivalence classes for each fixed t. In the remainder, we omit the subscript t when it is clear from the context. Observe that these notions formalize the idea behind protrusion replacement sketched above: if $G \equiv_{\Pi,t} H$, then replacing G by H changes the optimum by exactly Δ.

Our results. We analyze the canonical equivalence relation $\equiv_{\text{is},t}$ on t-boundaried graphs for the Independent Set (is) problem, which asks for the maximum size of an independent

[1] To avoid an abundance of cumbersome definitions, our terminology differs slightly from that in earlier work (cf. [2, 3], [5, §2]). In particular, we do not allow t-boundaried graphs with fewer than t boundary vertices. The fact that we consider optimization problems as in [5], rather than decision problems as in [1, 12], forms no essential difference; our lower bounds also apply to those settings.

set of pairwise non-adjacent vertices. We focus on Independent Set due to its simple combinatorial structure, but our techniques carry over to Dominating Set, as explained in §8. Define a *set of representatives* for $\equiv_{\text{IS},t}$ to be a set $\mathcal{R}_t$ of t-boundaried graphs, such that for every t-boundaried graph G there exists $H \in \mathcal{R}_t$ with $G \equiv_{\text{IS},t} H$. Let the *critical size* of a set of representatives be the number of vertices of its largest graph. We aim to give a lower bound on the critical size of any set of representatives for Independent Set in terms of t. Our approach consists of two steps. First, we construct a large set of pairwise nonequivalent graphs to give a lower bound on the number of equivalence classes of $\equiv_{\text{IS},t}$. Then we use a counting argument to leverage this into a lower bound on the critical size. Observe that each equivalence class must be represented by a different graph. It follows that if the number of distinct t-boundaried graphs with at most s vertices is smaller than the number of equivalence classes, then the critical size of any set of representatives must be larger than s to give each class a distinct representative. By relating the number of small graphs to the number of equivalence classes, we therefore obtain the desired lower bounds.

Protrusion replacement is often applied in the context of restricted graph classes, where the protrusions to be replaced are known to have bounded treewidth and may even belong to a family of embeddable graphs such as planar graphs. With these application areas in mind, we develop our lower bounds to apply even when we wish only to have a representative for each equivalence class that contains a planar graph whose treewidth is $t+\mathcal{O}(1)$, for boundary size t. To find a large set of nonequivalent graphs we adapt a construction of Lokshtanov et al. [16], which they used to prove that Independent Set on graphs of treewidth w cannot be solved in time $\mathcal{O}^*((2-\varepsilon)^w)$ for any $\varepsilon > 0$ unless the Strong Exponential Time Hypothesis fails. We show that the graphs they construct can be made planar while increasing the treewidth (and in fact the pathwidth) by only a small additive term. More importantly, we show how to use this adapted construction to build a set of $M(t)-2$ planar graphs of small treewidth which are pairwise nonequivalent under $\equiv_{\text{IS},t}$, for all t. The term $M(t) \geq 2^{\binom{t}{\lfloor t/2 \rfloor}} \geq 2^{2^t/\sqrt{4t}}$ denotes the t-th *Dedekind number*, which counts the number of *monotone Boolean functions* of t variables. The number of equivalence classes therefore grows double-exponentially with t. Using the counting argument above, this allows us to give a lower bound of $\Omega(\log M(t)) \geq \Omega(2^t/\sqrt{4t})$ on the critical size of any set of planar representatives for the equivalence classes of $\equiv_{\text{IS},t}$ that contain a planar graph of bounded pathwidth.

While developing a lower bound on the number of equivalence classes for planar graphs of bounded pathwidth, we also found an exact characterization of the number of equivalence classes of $\equiv_{\text{IS},t}$ in general. We define a natural class of functions from $\{0,1\}^t$ to $\mathbb{N}$ that we call *t-representative functions*. We give a bijection between the t-representative functions and the equivalence classes of $\equiv_{\text{IS},t}$ for t-boundaried graphs. As we will show that all monotone Boolean functions which are not constantly zero yield a distinct t-representative function, this gives a lower bound of $M(t)-1$ on the number of equivalence classes of $\equiv_{\text{IS},t}$. On the other hand, we show that the number of such functions is at most 2^{2^t-1}. The double-exponential lower bound for the number of equivalence classes containing a bounded-pathwidth planar graph is therefore not far off from the upper bound of 2^{2^t-1} in general graphs. The fact that the base of the double-exponential in this expression is independent of t is noteworthy. The naive way to bound the number of equivalence classes is to associate a table to each t-boundaried graph. For each subset S of the boundary vertices B, the table stores the maximum size of an independent set containing no vertex of $B \setminus S$. There are at most $t+1$ distinct values in such a table, and two boundaried graphs whose tables differ in the same universal constant in all positions are easily shown to be equivalent. As there are 2^t entries in the table, and $t+1$ different options per entry, this gives an upper bound of $(t+1)^{2^t}$ on the

number of equivalence classes. Garnero et al. [12, Lemma 3.7] obtain the same bound using a subtly different definition for the table. Our result of 2^{2^t-1} yields a slight improvement.

2 Preliminaries

We use $\mathbb{N}$ to denote the natural numbers, including 0. For a positive integer n and a set X we use $\binom{X}{n}$ to denote the collection of all subsets of X of size n. The *power set* of X is denoted 2^X. The set $\{1, \ldots, n\}$ is abbreviated as $[n]$. A *Boolean function* is a function of the form $f\colon \{0,1\}^n \to \{0,1\}$. We sometimes use the equivalent view that a Boolean function assigns a 0/1-value to every subset $S \subseteq [n]$, which is the value of f when the arguments whose index is in S are set to 1 and the remaining arguments are set to 0. A Boolean function $f\colon 2^{[n]} \to \{0,1\}$ is *monotone* if $f(S') \leq f(S)$ whenever $S' \subseteq S \subseteq [n]$. We will call Boolean functions in this form set-functions, and may replace $[n]$ by other finite sets of ordered elements. A formula in conjunctive normal form (CNF) is *monotone* if no literal appears negated. Proofs for statements marked (★) can be found in the full version [14].

▶ **Proposition 1 (★).** *For every non-constant monotone Boolean set-function $f\colon 2^{[n]} \to \{0,1\}$ there is a monotone CNF formula ϕ such that for all $x_1, \ldots, x_n \in \{0,1\}^n$ we have $\phi(x_1, \ldots, x_n) = 1$ if and only if $f(\{i \mid x_i = 1\}) = 1$.*

Graphs. We will denote the treewidth of a graph G by $\mathbf{tw}(G)$ and its pathwidth by $\mathbf{pw}(G)$. It is well-known that $\mathbf{pw}(G) \geq \mathbf{tw}(G)$; refer to a textbook for further details [4, §7]. We use the following consequence of the gluing operation.

▶ **Proposition 2 (★).** *Let G and H be t-boundaried graphs that share the same set of boundary vertices $B = \{v_1, \ldots, v_t\}$ but are otherwise vertex-disjoint. Then a vertex set $X \subseteq V(G \oplus H)$ is independent in $G \oplus H$ if and only if $X \cap V(G)$ is independent in G and $X \cap V(H)$ is independent in H.*

3 Characterizing equivalence classes for Independent Set

In this section we derive several tools to analyze the equivalence classes of $\equiv_{\text{IS}}$. For each t-boundaried graph G we define a function that captures the interaction of optimal independent sets with its boundary. These will be useful to reason about the (non)equivalence of pairs of graphs with respect to $\equiv_{\text{IS}}$.

▶ **Definition 3.** Let G be a t-boundaried graph with boundary $B = \{v_1, \ldots, v_t\}$. The function $\mathfrak{s}_G\colon 2^B \to \mathbb{N}$ expresses the size of a maximum independent set in G whose intersection with the boundary is a *subset* of a given set:

$$\mathfrak{s}_G(S) := \max\big\{|X| \,\big|\, X \text{ is an independent set in } G \text{ with } X \cap B \subseteq S\big\}. \tag{2}$$

We will see that equivalence classes can be characterized by the functions $\mathfrak{s}_G$ of the graphs G in that class. The next lemma shows that when gluing two boundaried graphs G and H together, the optimum of the resulting graph $G \oplus H$ can be deduced from $\mathfrak{s}_G$ and $\mathfrak{s}_H$. The identity we prove is reminiscent of the recurrence that is used for JOIN nodes when solving INDEPENDENT SET on graphs of bounded treewidth [4, §7.3.1].

▶ **Lemma 4 (★).** *Let G and H be t-boundaried graphs for some t. The following holds:*

$$\max_{S \subseteq B}\{\mathfrak{s}_G(S) + \mathfrak{s}_H(S) - |S|\} = \text{OPT}_{\text{IS}}(G \oplus H).$$

To relate the equivalence of graphs to properties of the corresponding functions $\mathfrak{s}$, the following indicator graphs will be convenient.

▶ **Definition 5.** Let t be a positive integer and $B = \{v_1, \ldots, v_t\}$. For each $S \subseteq B$ define the t-boundaried *indicator* graph I_S with boundary B as the result of the following process: starting from an edgeless graph with vertex set B, for each $v_i \in B \setminus S$ add vertices u_i, u_i' and the edges $\{v_i, u_i\}, \{v_i, u_i'\}$ to I_S.

Each boundary vertex not in S thus becomes the center of a star with two leaves in I_S, and boundary vertices in S are isolated vertices in I_S. The next proposition shows that maximum independent sets of $F \oplus I_S$ reveal the value of $\mathfrak{s}_F(S)$.

▶ **Proposition 6 (★).** $\text{OPT}_{\text{IS}}(F \oplus I_S) = \mathfrak{s}_F(S) + 2(t - |S|)$ *for all t-boundaried graphs F.*

Using Proposition 6 we can show that the equivalence class of a boundaried graph G with respect to $\equiv_{\text{IS}}$ is completely characterized by the function $\mathfrak{s}_G$.

▶ **Theorem 7 (★).** *Let G and H be two t-boundaried graphs with boundary $B = \{v_1, \ldots, v_t\}$. Then $G \equiv_{\text{IS},t} H$ if and only if there exists a constant $c \in \mathbb{Z}$ such that $\mathfrak{s}_G(S) = \mathfrak{s}_H(S) + c$ for all $S \subseteq B$.*

Theorem 7 shows that two t-boundaried graphs G and H are equivalent under $\equiv_{\text{IS}}$ if the functions $\mathfrak{s}_G$ and $\mathfrak{s}_H$ differ by a fixed constant for all inputs. It will be convenient to eliminate this degree of freedom by normalizing the functions.

▶ **Definition 8.** The *normalized boundary function* of a t-boundaried graph G with boundary B is the function $\mathfrak{s}_G^0 \colon 2^B \to \mathbb{N}$ given by $\mathfrak{s}_G^0(S) := \mathfrak{s}_G(S) - \mathfrak{s}_G(\emptyset)$.

Intuitively, $\mathfrak{s}_G^0(S)$ represents how much larger an independent set can be if we are allowed to use the boundary vertices from S, compared to when we are not allowed to use any boundary vertices in the independent set.

▶ **Corollary 9 (★).** *Let G and H be two t-boundaried graphs with boundary $B = \{v_1, \ldots, v_t\}$. Then $G \equiv_{\text{IS}} H$ if and only if $\mathfrak{s}_G^0 = \mathfrak{s}_H^0$.*

Corollary 9 shows that equivalence classes of $\equiv_{\text{IS}}$ are determined by the normalized boundary functions of the graphs in the class. To see how many different equivalence classes there can be, it is therefore useful to analyze the properties of normalized boundary functions.

▶ **Definition 10.** Let t be a positive integer and let $B := \{v_1, \ldots, v_t\}$. A function $f \colon 2^B \to \mathbb{N}$ is called a *t-representative function* if it satisfies the following three properties:

1. $f(\emptyset) = 0$.
2. *Monotonicity*: for any $S' \subseteq S \subseteq B$ we have $f(S') \leq f(S)$.
3. *Bounded increase*: For every nonempty set $S \subseteq B$ we have $f(S) \leq 1 + \min_{v \in S} f(S \setminus \{v\})$.

▶ **Lemma 11.** *Let G be a t-boundaried graph with boundary $B := \{v_1, \ldots, v_t\}$. Then $\mathfrak{s}_G^0$ is a t-representative function.*

Proof. We prove that $\mathfrak{s}_G^0$ has the three properties given in Definition 10.

1. By definition of $\mathfrak{s}_G^0$ we have $\mathfrak{s}_G^0(\emptyset) = \mathfrak{s}_G(\emptyset) - \mathfrak{s}_G(\emptyset) = 0$.
2. This follows directly from Definitions 3 and 8: the collection of independent sets over which $\mathfrak{s}_G(S')$ optimizes is a subset of the independent sets over which $\mathfrak{s}_G(S)$ optimizes.

3. Consider a nonempty set $S \subseteq B$ and let X be an independent set in G of size $\mathfrak{s}_G(S)$ with $X \cap B \subseteq S$, which exists by Definition 3. For every $v \in S$ we have that $X \setminus \{v\}$ is an independent set of size $|X| - 1$ in G whose intersection with B is a subset of $S \setminus \{v\}$, implying that $\mathfrak{s}_G(S \setminus \{v\}) \geq |X| - 1 = \mathfrak{s}_G(S) - 1$. Adding $1 - \mathfrak{s}_G(\emptyset)$ on both sides we obtain $\mathfrak{s}^0_G(S) = \mathfrak{s}_G(S) - \mathfrak{s}_G(\emptyset) \leq 1 + \mathfrak{s}_G(S \setminus \{v\}) - \mathfrak{s}_G(\emptyset) = 1 + \mathfrak{s}^0_G(S \setminus \{v\})$. As this holds for all $v \in S$, it holds in particular for $v \in S$ minimizing $\mathfrak{s}^0_G(S \setminus \{v\})$. ◀

4 Defining graphs with given boundary characteristics

Corollary 9 shows that t-boundaried graphs with the same normalized boundary function belong to the same equivalence class. Since each normalized boundary function is a t-representative function by Lemma 11, this implies that the number of equivalence classes of $\equiv_{\mathrm{IS},t}$ is *at most* the number of distinct t-representative functions. In Lemma 13 we will show that, surprisingly, the converse also holds: for each t-representative function there is a distinct equivalence class of $\equiv_{\mathrm{IS},t}$. Before proving that lemma, we first derive a useful property of t-representative functions.

▶ **Proposition 12 (★).** *Each t-representative function f satisfies $f(S') - |S' \setminus S| \leq f(S)$ for all $S, S' \subseteq B$.*

▶ **Lemma 13.** *For every t-representative function f, there exists a t-boundaried graph G with boundary $B := \{v_1, v_2, \ldots, v_t\}$, such that $\mathfrak{s}^0_G(S) = f(S)$ for every $S \subseteq B$.*

Proof. Consider an arbitrary t-representative function f, which assigns a non-negative integer to each $S \subseteq B$. We construct a t-boundaried graph G for which $\mathfrak{s}^0_G = f$, as follows:

1. Start from an edgeless graph with vertex set B, which is the boundary of the graph.
2. For each $i \in [t]$ add a vertex u_i and the edge $\{u_i, v_i\}$.
3. For each $S \subseteq B$ with $f(S) > 0$, add a set $V_S = \{v_{S,1}, \ldots, v_{S,f(S)}\}$ consisting of $f(S)$ vertices to the graph. These vertices are false twins (all share the same open neighborhood) and are connected to the rest of the graph as follows:
   a. For each $i \in [t]$ with $v_i \in S$, all vertices of V_S are adjacent to u_i.
   b. For each $i \in [t]$ with $v_i \notin S$, all vertices of V_S are adjacent to v_i.
   c. All vertices of V_S are adjacent to all vertices $V_{S'}$ that are created for sets $S' \neq S$.

We show that $\mathfrak{s}_G(S) = t + f(S)$ for all $S \subseteq B$. This will imply that $\mathfrak{s}^0_G(S) = \mathfrak{s}_G(S) - \mathfrak{s}_G(\emptyset) = (t + f(S)) - (t + f(\emptyset)) = (t + f(S)) - (t + 0) = f(S)$ for all $S \subseteq B$, since $f(\emptyset) = 0$ by Definition 10. We therefore conclude the proof by showing that $\mathfrak{s}_G(S) = t + f(S)$ for all $S \subseteq B$, by establishing two inequalities. Consider an arbitrary $S \subseteq B$.

($\geq$) To show $\mathfrak{s}_G(S) \geq t + f(S)$ we construct an independent set X in G of size $t + f(S)$ that intersects B in a subset of S. If $f(S) = 0$ then $X = \{u_1, \ldots, u_t\}$ suffices, so assume in the remainder that $f(S) > 0$. Let X consist of the $f(S)$ vertices in V_S, together with the vertices $\{u_i \mid i \in [t], v_i \notin S\}$ and $\{v_i \mid i \in [t], v_i \in S\}$. Then $|X| = t + f(S)$, and using the construction above it is straight-forward to verify that X is an independent set. Since $X \cap B = S$, this shows that $\mathfrak{s}_G(S) \geq t + f(S)$.

($\leq$) Now we argue that $\mathfrak{s}_G(S) \leq t + f(S)$. Consider a maximum independent set X in G that intersects B in a subset of S, which has size $\mathfrak{s}_G(S)$ by Definition 3. If X contains no vertices of $V_{S'}$ for any $S' \subseteq B$, then X has at most t vertices: an independent set contains at most one vertex of each edge $\{v_i, u_i\}$ for each $i \in [t]$. Hence $|X| \leq t$ in this case, which is at most $t + f(S)$ since $f(S) \geq 0$ by Properties 1 and 2. In the remainder, assume X contains a vertex of $V_{S'}$ for some $S' \subseteq B$. This implies that X contains no vertices from $V_{S''}$

for any $S'' \neq S'$, since all vertices of $V_{S'}$ are adjacent to all vertices of $V_{S''}$ by construction of G. Hence besides the vertices from $V_{S'}$, the set X only contains vertices of edges $\{v_i, u_i\}$ for $i \in [t]$. The independent set X contains at most one vertex from each such edge. For each $v_i \in S' \setminus S$, observe that X does not contain v_i (since $X \cap B \subseteq S$), and X does not contain u_i either (since u_i is adjacent to all members of $V_{S'}$). So X has at most $f(S')$ vertices from $V_{S'}$, no vertices of $\{v_i, u_i\}$ for each $v_i \in S' \setminus S$, and at most one vertex from each of the remaining $t - |S' \setminus S|$ edges. It follows that $|X| \leq f(S') + (t - |S' \setminus S|)$. By Proposition 12 we have $f(S') - |S' \setminus S| \leq f(S)$, which shows that $|X| \leq t + f(S)$ and concludes the proof. ◀

5 Counting t-representative functions

We say that two t-representative functions are distinct if their function values differ on some input. Lemma 13 shows that for each t-representative function f, there exists a t-boundaried graph whose normalized boundary function equals f. Together with Corollary 9, which says that boundaried graphs with the same normalized boundary function are equivalent under $\equiv_{\text{IS},t}$, this establishes a bijection between the equivalence classes of $\equiv_{\text{IS},t}$ and the t-representative functions. To bound the number of equivalence classes of $\equiv_{\text{IS},t}$ it therefore suffices to bound the number of t-representative functions. Recall that $M(t)$ denotes the t-th *Dedekind number*, the number of distinct monotone Boolean functions of t variables.

▶ **Lemma 14 (★).** *There are at least $M(t) - 1$ distinct t-representative functions.*

It is known that $M(t) \geq 2^{\binom{t}{\lfloor t/2 \rfloor}}$. To see this, consider the subsets $\mathcal{S}_t = \binom{[t]}{\lfloor t/2 \rfloor}$ of $[t]$ of size $\lfloor t/2 \rfloor$. For each subset $\mathcal{S}'_t \subseteq \mathcal{S}_t$ we obtain a different monotone set-function by saying that $f(S) = 1$ if and only if S contains one of the subsets in $\mathcal{S}'_t$. By Stirling's approximation we have $\binom{t}{\lfloor t/2 \rfloor} \geq 2^t/\sqrt{4t}$, which implies that $M(t) \geq 2^{2^t/\sqrt{4t}}$. The following lemma gives an upper bound on the number of t-representative functions.

▶ **Lemma 15 (★).** *The number of distinct t-representative functions is at most 2^{2^t-1}.*

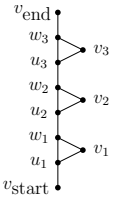
Lemmata 14 and 15 give the following corollary for each positive integer t.

▶ **Corollary 16.** *The number of equivalence classes of $\equiv_{\text{IS},t}$ lies between $2^{2^t/\sqrt{4t}}$ and 2^{2^t-1}.*

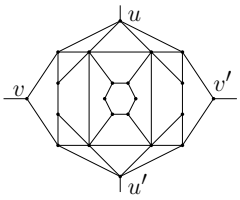
6 Defining planar graphs with given boundary characteristics

In Lemma 13 we constructed nonequivalent t-boundaried graphs based on distinct t-representative functions. The graphs constructed in that lemma have large treewidth and are far from being planar; they contain cliques of size roughly 2^t. To derive lower bounds that are meaningful even when protrusion replacement is applied for planar graphs of bounded treewidth, we present an alternative construction to lower bound the number of equivalence classes that contain a planar graph of small pathwidth (and therefore have small treewidth). The following gadget, of which several variations were used in earlier work (cf. [13, Theorem 5.3] and [9, 16]), will be useful in our construction.

▶ **Definition 17.** Let k be a positive integer. The *clause gadget of size k* is the graph $\mathcal{C}_k$ constructed as follows (see Figure 1a). For each $i \in [k]$ create a triangle on vertices $\{u_i, v_i, w_i\}$. Connect these into a path by adding all edges $\{w_i, u_{i+1}\}$ for $i \in [k-1]$. Finally, add vertices $v_{\text{start}}, v_{\text{end}}$ and the edges $\{v_{\text{start}}, u_1\}$ and $\{w_k, v_{\text{end}}\}$. The vertices $(v_1, \ldots, v_k)$ are the *terminals* of the clause gadget.



(a) Clause gadget $\mathcal{C}_3$



(b) Crossover gadget $G_\times$

j \ i	0	1	2
0	7	8	8
1	8	9	9
2	7	8	9

(c) Values of OPT_{IS} in $G_\times$

Figure 1 Gadgets for Independent Set. The crossover gadget is due to Garey et al. [11, Fig. 11 and Table 1]. The table on the right shows for all relevant combinations of i and j what the maximum size is of an independent set X satisfying $|\{v, v'\} \cap X| = i$ and $|\{u, u'\} \cap X| = j$.

▶ **Observation 18** (Cf. [9, Obs. 6–8]). *For each positive $k \in \mathbb{N}$, the clause gadget $\mathcal{C}_k$ has the following properties:*

1. $\text{OPT}_{\text{IS}}(\mathcal{C}_k) = k + 2$.
2. *Every maximum independent set in $\mathcal{C}_k$ contains a terminal vertex v_i for some $i \in [k]$.*
3. *$\forall i \in [k]$ there is a maximum independent set in $\mathcal{C}_k$ containing v_i but no other terminals.*
4. *$\mathcal{C}_k$ is planar and $\boldsymbol{pw}(\mathcal{C}_k) = 2$.*

To ensure our construction yields a planar graph, we use a *crossover gadget* for Independent Set due to Garey et al. [11]. It was originally designed for Vertex Cover, but since the complement of a maximum independent set is a minimum vertex cover, we can rephrase the properties of the gadget in terms of independent sets. The crossover gadget $G_\times$ is the 22-vertex graph illustrated in Figure 1b, which has four terminals (u, u', v, v'). When we have a drawing of a graph G in which exactly two edges $\{a, b\}, \{c, d\}$ cross in a common point, we can *planarize the crossing* by removing edges $\{a, b\}$ and $\{c, d\}$, introducing a new copy of $G_\times$ at the position of the crossing, and adding the edges $\{a, v\}, \{v', b\}, \{c, u\}, \{u', d\}$. Garey et al. [11] analyzed the size of a maximum independent set in $G_\times$ when restricting which terminal vertices may occur in the set, as shown in Figure 1c. As $G_\times$ is symmetric in both the horizontal and vertical axis, and the table shows that a maximum size independent set size of nine can already be obtained using $i = 1$ of the terminals $\{v, v'\}$ and $j = 1$ of the terminals $\{u, u'\}$, we observe the following.

▶ **Observation 19.** *For any choice of terminals $v^* \in \{v, v'\}$ and $u^* \in \{u, u'\}$ there is a maximum independent set of size nine in $G_\times$ that does not contain v^* or u^*.*

The following proposition summarizes the essential features of a planarization operation.

▶ **Proposition 20 (★).** *Let G be a graph drawn in the plane such that no edge contains a vertex in its interior and no more than two edges cross in any single point. Let G' be the result of planarizing an edge crossing by a crossover gadget. The following holds.*

1. *For every independent set X in G there is an independent set X' in G' of size $|X| + 9$ such that $X' \cap V(G) = X$.*
2. *For every independent set X' in G' there is an independent set X'' in G' with $|X'| = |X''|$ containing exactly nine vertices from $G_\times$ with $X'' \cap V(G) \subseteq X' \cap V(G)$.*
3. *For every independent set X' in G' there is an independent set X in G of size $|X'| - 9$ such that $X \subseteq X' \cap V(G)$.*
4. $\text{OPT}_{\text{IS}}(G') = \text{OPT}_{\text{IS}}(G) + 9$.

In most applications of crossover gadgets, the only important property is that they have a fixed effect on the optimum (Property 4). In our case we also have to ensure that the

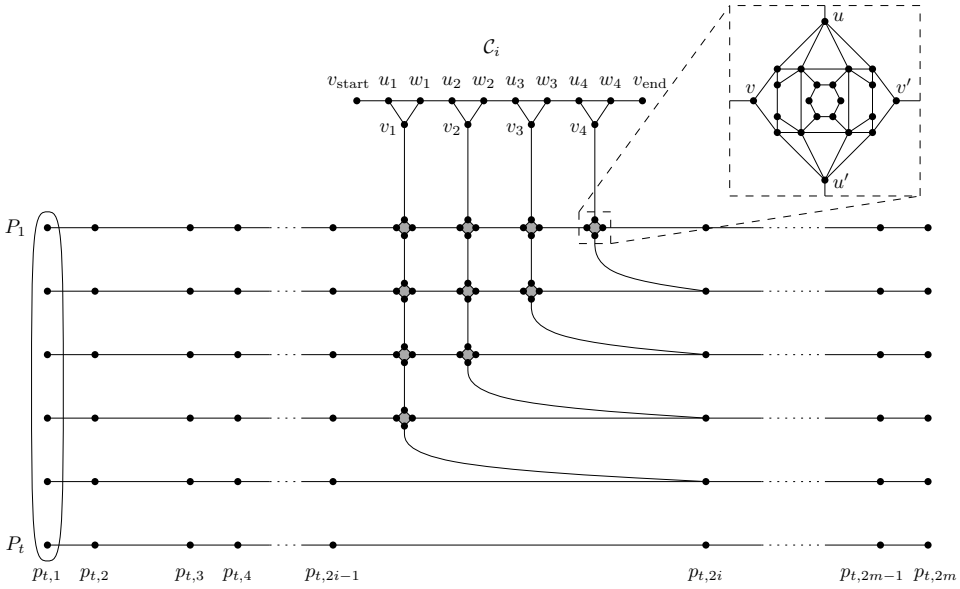


Figure 2 Planarizing the graph G_ϕ to obtain G'_ϕ in the proof of Lemma 21. Only the clause gadget for the clause $C_i = (x_5 \vee x_4 \vee x_3 \vee x_2)$ is shown. Shaded diamonds represent crossover gadgets. The boundary B of the graph is circled, containing the first vertex from each path.

crossover gadgets do not disturb how the solutions intersect the boundary of the graph. Properties 1–3 will be used for this purpose. Using these gadgets we present the construction.

▶ **Lemma 21.** *Let t be a positive integer and $B := \{p_{1,1}, p_{2,1}, \ldots, p_{t-1,1}, p_{t,1}\}$. For every non-constant monotone set-function $f\colon 2^B \to \{0,1\}$ there is a planar graph G with boundary B such that $\boldsymbol{pw}(G) \leq t + \mathcal{O}(1)$ and for every $S \subseteq B$: $f(S) = 1$ if and only if $\mathfrak{s}_G(S) = \text{OPT}_{\text{IS}}(G)$.*

Proof. Consider a monotone set-function f and let ϕ be a monotone CNF formula that represents f in the sense of Proposition 1. Let the clauses of ϕ be $C_1, \ldots, C_m$ such that each clause C_i is a subset of $[t]$ giving the indices of the variables appearing in the clause. Since ϕ is monotone, all variables appear positively. The number of literals in C_i is denoted $|C_i|$.

We first construct a nonplanar graph G_ϕ of small pathwidth such that for all $S \subseteq B$ we have $f(S) = 1$ if and only if $\mathfrak{s}_{G_\phi}(S) = \text{OPT}_{\text{IS}}(G_\phi)$. Then we will use crossover gadgets to turn G_ϕ into a planar graph G'_ϕ while preserving these properties. The construction is inspired by a reduction of Lokshtanov et al. [16, Thm. 3.1], and proceeds as follows.

1. We start by creating t paths $P_1, \ldots, P_t$, where every path P_i for $i \in [t]$ consists of $2m$ vertices $p_{i,1}, \ldots, p_{i,2m}$. The boundary $B = \{p_{1,1}, \ldots, p_{t,1}\}$ of graph G_ϕ contains the first vertex from each path.
2. For each clause $i \in [m]$, add a copy of the clause gadget $\mathcal{C}_{|C_i|}$ to the graph and denote its terminals by $(v_1, \ldots, v_{|C_i|})$. Let $\ell(j)$ denote the j-th variable in the clause for each $j \in [|C_i|]$ and sort these such that $\ell(1) > \ell(2) > \ldots > \ell(|C_i|)$; this will be useful later on when planarizing the graph. For each $j \in [|C_i|]$ make terminal v_j in the clause gadget adjacent to vertex $p_{\ell(j),2i}$ on path $P_{\ell(j)}$. Observe that clause gadgets only connect to *even-numbered* vertices on the paths.

▶ **Claim 22 (★).** *The graph G_ϕ with boundary $B := \{p_{1,1}, \ldots, p_{t,1}\}$ satisfies:*

1. $\mathfrak{s}_{G_\phi}(B) = \text{OPT}_{\text{IS}}(G_\phi) \leq mt + \sum_{1 \leq i \leq m}(|C_i| + 2)$.
2. $\mathfrak{s}_{G_\phi}(B) = \text{OPT}_{\text{IS}}(G_\phi) = mt + \sum_{1 \leq i \leq m}(|C_i| + 2)$.
3. *For each $S \subseteq B$ we have $f(S) = 1$ if and only if $\mathfrak{s}_{G_\phi}(S) = \text{OPT}_{\text{IS}}(G_\phi)$.*

Claim 22 shows that the boundary function of G_ϕ expresses the monotone Boolean function f. The same argumentation as used by Lokshtanov et al. [16, Lemma 3.3] shows that G_ϕ has pathwidth $t + \mathcal{O}(1)$. However, we will not prove this here for the non-planar graph G_ϕ; we will prove a pathwidth bound after planarizing the graph. The planarization starts from a drawing of G_ϕ in the plane in which the crossings have a fixed structure. This drawing is defined as follows (see Figure 2):

- Draw each path $P_1, \ldots, P_t$ horizontally. Place the paths above each other so that P_1 is the highest and P_t is the lowest.
- For each clause $i \in [m]$ of ϕ, draw the clause gadget in a planar fashion above the paths, so that its terminals stick out at the bottom, the lowest-indexed terminal on the left and the highest-numbered terminal on the right. Draw the gadget for clause i between the vertical lines containing the $2i-1$-th and the $2i$-th vertices on each path. Consider the set of edges E_{C_i} connecting the gadget for clause C_i to the vertices of the paths. By construction of G_ϕ, the gadget only connects to vertices with index $2i$ on the paths. Draw the edges from E_{C_i} in such a way that $e \in E_{C_i}$ only crosses the edges between the vertices $p_{j,2i-1}$ and $p_{j,2i}$ of the paths P_j for $j \in [t]$, and do not cross any other edge $e' \in E_{C_i}$. Since the left-to-right order of the variables in a clause matches the order in which the paths are laid out from top to bottom, this is possible.

Based on this drawing we planarize the graph G_ϕ by repeatedly replacing crossings by crossover gadgets, resulting in a planar graph G'_ϕ as shown in Figure 2. Let N denote the number of crossover gadgets which were introduced during the planarization process. By Proposition 20 we know that $\textsc{opt}_{\textsc{is}}(G'_\phi) = \textsc{opt}_{\textsc{is}}(G_\phi) + 9N = mt + 9N + \sum_{1 \leq i \leq m}(|C_i| + 2)$, where we use Property 2 of Claim 22 for the second equality. To conclude the proof, it remains to show that $\mathbf{pw}(G'_\phi) \leq t + \mathcal{O}(1)$ (Claim 24) and that for all subsets $S \subseteq B$ we have $f(S) = 1$ if and only if $\mathfrak{s}_{G'_\phi}(S) = \textsc{opt}_{\textsc{is}}(G'_\phi)$ (Claim 23).

▶ **Claim 23 (★).** *For every $S \subseteq B$ we have $f(S) = 1$ if and only if $\mathfrak{s}_{G'_\phi}(S) = \textsc{opt}_{\textsc{is}}(G'_\phi)$.*

▶ **Claim 24 (★).** *The graph G'_ϕ has pathwidth $t + \mathcal{O}(1)$.*

This concludes the proof of Lemma 21. ◀

7 Lower bound for protrusion replacement

To leverage the construction of Lemma 21 into a lower bound on the critical size of a set of representatives, we need the following lemma. Observe that its second condition shows that no pair of graphs from the constructed set $\mathcal{G}$ is equivalent under $\equiv_{\textsc{is},t}$, and this is witnessed already by gluing planar graphs of pathwidth one onto them. This implies that in any protrusion reduction scheme applied to planar graphs that aims to replace occurrences of bounded-pathwidth protrusions by representatives, there should be a distinct representative for each graph in $\mathcal{G}$.

▶ **Lemma 25 (★).** *For each positive integer t there is a set $\mathcal{G}$ of $M(t)-2$ distinct t-boundaried planar graphs of pathwidth $t + \mathcal{O}(1)$, such that for each pair of distinct graphs $G_f, G_{f'} \in \mathcal{G}$ there are two indicator graphs I_S and I_B as in Definition 5 such that:*

1. *The graphs $G_f \oplus I_S, G_f \oplus I_B, G_{f'} \oplus I_S$, $G_{f'} \oplus I_B$ are planar and have pathwidth $t + \mathcal{O}(1)$.*
2. $\textsc{opt}_{\textsc{is}}(G_f \oplus I_S) - \textsc{opt}_{\textsc{is}}(G_{f'} \oplus I_S) \neq \textsc{opt}_{\textsc{is}}(G_f \oplus I_B) - \textsc{opt}_{\textsc{is}}(G_{f'} \oplus I_B)$.

Finally, we can combine our lower bound on the number of distinct equivalence classes of Lemma 25 with an upper bound on the number of small graphs to obtain our main result.

▶ **Theorem 26 (★).** *Let $t \geq t_0$ be a sufficiently large positive integer. Let $\mathcal{R}_t$ be a set of t-boundaried planar graphs such that every equivalence class of $\equiv_{\text{IS},t}$ that contains a planar graph of pathwidth $t + \mathcal{O}(1)$ is represented by some graph in $\mathcal{R}_t$. Then $\mathcal{R}_t$ contains a graph with $\Omega(\log M(t)) \geq \Omega(2^t/\sqrt{4t})$ vertices.*

8 Conclusion

We presented lower and upper bounds on the number of equivalence classes of the canonical equivalence relation $\equiv_{\text{IS},t}$ for Independent Set on t-boundaried graphs. We combined these lower bounds with upper bounds on the number of small graphs to give lower bounds for the critical sizes of sets of representatives. For a set of *planar* representatives that represent all equivalence classes containing a bounded-pathwidth planar graph, we gave a lower bound of $\Omega(\log M(t)) \geq \Omega(2^t/\sqrt{4t})$ on the critical size. The same argumentation can also be used to obtain lower bounds on the critical size of sets of potentially *nonplanar* representatives. The number of distinct t-boundaried (unrestrained) graphs is at most $2^{\binom{n}{2}} \cdot \binom{n}{t} \leq 2^{n^2/2}$. Using this bound in the proof of Theorem 26 yields a lower bound of $\Omega(\sqrt{\log M(t)}) \geq \Omega(2^{t/2}/\sqrt[4]{4t})$ on the critical size of a set of representatives that contains at least $M(t) - 2$ distinct graphs.

In their work, Garnero et al. [12] (roughly) show that each equivalence class of $\equiv_{\text{IS},t}$ containing a planar graph of treewidth at most t can be represented by a planar graph with $2^{(t+1)^{2^t}}$ vertices and treewidth at most t. Our lower bound shows that to represent all equivalence classes containing a planar graph of *pathwidth* $t + \mathcal{O}(1)$ (a subset of the graphs of treewidth $t + \mathcal{O}(1)$), requires a graph with $\Omega(2^t/\sqrt{4t})$ vertices. Our single-exponential lower bound is very far from the triple-exponential upper bound. However, we believe that the correct bound is single-exponential. Since Corollary 9 shows that each equivalence class is completely characterized by its normalized boundary function, and the construction of Lemma 13 produces a boundaried graph with $2^{\mathcal{O}(t)}$ vertices for any given boundary function, it follows that every equivalence class of $\equiv_{\text{IS},t}$ has a representative with $2^{\mathcal{O}(t)}$ vertices. Note, however, that the representatives constructed in this way are nonplanar and have pathwidth and treewidth $2^{\Theta(t)}$.

The main conceptual contribution of this work is the fact that nontrivial lower bounds can be obtained by counting equivalence classes. The fact that a significant portion of the equivalence classes (at least $M(t) \geq 2^{2^t/\sqrt{4t}}$ out of the total of at most 2^{2^t}) can be generated from monotone Boolean functions was useful in the construction of nonequivalent planar graphs of bounded pathwidth. We showed that the lower bound construction of Lokshtanov et al. [16] can be planarized while increasing the pathwidth by an additive constant. The planarization argument employed here can also be used to strengthen the SETH-based runtime lower bound of $\Omega((2-\varepsilon)^w \cdot n^{\mathcal{O}(1)})$ for solving Independent Set on graphs of treewidth w, to *planar* graphs of treewidth w. Not all bounded-pathwidth graphs can be planarized with a bounded increase in pathwidth. In particular, when planarizing $K_{3,n}$ for sufficiently large n the pathwidth grows arbitrarily large [6].

The lower bounds for Independent Set given in Theorem 26 carry over to the Dominating Set problem, for which protrusion replacement is used frequently. In the full version [14] we describe this extension, which is based on the folklore planarity-preserving NP-completeness reduction from Vertex Cover to Dominating Set.

Acknowledgments. We are grateful to Daniel Lokshtanov and David Eppstein for insightful discussions regarding planarization, and to an anonymous referee of IPEC 2016 for suggesting a simplification in the proof of Theorem 26.

References

1. Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. In *Proc. 50th FOCS*, pages 629–638. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.46.
2. Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. *CoRR*, 2013. arXiv:0904.0727.
3. Hans L. Bodlaender and Babette van Antwerpen-de Fluiter. Reduction algorithms for graphs of small treewidth. *Inf. Comput.*, 167(2):86–119, 2001. doi:10.1006/inco.2000.2958.
4. 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.
5. Babette de Fluiter. *Algorithms for Graphs of Small Treewidth*. PhD thesis, Utrecht University, 1997.
6. David Eppstein. Pathwidth of planarized drawing of $K_{3,n}$. TheoryCS StackExchange question, 2016. URL: http://cstheory.stackexchange.com/questions/35974/.
7. Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting forbidden minors: Approximation and kernelization. *SIAM J. Discrete Math.*, 30(1):383–410, 2016. doi:10.1137/140997889.
8. Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar $\mathcal{F}$-deletion: Approximation, kernelization and optimal FPT algorithms. In *Proc. 53rd FOCS*, pages 470–479. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.62.
9. Fedor V. Fomin and Torstein J. F. Strømme. Vertex cover structural parameterization revisited. *CoRR*, 2016. arXiv:1603.00770.
10. Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sanchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. In *Proc. 21st ESA*, pages 529–540. Springer, 2013. doi:10.1007/978-3-642-40450-4_45.
11. M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.
12. Valentin Garnero, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. Explicit linear kernels via dynamic programming. *SIAM J. Discrete Math.*, 29(4):1864–1894, 2015. doi:10.1137/140968975.
13. Bart M. P. Jansen. *The Power of Data Reduction: Kernels for Fundamental Graph Problems*. PhD thesis, Utrecht University, The Netherlands, 2013. URL: http://igitur-archive.library.uu.nl/dissertations/2013-0612-200803/UUindex.html.
14. Bart M. P. Jansen and Jules J. H. M. Wulms. Lower bounds for protrusion replacement by counting equivalence classes. *CoRR*, 2016. arXiv:1609.09304.
15. Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Trans. Algorithms*, 12(2):21, 2016. doi:10.1145/2797140.
16. Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In *Proc. 22nd SODA*, pages 777–789. SIAM, 2011. doi:10.1137/1.9781611973082.61.

Treedepth Parameterized by Vertex Cover Number

Yasuaki Kobayashi[*1] **and Hisao Tamaki**[2]

1 **Kyoto University, Kyoto, Japan**
kobayashi@iip.ist.i.kyoto-u.ac.jp
2 **Meiji University, Kanagawa, Japan**
tamaki@cs.meiji.ac.jp

Abstract

To solve hard graph problems from the parameterized perspective, structural parameters have commonly been used. In particular, *vertex cover number* is frequently used in this context. In this paper, we study the problem of computing the treedepth of a given graph G. We show that there are an $O(\tau(G)^3)$ vertex kernel and an $O(4^{\tau(G)}\tau(G)n)$ time fixed-parameter algorithm for this problem, where $\tau(G)$ is the size of a minimum vertex cover of G and n is the number of vertices of G.

1998 ACM Subject Classification G.2.2 Graph Algorithms

Keywords and phrases Fixed-parameter algorithm, Polynomial kernelization, Structural parameterization, Treedepth, Vertex cover

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.18

1 Introduction

Treedepth is an important graph invariant which attracts a lot of attentions in several communities. One of the most important results related to treedepth is the work of Nešetřil and Ossona de Mendez [27, 28]. Roughly speaking, they showed that graphs in a sparse graph classes, called graphs of bounded expansion, can be decomposed into graphs of bounded treedepth. An important consequence of this result is a linear time algorithm for deciding first-order logic properties in graphs of bounded expansion [13].

The treedepth $\mathrm{td}(G)$ of an undirected graph G is defined to be the minimum height of a rooted tree T such that G can be embedded into T in such a way that the end vertices of each edge in G has an ancestor-descendant relationship in T. A formal definition of treedepth is given in Section 2. Treedepth has been studied in literature with different names such as vertex ranking numbers [4] and the minimum height of elimination trees [29]. This invariant has applications in solving linear systems [25] and VLSI layouts [24, 31]. Moreover, treedepth has a deep relation with other well-known graph invariants *treewidth* and *pathwidth*. Let $\mathrm{tw}(G)$ and $\mathrm{pw}(G)$ be the treewidth and the pathwidth of a graph G, respectively. Then, we have $\mathrm{tw}(G) \leq \mathrm{pw}(G) \leq \mathrm{td}(G) - 1 = O(\mathrm{tw}(G) \log n)$ [28], where n is the number of vertices of G. When these three invariants are bounded, we can develop efficient algorithms for many graph problems. More precisely, if the one of the above three invariants of the input graph is at most k, many NP-hard graph problems are *fixed-parameter tractable*, that is, there is an algorithm (called a *fixed-parameter algorithm*) that solves the target problem

* Y. K. was supported by JST, CREST.

in $f(k)n^{O(1)}$ time, where n is the number of vertices and the function f depends only on k (see, for example [3]). On the other hand, Gutin *et al.* [21] recently showed that the mixed Chinese postman problem is W[1]-hard parameterized by the pathwidth of the input graph and is fixed-parameter tractable parameterized by its treedepth.

Based on the above facts, it is natural to seek an efficient algorithm for computing the treedepth of graphs. Unfortunately, the problem of computing treedepth is known to be NP-hard [29] even when the input graphs are restricted to bipartite graphs [4] or chordal graphs [11]. On the other hand, there are polynomial time algorithms for some classes of graphs [1, 10]. This problem is studied from the perspective of parameterized complexity. In this context, we are asked whether the treedepth of the input graph is at most k. Since treedepth is monotone under taking minor operations, this problem is fixed-parameter tractable when k is given as a parameter. This follows from the celebrated work of Robertson and Seymour which proves, for every minor closed graph class $\mathcal{G}$, there is a fixed-parameter algorithm that deciding whether the input graph belongs to $\mathcal{G}$ in $f(\mathcal{G})n^{O(1)}$ time. Recently, Reidl *et al.* [30] gave an algorithm for the problem whose running time is $2^{O(k^2)}n$. Their algorithm in fact runs in time $2^{O(kt)}n$, where t is the treewidth of the input graph.

As we have already noted, many graph problems can be efficiently solved on graphs of bounded treewidth. However, there are some exceptions. For example, the Bandwidth problem is known to be NP-hard for graphs of pathwidth at most three [26]. Motivated by this hardness result [26], Fellows *et al.* [16] showed that the Bandwidth problem is fixed-parameter tractable when parameterized by the size $\tau(G)$ of a *minimum vertex cover* of the input graph G. Since, for every graph G with n vertices, $\mathrm{tw}(G) \leq \tau(G) = O(\mathrm{tw}(G) \log n)$, the parameterization of the size of a minimum vertex cover to the input graph is more restricted than that of treewidth. Therefore, this parameterization were often used to develop algorithms for various hard graph problems [9, 15, 18]. In particular, Chapelle *et al.* [8] gave algorithms for computing treewidth and pathwidth whose running time is $3^{\tau(G)}n^{O(1)}$. Although this parameterization is rather restrictive, their algorithms improve the running time of the best known exact exponential algorithms for treewidth [19, 20] and for pathwidth [23] on bipartite graphs. Moreover, this parameterization is used in the context of kernelizations. Here, a *kernelization* is a polynomial time algorithm that, given a pair of an instance I and a parameter k, computes a pair (I', k') such that (I, k) is a YES-instance if and only if (I', k') is a YES-instance for the same problem and the size of I' and k' are upper bounded by some function in k. An output of a kernelization is called a *kernel*. A kernelization is *polynomial* if the size of I' is upper bounded by a polynomial in k. Bodlaender *et al.* [7, 6] showed that the Treewidth and Pathwidth problem admit polynomial kernelizations when parameterized by the size of a minimum vertex cover, in contrast to the lower bound results of polynomial kernelizations [5, 12]. when parameterized by the solution size. Subsequently, Jansen [22] improved the size of kernel to $O(|\tau(G)|^2)$ for treewidth.

In this paper, we give counterparts for treedepth to the results of kernelizations [7, 6] and fixed-parameter algorithms [8] for treewidth and pathwidth. Our results are as follows.

▶ **Theorem 1.** *There is a polynomial time algorithm that, given a graph G, a vertex cover C of G, and an integer k, computes a graph H with $|V(H)| = O(|C|^3)$ such that the treedepth of G is at most k if and only if that of H is at most k. Moreover, $V(H) \cap C$ is a vertex cover of H.*

▶ **Theorem 2.** *The treedepth of G can be computed in $O(4^{\tau(G)}\tau(G)n)$ time, where $\tau(G)$ is the size of a minimum vertex cover of G and n is the number of vertices of G.*

Theorem 1 implies together with a well-known 2-approximation algorithm for vertex cover that the treedepth problem admits a kernel with $O(\tau(G)^3)$ vertices. Let us note that, for the problem deciding whether $\mathrm{td}(G) \leq k$ with parameter k, there is no polynomial kernelization [5, 12] unless NP $\subseteq$ coNP/poly and the running time of the fastest known fixed-parameter algorithm is $2^{O(k^2)}n$ time [30]. Our kernelization and algorithm are useful for the case $\tau(G) = td(G)^{O(1)}$ and $\tau(G) = o(\mathrm{td}(G)^2)$, respectively. In contrast to treewidth and pathwidth, our result does not improve the running time of the best known exact exponential algorithm for treedepth [17] even on bipartite graphs. However, we believe that our approach is relevant for improving the bipartite case.

The technique behind our algorithm in Theorem 2 is as follows. We are given a graph G and a vertex cover C of G. Our algorithm constructs an *optimal elimination tree*, defined in Section 2, of G by a bottom-up dynamic programming. To this end, we need to define subproblems. The first attempt to define subproblems is that for each $X \subseteq C$, construct an optimal elimination tree of an induced subgraph H of G with $V(H) \cap C = X$. However, this strategy does not work since we cannot know which vertex in the independent set $V(G) \setminus C$ is in $V(H) \setminus X$. The second attempt is that for each $X \subseteq C$ and each $P \subseteq C \setminus X$, construct an optimal elimination tree of H such that $V(H) \cap C = X$ and every vertex in P is committed to be an ancestor of the vertices of H in an optimal elimination tree of the whole graph G. Using this pair X and P, we can identify the vertices of H and therefore a subproblem on (X, P) for each $X \subseteq C$ and $P \subseteq C \setminus X$ is well-defined. For each subproblem (X, P), the remaining task is to compute an optimal elimination tree T of a graph corresponding to (X, P) from optimal elimination trees of graphs corresponding to smaller subproblems (X', P') for $X' \subset X$ and $P' \subseteq C \setminus X'$. To do this, we will exploit some nontrivial property of an optimal elimination tree (See Section 4).

This paper is organized as follows. The next section describes some notations and terminologies we use. In Section 3, we design a polynomial kernelization for proving Theorem 1. In Section 4, we show Theorem 2 by giving a fixed-parameter algorithm for treedepth. Finally, in Section 5, we conclude this paper.

2 Preliminaries

For an undirected graph G, $V(G)$ denotes the set of vertices of G and $E(G)$ denotes the set of edges of G. For $v \in V(G)$, the set of neighbors of v is denoted by $N_G(v)$. For $X \subseteq V(G)$, we set $N_G(X) = \bigcup_{x \in X} N_G(x) \setminus X$. We may drop the reference to G when it is clear from the context. For disjoint sets $X, Y \subseteq V(G)$, we use $E(X, Y)$ to denote the set of all edges with one end in X and the other end in Y. The induced subgraph by $X \subseteq V(G)$ of G is denoted by $G[X]$. For two graphs G and H, $H \subseteq G$ means that H is a subgraph of G.

Let T be a rooted tree. For $v \in V(T)$, the subtree rooted at v is denoted by T_v and the unique path between v and the root of T is denoted by P_v. A *branching point* of T is a vertex that has at least two children in T. For two vertices $u, v \in V(T)$, we say u is an *ancestor* of v (v is a *descendant* of u) if $u \in V(P_v)$. A set of disjoint rooted trees is called a *rooted forest*. For a rooted forest F, we use $V(F)$ to denote $\bigcup_{T \in F} V(T)$. The *depth* of v in T is defined by the number of vertices of P_v. The *height* of T is the maximum depth among the vertices in T and the height of rooted forest F is the maximum depth of a rooted tree that belongs to F. The height of rooted tree T and rooted forest F are denoted by $\mathrm{height}(T)$ and $\mathrm{height}(F)$, respectively.

Let F be a rooted forest. The *closure* of F is a graph with vertex set $V(F)$ such that the graph contains an edge $\{u, v\}$ if and only if u is an ancestor of v in F or vice versa. We

denote by $\mathrm{clos}(F)$ the closure of F. In particular, when F consists of a single rooted tree T, we may write $\mathrm{clos}(T)$ instead of $\mathrm{clos}(\{T\})$. For a (not necessary connected) graph G, the *treedepth* $\mathrm{td}(G)$ of G is the minimum integer k such that there is a rooted forest F with $\mathrm{height}(F) = k$ and $G \subseteq \mathrm{clos}(F)$.

As mentioned before, the notion of treedepth has equivalent definitions. In this paper, we frequently use the notion of *elimination trees*. For a rooted forest F and a vertex v not in $V(F)$, we use the notation $F \circ v$ to denote the rooted tree with root v obtained from F by adding an edge between v and the root of T for each $T \in F$. In particular, if F consists of a single rooted tree T, we write $T \circ v$ instead of $\{T\} \circ v$.

▶ **Definition 3.** An *elimination tree* of a connected graph G is recursively defined as follows.

1. If G consists of a single vertex then the elimination tree of G is itself.
2. Otherwise, choose $v \in V(G)$ arbitrary. Let $F = \{T_1, T_2, \ldots, T_t\}$ be elimination trees of the connected components of $G[V(G) \setminus \{v\}]$. Then, $F \circ v$ is an elimination tree of G.

We say that an elimination tree T of G is *optimal* if there is no elimination trees of G whose height is smaller than the height of T. It is easy to see that, for every elimination tree T of G, the closure of T contains G as a subgraph. The following proposition shows that the converse also holds when G is connected.

▶ **Proposition 4** ([27]). *Let G be a connected graph and let T be a rooted tree. Then, T is an elimination tree of G if and only if $G \subseteq \mathrm{clos}(T)$. In particular, the height of an optimal elimination tree of G is $\mathrm{td}(G)$.*

When we refer to an elimination tree T of G, we use the fact $G \subseteq \mathrm{clos}(T)$ without the reference to Proposition 4 and vice versa.

3 Kernelization

The aim of this section is to develop a polynomial kernelization for treedepth for proving Theorem 1. The technique that we use is very similar to the kernelization result for pathwidth [6].

Fix a vertex cover C of G. Let $I = V(G) \setminus C$ and let k be a positive integer. We consider the problem of deciding whether $\mathrm{td}(G) \leq k$. We assume that C is not empty since otherwise the problem is trivial.

We will describle three reduction rules. The following rule trivially does not change the treedepth of G.

▶ **Rule 5.** *Let $u \in I$ be an isolated vertex. Then, delete u from G.*

▶ **Lemma 6.** *Suppose $\mathrm{td}(G) \leq k$. Let u, v be vertices with $|N_G(u) \cap N_G(v)| \geq k$ and let G' be the graph obtained by adding an edge $\{u, v\}$ to G. Then, $\mathrm{td}(G) = \mathrm{td}(G')$.*

Proof. Since G is a subgraph of G', $\mathrm{td}(G) \leq \mathrm{td}(G')$. For the inverse direction, let T be an optimal elimination tree of G. When T_u and T_v are not disjoint, T is also an elimination tree of G' and hence $\mathrm{td}(G) \geq \mathrm{td}(G')$. Here, we assume otherwise. This assumption implies the vertices $N_G(u) \cap N_G(v)$ are common ancestors of u and v. This contradicts the fact that $\mathrm{height}(T) \leq k$ and $|N_G(u) \cap N_G(v)| \geq k$. ◀

This lemma verifies the following rule does not change the treedepth of G.

▶ **Rule 7.** *Let u, v be vertices with $|N(u) \cap N(v)| \geq k$. Suppose at least one of $u \in C$ and $v \in C$ holds. Then, add an edge $\{u, v\}$ to G.*

Let us note that C is also a vertex cover of the result of an application of Rule 7 to G.

For the next rule, we need the following observation, which is clear from the definition of closure.

▶ **Observation 8.** *Let K be a clique in G and let T be an elimination tree of G. Then, there is $v \in K$ with $K \subseteq V(P_v)$.*

A vertex v is *simplicial* in G if $N_G(v)$ forms a clique.

▶ **Lemma 9.** *Suppose* $\mathrm{td}(G) \leq k$. *Let u be a simplicial vertex such that $N(u)$ is not empty and $|N(v)| \geq k+1$ for each $v \in N(u)$. Then* $\mathrm{td}(G) = \mathrm{td}(G[V(G) \setminus \{u\}])$.

Proof. The subgraph relation proves $\mathrm{td}(G) \geq \mathrm{td}(G[V(G) \setminus \{u\}])$. In the following, we show the converse inequality. Let T be an optimal elimination tree of $G[V(G) \setminus \{u\}]$. Since $N(u)$ forms a clique in $G[V(G) \setminus \{u\}]$, by Observation 8, there is $v \in N(u)$ with $N(u) \subseteq V(P_v)$. From the assumption of this lemma, v has at least k neighbors different from u. Since $\mathrm{height}(T) \leq k$, at least one of them is a descendant of v in T. Observe that a rooted tree T' obtained from T by adding u as a child of v is an elimination tree of G. This follows from $N(u) \subseteq V(P_v)$. Moreover, since v is not a leaf in T, $\mathrm{height}(T) = \mathrm{height}(T')$ and hence $\mathrm{td}(G) \leq \mathrm{td}(G[V(G) \setminus \{u\}])$ holds. ◀

▶ **Rule 10.** *Let $u \in I$ be a simplicial vertex of G. Suppose each $v \in N(u)$ has at least $k+1$ neighbors. Then, delete u from G.*

The above two rules give us a small kernel for treedepth.

▶ **Lemma 11.** *Suppose* $\mathrm{td}(G) \leq k$, $|C| \geq k$, *and neither Rule 5, Rule 7, nor Rule 10 are applicable to G. Then, the number of vertices of G is $O(|C|^3)$.*

Proof. Let S be the set of simplicial vertices of G, let $P = S \cap I$, and let $Q = I \setminus P$. By Rule 5, each vertex in P has at least one neighbor in C. For each $u \in P$, by Rule 10, there is a vertex $v \in N(u)$ whose degree is at most k. We associate u with $v \in C$. Each vertex in P is associated with some vertex in C and at most k vertices in P are associated with each vertex in C. Hence, $|P| \leq k \cdot |C| \leq |C|^2$. Next, consider non-adjacent vertices u, v of C. Observe that u and v have at most $k-1$ common neighbors in G. This follows from Rule 7. As each vertex in Q has at least one pair of non-adjacent vertices in C, we have $|Q| \leq (k-1) \cdot |C|(|C|-1)/2 \leq |C|^3/2$. Therefore, $V(G) = |C| + |P| + |Q| \leq 3 \cdot |C|^3$. ◀

Let (G, C, k) be an instance of our decision problem. Suppose $|C| < k$. Obviously, (G, C, k) is a YES-instance. In this case, we output a constant-sized YES-instance. Suppose otherwise. We exhaustively apply Rule 7 and Rule 10 to G until both of them are not applicable to G. By Lemmas 6 and 9, the resulting graph H satisfies $\mathrm{td}(H) \leq k$ if and only if $\mathrm{td}(G) \leq k$. Moreover, $V(H) \cap C$ is a vertex cover of H. Thus, $(H, V(H) \cap C, k)$ is a valid instance. By Lemma 11, if $|V(H)| > 3 \cdot |V(H) \cap C|^3$, then (G, C, k) is a NO-instance. In this case, we output a constant-sized NO-instance. Overall, we have Theorem 1.

4 Fixed-Parameter Algorithm

The aim of this section is to develop an algorithm computing the treedepth of G whose running time is upper bounded by $O(4^{\tau(G)}\tau(G)n)$, where $n = |V(G)|$. First, we use the following algorithm to obtain a minimum vertex cover in advance.

▶ **Proposition 12** (folklore). *There is a $O(2^{\tau(G)}(n+m))$ time algorithm that finds a minimum vertex cover of a graph G, where n and m are respectively the number of vertices and edges of G.*

Note that $m = O(n \cdot \tau(G))$. In the rest of this section, fix a vertex cover C of G. We assume that C is not empty since otherwise the problem is trivial. Let $I = V(G) \setminus C$. For $X \subseteq C$, we set $I(X) = \{v \in I \mid N(v) \subseteq X\}$. We say that a rooted tree is *atomic* if it consists of a single vertex.

▶ **Definition 13.** For $X \subseteq C$ and $P \subseteq C \setminus X$, we say that a rooted forest F is *compatible* with (X, P) if the following three conditions are satisfied:
C1. $V(F) = X \cup I(X \cup P)$,
C2. $G[V(F)] \subseteq \mathrm{clos}(F)$, and
C3. every vertex in $I(P)$ forms an atomic rooted tree in F.

Note that a rooted forest that is compatible with (X, P) does not contain any vertex in P and contains every vertex in $I(P)$. We denote by $\mathrm{td}(X, P)$ the minimum height over all rooted forests that are compatible with (X, P). We say that F is *optimal for* (X, P) if F is compatible with (X, P) and $\mathrm{height}(F) = \mathrm{td}(X, P)$. Also, note that if $G[X \cup I(X \cup P)]$ is connected, there is a rooted tree T that is optimal for (X, P).

In what follows, we will give recurrences for computing $\mathrm{td}(X, P)$ for $X \subseteq C$ and $P \subseteq C \setminus X$. The algorithm evaluates those recurrences by a straight forward dynamic programming. The following lemma is the base case of our recurrences and is easy to verify.

▶ **Lemma 14.** *Let $P \subseteq C$. If $I(P)$ is not empty, then $\mathrm{td}(\emptyset, P) = 1$. Otherwise, $\mathrm{td}(\emptyset, P) = 0$.*

Let $X \subseteq C$ and $P \subseteq C \setminus X$. From now on, we consider the case $X \neq \emptyset$.

▶ **Lemma 15.** *Let $x \in X$ be arbitrary and let F be a rooted forest that is compatible with $(X \setminus \{x\}, P \cup \{x\})$. Then the rooted forest $F' := I(P) \cup (F \setminus I(P)) \circ x$ is compatible with (X, P). Here and in similar situations later, $I(P)$ is also interpreted as the set of atomic rooted trees.*

Proof. Clearly, F' satisfies condition C3 for (X, P). As $V(F) \setminus V(F') = \{x\}$, F' satisfies condition C1 for (X, P). Let $T = (F \setminus I(P)) \circ x$. Since every vertex in $N_{G[V(T)]}(x)$ is a descendant of x in T, T is an elimination tree of $G[V(T)]$. As $X \cap P = \emptyset$, we have $E(X, I(P)) = \emptyset$. Thus, condition C2 holds for (X, P) and hence the lemma follows. ◀

We say that a bipartition (Y, Z) of X is *separated* if neither Y nor Z is empty and $E(Y, Z) = \emptyset$. The following observation is easy to verify.

▶ **Observation 16.** *Let (Y, Z) be a bipartition of X. Then, $I(X \cup P)$ is partitioned into $N(Y) \cap N(Z) \cap I(X \cup P)$, $I(Y \cup P) \setminus I(P)$, $I(Z \cup P) \setminus I(P)$, and $I(P)$.*

Let (Y, Z) be a separated bipartition of X. We define the rooted forest $F_{Y,Z}$ from F_Y and F_Z, where F_Y and F_Z are rooted forests that are compatible with (Y, P) and (Z, P), respectively, as follows. Let $v_0, v_1, \ldots v_p$ be the vertices in $N(Y) \cap N(Z) \cap I(X \cup P)$ with arbitrary order. Note that F_Y and F_Z may share the set of vertices $I(P)$. Set $F_0 := (F_Y \cup F_Z) \setminus I(P)$ and, for each $0 \leq i \leq p$, set $F_{i+1} := F_i \circ v_i$. Finally, set $F_{Y,Z} := F_{p+1} \cup I(P)$. By the above definition, if $N(Y) \cap N(Z) \cap I(X \cup P)$ is empty, $F_{Y,Z}$ is indeed $F_Y \cup F_Z$.

▶ **Lemma 17.** *Let (Y, Z) be a separated bipartition of X and let F_Y and F_Z be rooted forests that are compatible with (Y, P) and (Z, P), respectively. Then, $F_{Y,Z}$ is compatible with (X, P).*

Proof. We show that $F_{Y,Z}$ satisfies the three conditions of compatibility with (X, P).

Clearly, $F_{Y,Z}$ satisfies condition C3. By the construction of G, $V(F_{Y,Z})$ is composed of $(V(F_Y) \cup V(F_Z)) \setminus I(P)$, $N(Y) \cap N(Z) \cap I(X \cup P)$, and $I(P)$. Since $V(F_Y) \cap I = I(Y \cup P)$ and $V(F_Z) \cap I = I(Z \cup P)$, by Observation 16, we have $V(F_{Y,Z}) \cap I = I(X \cup P)$. Recall that $Y \cup Z = X$. Thus, we have $V(F_{Y,Z}) = X \cup I(X \cup P)$, that is, $F_{Y,Z}$ satisfies condition C1. To show the condition C2, it is enough to show that for each edge $\{u, v\}$ in $G[X \cup I(X \cup P)]$, u is an ancestor of v or vice versa in $F_{Y,Z}$. Since X is a vertex cover of $G[X \cup I(X \cup P)]$, at least one of u and v is in X. Assume without loss of generality u is in Y. Since F_Y is compatible with (Y, P), if $\{u, v\} \subseteq V(F_Y)$, we are done. Otherwise, as $u \notin V(F_Z)$, u must be in $N(Y) \cap N(Z) \cap I(X \cup P)$, and hence u is an ancestor of v. This finishes the proof. ◀

Let T be a rooted tree that is compatible with (X, P) and let $v \in V(T)$ be a maximum depth vertex such that every vertex in $V(P_v) \setminus \{v\}$ is not a branching point of T. That is v is either a leaf or the minimum depth branching point of T. Observe that a rooted tree that is obtained by swapping the positions of an arbitrary pair of vertices in $V(P_v)$ is also compatible with (X, P). This implies the following observation.

▶ **Observation 18.** *Let T be a rooted tree that is compatible with (X, P). Let $v \in V(T)$ be a maximum depth vertex such that every vertex in $u \in V(P_v) \setminus \{v\}$ is not a branching point of T. Suppose there is a vertex $x \in X \cap V(P_v)$. Then, there is a rooted tree T' with root x that is compatible with (X, P) whose depth is* $\mathrm{height}(X, P)$.

Observation 18 implies that if there is no rooted tree T that is optimal for (X, P) whose root belongs to X, then there is the minimum depth branching point v in T such that $V(P_v) \subseteq I(X \cup P)$ for every rooted tree T that is optimal for (X, P).

The following lemma plays a key role for the construction of optimal elimination trees. To this end, we need some operation on rooted trees. A *vertex removal* of a non-root vertex u (or u is *removed*) from a rooted tree T results a rooted tree that is obtained from T by deleting u and adding an edge between the parent of u and each child of u. When u is the root of T and has only one child, the vertex removal of u is simply deleting u from T.

▶ **Lemma 19.** *Assume that $G[X \cup I(X \cup P)]$ is connected. Then, there exists an optimal rooted tree T for (X, P) such that either (1) the root x of T is in X and every vertex in $I(P \cup \{x\})$ is a child of x or (2) there is a separated bipartition (Y, Z) of X such that $V(P_v) = N(Y) \cap N(Z) \cap I(X \cup P)$, where v is the minimum depth branching point in T.*

Proof. Let T be an optimal rooted tree for (X, P). From Observation 18, we can assume that either the root of T is in X or there is the minimum depth branching point v of T such that $V(P_v) \subseteq I(X \cup P)$.

Suppose first that the root of T is $x \in X$. Let u be the vertex in $I(P \cup \{x\})$ that is not a child of x. Since u has exactly one neighbor x in $G[X \cup I(X \cup P)]$, the rooted tree obtained from T by removing u and adding u as a child of x is also an elimination tree of height not larger than T. The repeated applications of the above argument show that every vertex in $I(P \cup \{x\})$ is a child of x.

Suppose otherwise. Let T be an arbitrary optimal rooted tree for (X, P). Recall that we have $V(P_v) \subseteq I(X \cup P)$ in this case. We construct a separated bipartition (Y, Z) of X as follows. Let v be the minimum depth branching point of T and let W be the set of children of v. Observe that for each $w \in W$, $V(T_w)$ contains at least one vertex of X as otherwise $G[X \cup I(X \cup P)]$ is not connected, which contradicts our assumption. Choose a non-empty proper subset $W' \subset W$. Let $Y = \bigcup_{w \in W'} (V(T_w) \cap X)$ and let $Z = X \setminus Y$. Clearly, (Y, Z) is a separated bipartition of X. This implies that every vertex in $N(Y) \cap N(Z) \cap I(X \cup P)$

belongs to $V(P_v)$. Hence, we have $V(P_v) \supseteq N(Y) \cap N(Z) \cap I(X \cup P)$. In the following we will show that, under an appropriate choice of T, this separated bipartition (Y, Z) satisfies $V(P_v) \subseteq N(Y) \cap N(Z) \cap I(X \cup P)$.

We choose an optimal rooted tree T for (X, P) and a separated bipartition (Y, Z) of X in such a way as to minimize the number of vertices in $V(P_v) \setminus (N(Y) \cap N(Z) \cap I(X \cup P))$. Let $Q = V(P_v) \setminus (N(Y) \cap N(Z) \cap I(X \cup P))$ be the set of vertices in $V(P_v)$ each of which has a neighbor neither in Y nor in Z. We claim that Q is empty and hence such T and (Y, Z) satisfy $V(P_v) \subseteq N(Y) \cap N(Z) \cap I(X \cup P)$. Suppose, for contradiction, let $u \in Q$. As $G[X \cup I(X \cup P)]$ is connected, $V(P_v)$ must have at least two vertices and hence u is removable from T (either u is not the root of T or u is the root which has only one child). We assume, without loss of generality, u has a neighbor in Z. Let W be the set of children w of v with $V(T_w) \cap Y \neq \emptyset$ and let $F_W = \{T_w : w \in W\}$. We construct another rooted tree from T as follows. Remove u and delete the vertices of F_W from T. Then, combine T with $F_W \circ u$ by adding an edge between w and u. It is easy to see that the result of the above operations is an elimination tree of $G[X \cup I(X \cup P)]$ and the height is not larger than the original T. Moreover, the size of Q in the result is strictly smaller than the original one. This is contracting to the minimality of Q. Hence, the lemma follows. ◀

▶ **Lemma 20.** *Let $X \subseteq C$ with $X \neq \emptyset$ and let $P \subseteq C \setminus X$. Then, $\mathrm{td}(X, P)$ is equal to the smaller value of*

$$\min_{x \in X} \mathrm{td}(X \setminus \{x\}, P \cup \{x\}) + 1 \tag{1}$$

and

$$\min_{Y,Z} \max(\mathrm{td}(Y, P), \mathrm{td}(Z, P)) + |N(Y) \cap N(Z) \cap I(X \cup P)|, \tag{2}$$

where, in expression (2), the minimum is taken among all separated bipartitions (Y, Z) of X and if there is no separated bipartition of X then the value of (2) is defined to be infinity.

Proof. First, we show that $\mathrm{td}(X, P)$ is not smaller than the value of both (1) and (2). Consider the case where the value of (1) is not smaller than that of (2). Let x be a vertex in C that attains the minimum of expression (1) and let F be a minimum height rooted forest among all rooted forests that are compatible with $(X \setminus \{x\}, P \cup \{x\})$. By Lemma 15, $F \circ x$ is compatible with (X, P). Therefore, we have

$$\mathrm{td}(X, P) \leq \mathrm{height}(F \circ x) = \mathrm{height}(F) + 1 = \min_{x \in X} \mathrm{td}(X \setminus \{x\}, P \cup \{x\}) + 1.$$

Suppose the other case: the value of (1) is larger than that of (2). In this case, there is a separated bipartition (Y, Z) of X that attains the minimum of expression (2). Let F_Y and F_Z be minimum height rooted forests among all rooted forests that are compatible with (Y, P) and (Z, P), respectively. Let $F_{Y,Z}$ be a rooted forest constructed from F_Y and F_Z as in Lemma 17. Since $F_{Y,Z}$ is compatible with (X, P), we have

$$\begin{aligned} \mathrm{td}(X, P) &\leq \mathrm{height}(F_{Y,Z}) \\ &= \max(\mathrm{height}(F_Y), \mathrm{height}(F_Z)) + |N(Y) \cap N(Z) \cap I(X \cup P)| \\ &= \max(\mathrm{td}(Y, P), \mathrm{td}(Z, P)) + |N(Y) \cap N(Z) \cap I(X \cup P)| \end{aligned}$$

Hence, $\mathrm{td}(X, P)$ is not smaller than the value of both (1) and (2).

Next, we show the other direction. We distinguish the two cases: $G[X \cup I(X \cup P)]$ is connected or not.

Consider the case where $G[X \cup I(X \cup P)]$ is connected. Let T be an elimination tree described in Lemma 19. Then, either the root of T is in X or there exists a separated bipartition (Y, Z) of X with $V(P_v) = N(Y) \cap N(Z) \cap I(X \cup P)$, where v is the minimum depth branching point in T.

Case 1: The root x of T is in X. Let F' be a rooted forest which is obtained from T by deleting the root x. To prove our desired inequality $\mathrm{height}(T) \geq \mathrm{td}(X \setminus \{x\}, P \cup \{x\}) + 1$, we will show that F' is compatible with $(X \setminus \{x\}, P \cup \{x\})$. It is easy to verify that $V(F') \cap C = X \setminus \{x\}$, $V(F') \cap I = I((X \setminus \{x\}) \cup (P \cup \{x\}))$, and $G[V(F')] \subseteq \mathrm{clos}(F')$. By Lemma 19, each vertex $v \in I(P \cup \{x\})$ is a child of x, that is, $I(P \cup \{x\})$ is the set of atomic rooted trees in F'. Hence, F' is compatible with $(X \setminus \{x\}, P \cup \{x\})$.

Case 2: There is a separated bipartition (Y, Z) of X with $V(P_v) = N(Y) \cap N(Z) \cap I(X \cup P)$, where v is the minimum depth branching point of T. Let F' be the rooted forest obtained from T by deleting all the vertices in $V(P_v)$. We claim that F' is partitioned into two rooted forests F_Y and F_Z such that F_Y and F_Z are compatible with (Y, P) and (Z, P), respectively. Note that this claim establishes the inequality $\mathrm{height}(T) \geq \max(\mathrm{td}(Y, P), \mathrm{td}(Z, P)) + |N(Y) \cap N(Z) \cap I(X \cup P)|$. Consider the two rooted forests $F_Y = \{T' \in F' : V(T') \cap Y \neq \emptyset\}$ and $F_Z = \{T' \in F' : V(T') \cap Z \neq \emptyset\}$. Observe that F_Y and F_Z are disjoint since (Y, Z) is separated and every common neighbor of Y and Z must be in $V(P_v)$. As $V(P_v) \subseteq I$, for each child w of v, T_w contains at least one vertex of X. Thus, (F_Y, F_Z) is a bipartition of F'. It is easy to see that $V(F_Y) \cap C = Y$ and $G[V(F_Y)] \subseteq \mathrm{clos}(F_Y)$. Recall that $I(P)$ is empty. By Observation 16, $I(X \cup P)$ is partitioned into $I(Y \cup P)$, $I(Z \cup P)$, and $N(Y) \cap N(Z) \cap I(X \cup P)$. Since every vertex in $I(Y \cup P)$ has a neighbor in Y, $V(F_Y) \cap I = I(Y \cup P)$. Hence, F_Y is compatible with (Y, P). A similar argument shows that F_Z is compatible with (Z, P). Therefore, we have the claim.

Finally, we consider the case where $G[X \cup I(X \cup P)]$ is not connected. Let $\mathcal{C}$ be the set of connected components of $G[X \cup I(X \cup P)]$. We apply Lemma 19 to each component in $\mathcal{C} \setminus I(P)$ and obtain a rooted forest F from $\mathcal{C} \setminus I(P)$. Note that every component in $\mathcal{C} \setminus I(P)$ has at least one vertex of X. If F consists of a single rooted tree, we can apply the same argument with the connected case to the unique rooted tree $T \in F$. Otherwise, the connected components of F naturally induce some separated bipartition of X, which satisfies the desired inequality since $N(Y) \cap N(Z) \cap I(X \cup P)$ is empty. Hence, we have the lemma. ◀

By Lemmas 14 and 20, for $X \subseteq C$ and $P \subseteq C \setminus X$, we can compute $\mathrm{td}(X, P)$ via a standard dynamic programming. When the values $\mathrm{td}(X', P')$ are stored in a table for $X' \subset X$ and $P' \subseteq C \setminus X'$, the value $\mathrm{td}(X, P)$ is computed in $O(2^{|X|}|X|n)$ time. Hence, the running time of our dynamic programming is

$$\begin{aligned}\sum_{X \subseteq C} \sum_{P \subseteq C \setminus X} O(2^{|X|}|X|n) &= \sum_{X \subseteq C} 2^{|C|-|X|} \cdot O(2^{|X|}|X|n) \\ &= \sum_{X \subseteq C} O(2^{|C|}|C|n) \\ &= O(4^{|C|}|C|n).\end{aligned}$$

5 Conclusion

In this paper, we have given a polynomial kernelization and a fixed-parameter algorithm for treedepth when the minimum size of vertex cover of the input graph is parameterized.

The main open questions are improving on the size of kernel and the running time of fixed-parameter algorithm. Jansen [22] showed that there is a kernel for treewidth whose size is quadratic with respect to vertex cover number, which improves the previous result of polynomial kernelization due to Bodlaender *et al.* [7]. He concluded in his paper that the key lemma to the kernelization does not work for pathwidth. This obstacle also appears in the case of treedepth. Chapelle *et al.* [8] improved the running time of their algorithm for treewidth using the fast subset convolution technique due to Björklund *et al.* [2]. One may expect that the bottleneck of our running time (the computation of expression (2) of Lemma 20) can be broken by the fast subset convolution technique. However, this technique does not seem to apply to our fixed-parameter algorithm directly.

Finally, extending our result to more general cases would be interesting. One of such extensions is to use another structural parameterization. Feedback vertex set number would be a good candidate for this line. Another extension is to consider the problem on directed graphs. *Cycle rank* [14] is known to be a directed version of treedepth. To the best of our knowledge, no fixed-parameter tractability result for cycle rank is known.

Acknowledgements. We are grateful to anonymous referees for the suggestions for improving the presentation of the paper.

References

1 B. Aspvall and P. Heggernes. Finding minimum height elimination trees for interval graphs in polynomial time. *BIT Numerical Mathematics*, 34(4):484–509, 1994.

2 A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets möbius: Fast subset convolution. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC'07, pages 67–74. ACM, 2007.

3 H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–21, 1993.

4 H. L. Bodlaender, J. S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, and Z. Tuza. Rankings of graphs. *SIAM J. Discrete Math.*, 11(1):168–181, 1998.

5 H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

6 H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Kernel bounds for structural parameterizations of pathwidth. In *Algorithm Theory – SWAT 2012 – 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4-6, 2012. Proceedings*, pages 352–363, 2012.

7 H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Preprocessing for treewidth: A combinatorial analysis through kernelization. *SIAM J. Discrete Math.*, 27(4):2108–2142, 2013.

8 M. Chapelle, M. Liedloff, I. Todinca, and Y. Villanger. Treewidth and pathwidth parameterized by the vertex cover number. In *Algorithms and Data Structures – 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings*, pages 232–243, 2013.

9 M. Cygan, D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. On cutwidth parameterized by vertex cover. *Algorithmica*, 68(4):940–953, 2014.

10 J. S. Deogun, T. Kloks, D. Kratsch, and H. Müller. On the vertex ranking problem for trapezoid, circular-arc and other graphs. *Discrete Applied Mathematics*, 98(1-2):39–63, 1999.

11 D. Dereniowski and A. Nadolski. Vertex rankings of chordal graphs and weighted trees. *Inf. Process. Lett.*, 98(3):96–100, 2006.

12 Andrew Drucker. New limits to classical and quantum instance compression. *SIAM J. Comput.*, 44(5):1443–1479, 2015.

13 Z. Dvořák, D. Král, and R. Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36, 2013.

14 L. C. Eggan. Transition graphs and the star-height of regular events. *Michigan Math. J.*, 10(4):385–397, 12 1963.

15 M. R. Fellows, D. Hermelin, F. A. Rosamond, and H. Shachnai. Tractable parameterizations for the minimum linear arrangement problem. In *Algorithms – ESA 2013 – 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 457–468, 2013.

16 M. R. Fellows, D. Lokshtanov, N. Misra, F. A. Rosamond, and S. Saurabh. Graph layout problems parameterized by vertex cover. In *Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings*, pages 294–305, 2008.

17 F. V. Fomin, A. C. Giannopoulou, and M. Pilipczuk. Computing tree-depth faster than 2^n. *Algorithmica*, 73(1):202–216, 2015.

18 F. V. Fomin, B. M. P. Jansen, and M. Pilipczuk. Preprocessing subgraph and minor problems: When does a small vertex cover help? *J. Comput. Syst. Sci.*, 80(2):468–495, 2014.

19 F. V. Fomin and Y. Villanger. Finding induced subgraphs via minimal triangulations. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, pages 383–394, 2010.

20 F. V. Fomin and Y. Villanger. Treewidth computation and extremal combinatorics. *Combinatorica*, 32(3):289–308, 2012.

21 G. Gutin, M. Jones, and M. Wahlström. Structural parameterizations of the mixed chinese postman problem. In *Algorithms – ESA 2015 – 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 668–679, 2015.

22 B. M. P. Jansen. On sparsification for computing treewidth. *Algorithmica*, 71(3):605–635, 2015.

23 K. Kitsunai, Y. Kobayashi, K. Komuro, H. Tamaki, and Toshihiro Tano. Computing directed pathwidth in $o(1.89^n)$ time. *Algorithmica*, 75(1):138–157, 2016.

24 C. E. Leiserson. Area-efficient graph layouts. In *Foundations of Computer Science, 1980., 21st Annual Symposium on*, pages 270–281, Oct 1980.

25 J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11(1):134–172, January 1990.

26 B. Monien. The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete. *SIAM J. Algebraic Discrete Methods*, 7(4):505–512, October 1986.

27 J. Nešetřil and P. O. de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006.

28 J. Nešetřil and P. O. de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28. Springer, 2012.

29 A. Pothen. The complexity of optimal elimination trees. Technical Report CS-88-13, Pennsylvania State University, 1988.

30 F. Reidl, P. Rossmanith, F. Sanchez Villaamil, and S. Sikdar. A faster parameterized algorithm for treedepth. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 931–942, 2014.

31 A. Sen, H. Deng, and S. Guha. On a graph partition problem with application to vlsi layout. *Inf. Process. Lett.*, 43(2):87–94, August 1992.

Dynamic Parameterized Problems

R. Krithika[1], Abhishek Sahu[2], and Prafullkumar Tale[3]

1 **The Institute of Mathematical Sciences, HBNI, Chennai, India**
rkrithika@imsc.res.in
2 **The Institute of Mathematical Sciences, HBNI, Chennai, India**
asahu@imsc.res.in
3 **The Institute of Mathematical Sciences, HBNI, Chennai, India**
pptale@imsc.res.in

Abstract

In this work, we study the parameterized complexity of various classical graph-theoretic problems in the dynamic framework where the input graph is being updated by a sequence of edge additions and deletions. Vertex subset problems on graphs typically deal with finding a subset of vertices having certain properties that are of interest to us. In real-world applications, the graph under consideration often changes over time and due to this dynamics, the solution at hand might lose the desired properties. The goal in the area of dynamic graph algorithms is to efficiently maintain a solution under these changes. Recomputing a new solution on the new graph is an expensive task especially when the number of modifications made to the graph is significantly smaller than the size of the graph. In the context of parameterized algorithms, two natural parameters are the size k of the symmetric difference of the edge sets of the two graphs (on n vertices) and the size r of the symmetric difference of the two solutions. We study the Dynamic Π-Deletion problem which is the dynamic variant of the Π-Deletion problem and show NP-hardness, fixed-parameter tractability and kernelization results. For specific cases of Dynamic Π-Deletion such as Dynamic Vertex Cover and Dynamic Feedback Vertex Set, we describe improved FPT algorithms and give linear kernels. Specifically, we show that Dynamic Vertex Cover admits algorithms with running times $1.1740^k n^{\mathcal{O}(1)}$ (polynomial space) and $1.1277^k n^{\mathcal{O}(1)}$ (exponential space). Then, we show that Dynamic Feedback Vertex Set admits a randomized algorithm with $1.6667^k n^{\mathcal{O}(1)}$ running time. Finally, we consider Dynamic Connected Vertex Cover, Dynamic Dominating Set and Dynamic Connected Dominating Set and describe algorithms with $2^k n^{\mathcal{O}(1)}$ running time improving over the known running time bounds for these problems. Additionally, for Dynamic Dominating Set and Dynamic Connected Dominating Set, we show that this is the optimal running time (up to polynomial factors) assuming the Set Cover Conjecture.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases dynamic problems, fixed-parameter tractability, kernelization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.19

1 Introduction

Graphs are discrete mathematical structures that represent binary relations between objects. Due to their tremendous power to model real-world systems, many problems of practical interest can be represented as problems on graphs. Consequently, the design of algorithms on graphs is of major importance in computer science. Applications that employ graph algorithms typically involve large graphs that change over time and a natural goal in this setting is to design algorithms that efficiently maintain a solution under these updates.

Table 1 Summary of known and new results for different dynamic parameterized problems. All running time bounds suppress polynomial factors.

Dynamic Problem	Parameter k	Parameter r
Vertex Cover	1.1740^k, 1.1277^k †	1.2738^r
	$\mathcal{O}(k)$ kernel	$\mathcal{O}(r^2)$ kernel
Connected Vertex Cover	4^k [1], 2^k	W[2]-hard [1]
Feedback Vertex Set	1.6667^k $	3.592^r, 3^r $
	$\mathcal{O}(k)$ kernel	$\mathcal{O}(r^2)$ kernel
Dominating Set	$2^{\mathcal{O}(k^2)}$ [9], 2^k ‡	W[2]-hard [9]
Connected Dominating Set	4^k [1], 2^k ‡	W[2]-hard [1]
Π-Deletion	See Section 2 for hardness and tractability results	

† exponential-space algorithm $ randomized algorithm ‡ optimal under Set Cover Conjecture

In this work, we only consider instances where the possible changes to a graph are edge additions and deletions. Formally, a dynamic version of a graph-theoretic problem is a quintuple (G, G', S, k, r) where G and G' are graphs on the same vertex set and the size of the symmetric difference of their edge sets is upper bounded by k. The set S is a subset of vertices of G satisfying certain property and the task is to determine whether G' has a set S' of vertices satisfying the same property such that the symmetric difference of S and S' is at most r.

Dynamic problems have been recently studied in the parameterized complexity framework [1, 9, 15]. Two relevant parameters for dynamic problem instances are the edit parameter k and the distance parameter r. In this work, we revisit the dynamic versions of several classical problems with these parameterizations and describe parameterized complexity results. For a fixed collection of graphs Π, given a graph G and an integer l, the Π-Deletion problem is to determine if G has a set $S \subseteq V(G)$ of vertices with $|S| \leq l$ such that $G - S \in \Pi$. Π-Deletion is an abstraction of various problems in the graph-theoretic framework and it is known that finding a minimum solution to Π-Deletion is NP-hard for most choices of Π [19]. Hence, it has been extensively studied in various algorithmic realms. We define the dynamic version of this problem referred to as Dynamic Π-Deletion and show NP-hardness, fixed-parameter tractability and kernelization results. Then, for the specific cases of Π-Deletion such as Dynamic Vertex Cover and Dynamic Feedback Vertex Set, we describe improved FPT algorithms with respect to the edit parameter and give linear kernels. Then, for the same parameterization, we describe algorithms for Dynamic Connected Vertex Cover, Dynamic Dominating Set and Dynamic Connected Dominating Set. We also show running time lower bounds for algorithms solving Dynamic Dominating Set and Dynamic Connected Dominating Set assuming the Set Cover Conjecture [5]. Table 1 summarizes our results along with the running time bounds known for these problems.

All graphs considered in this paper are finite, undirected, unweighted and simple. Notation/definitions not given explicitly here can be found in [8]. For a graph G, $V(G)$ and $E(G)$ denote the sets of vertices and edges respectively. The symmetric difference of two subsets $S, S' \subseteq V(G)$, denoted by $d_v(S, S')$, is defined as the size of the set $(S \setminus S') \cup (S' \setminus S)$. For two graphs G and G' on the same vertex set, $d_e(G, G')$ denotes the size of the symmetric difference of $E(G)$ and $E(G')$. For a vertex $u \in V(G)$, its neighbourhood $N_G(u)$ is set of all the vertices adjacent to it and its closed neighbourhood $N_G[u]$ is the set $N_G(u) \cup \{u\}$. This notation is extended to subsets of vertices as $N_G[S] = \bigcup_{v \in S} N_G[v]$ and $N_G(S) = N_G[S] \setminus S$ where $S \subseteq V(G)$. The subscript in the notation for neighbourhood is omitted if the graph

under consideration is clear from the context. The degree of a vertex is the size of its neighbourhood. For a set E' of edges, $V(E')$ denotes the union of the endpoints of the edges in E'. For a set $S \subseteq V(G)$, $G[S]$ and $G - S$ denote the subgraphs of G induced on sets S and $V(G) \setminus S$ respectively. The contraction operation of an edge $e = uv$ in G results in the addition of a new vertex w adjacent to $N_G(u) \cup N_G(v)$ and the deletion of u and v. An independent set is a set of pairwise non-adjacent vertices and a clique is a set of pairwise adjacent vertices. A triangle is a clique of size 3. We also refer to an edgeless graph as an independent set and a complete graph as a clique. A forest is a graph with no cycles and a tree is a connected forest.

In parameterized complexity, every instance of a problem is associated with an non-negative integer called as the parameter that often measures some structural property of the instance. A common parameter is a bound l on the size of an optimum solution to the problem instance I. A problem is fixed-parameter tractable (FPT) with respect to the parameter l if it has an algorithm with $f(l)|I|^{\mathcal{O}(1)}$ running time for some computable function f. The running time $f(l)|I|^{\mathcal{O}(1)}$ where f is an exponential function is also specified as $\mathcal{O}^*(f(l))$ suppressing the polynomial factors. In order to classify parameterized problems as being FPT or not, the W-hierarchy is defined as $\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \cdots \subseteq \mathsf{XP}$ using Boolean circuits and parameterized reductions. It is believed that the subset relations in this sequence are all strict and a parameterized problem that is hard for some complexity class above FPT in this hierarchy is unlikely to be FPT. A kernelization algorithm is a polynomial-time algorithm that transforms an instance (I, l) of the problem to an equivalent instance (I', l') of the same problem such that $|I'| + l' = g(l)$ for some computable function g. The instance (I', l') is called a kernel and if $g(l) = l^{\mathcal{O}(1)}$, then it is called a polynomial kernel and we say that the problem admits a polynomial kernelization (or kernel). We refer the reader to [6, 10, 11] for an introduction to parameterized complexity and kernelization.

2 Dynamic Π-Deletion

A graph property Π is a collection of graphs. Π is said to be (induced) hereditary if for any graph in Π, all of its (induced) subgraphs are in Π. The membership testing problem for Π is the task of determining if a graph is in Π or not. Let I_n denote the graph on n vertices with no edges and K_n denote the complete graph on n vertices. For most natural choices of Π, the Π-Deletion problem is NP-complete [19] and interesting dichotomy results are known in the parameterized complexity framework [2, 16]. We formally define its dynamic variant referred to as Dynamic Π-Deletion as follows.

Dynamic Π-Deletion	**Parameter:** k, r
Input: Graphs G, G' on the same vertex set, a set $S \subseteq V(G)$ such that $G - S \in \Pi$ and integers k, r with $d_e(G, G') \leq k$.	
Question: Does there exist $S' \subseteq V(G')$ with $d_v(S, S') \leq r$ such that $G' - S' \in \Pi$?	

Observe that if Π-Deletion is in NP then so is Dynamic Π-Deletion. We are now ready to state our first result.

▶ **Theorem 2.1.** *Let* Π *be a graph property that includes all independent sets or all cliques. Then,* Π-Deletion *reduces to* Dynamic Π-Deletion *in polynomial time.*

Proof. Let (H, l) be an instance of Π-Deletion. We reduce (H, l) to the instance $(G, G' = H, S = \emptyset, k, r = l)$ of Dynamic Π-Deletion as follows. If Π includes all independent sets then $G = I_{|V(H)|}$ and $k = |E(H)|$. Otherwise, $G = K_{|V(H)|}$ and $k = \binom{|V(H)|}{2} - |E(H)|$. In

both the cases, by the property of Π, $G - S \in \Pi$. Also, the vertex sets of H, G and G' are the same. Then, for a set $S' \subseteq V(H)$, we have $H - S' \in \Pi$ if and only if $G' - S' \in \Pi$ such that $d_v(S, S') = |S'|$. ◀

As a consequence of Theorem 2.1, we have the following hardness result.

▶ **Corollary 2.2.** *Let Π be a property that includes all independent sets or all cliques.*

- *If* Π-Deletion *is* NP*-hard, then* Dynamic Π-Deletion *is* NP*-hard.*
- *If* Π-Deletion *parameterized by solution size is fixed-parameter intractable then* Dynamic Π-Deletion *parameterized by r is fixed-parameter intractable.*
- *If* Π-Deletion *is* NP*-complete and does not admit a polynomial kernel when parameterized by solution size then* Dynamic Π-Deletion *parameterized by r does not admit a polynomial kernel.*

Proof. The NP-hardness and the fixed-parameter intractability results follow straightaway from Theorem 2.1. If Π-Deletion is NP-complete, then Dynamic Π-Deletion (which is in NP) reduces to Π-Deletion in polynomial time. Therefore, if Dynamic Π-Deletion parameterized by r admits a polynomial kernel, such a kernel can be transformed to a polynomial (in solution size) kernel for Π-Deletion using this reduction and the reduction described in Theorem 2.1.Thus, the claimed kernelization hardness follows too. ◀

The following lemma shows that for many choices of Π, to solve Dynamic Π-Deletion, it suffices to look for a solution that contains the current solution.

▶ **Lemma 2.3.** *Let Π be an induced hereditary property. If S' is a solution to the* Dynamic Π-Deletion *instance (G, G', S, k, r) with $d_v(S, S') = r'$, then there is another solution S'' with $d_v(S, S'') \leq r'$ and $S \subseteq S''$.*

Proof. We have $d_v(S, S') = |S \setminus S'| + |S' \setminus S| = r'$. Let S'' be the set $S \cup S'$. Then, $d_v(S, S'') = |S \setminus S''| + |S'' \setminus S| = |S'' \setminus S| = |S' \setminus S| \leq r'$. Now, as $G' - S' \in \Pi$ and Π is hereditary, it follows that $G' - S'' \in \Pi$. ◀

Now, we proceed to show that Dynamic Π-Deletion reduces to Π-Deletion for many choices of Π.

▶ **Theorem 2.4.** *Let Π be an induced hereditary property whose membership testing problem is polynomial-time solvable. Then,* Dynamic Π-Deletion *reduces to* Π-Deletion *in polynomial time.*

Proof. Consider an instance (G, G', S, k, r) of Dynamic Π-Deletion. The task is to determine if G' has a solution S' with $d_v(S, S') \leq r$. If $G' - S \in \Pi$, then S is the required solution S'. Otherwise, from Lemma 2.3, assume that the required set S' contains S. Let H denote the graph $G' - S$. Then, $H - (S' \setminus S) \in \Pi$. Therefore, for a set $T \subseteq V(H)$, we have $H - T \in \Pi$ if and only if $G' - (S \cup T) \in \Pi$ such that $d_v(S, S') = |T|$. ◀

Then, the following claim holds.

▶ **Corollary 2.5.** *Let Π be a hereditary property whose membership testing problem is polynomial-time solvable. If* Π-Deletion *is* FPT *with respect to the solution size l as the parameter, then* Dynamic Π-Deletion *is* FPT *with respect to both r and k as parameters.*

Proof. Consider an instance (G, G', S, k, r) of Dynamic Π-Deletion. Suppose Π-Deletion admits an algorithm with $\mathcal{O}^*(f(l))$ running time. Then, from Theorem 2.4, there is an algorithm $\mathcal{A}$ solving (G, G', S, k, r) in $\mathcal{O}^*(f(r))$ time. Thus, the problem is FPT when parameterized by r. Let $\tilde{E}$ denote the set $E(G') \setminus E(G)$. Let T be a set of vertices of G' of size at most k that contains at least one endpoint of each edge in $\tilde{E}$. As Π is hereditary and $G - S \in \Pi$, it follows that $G' - (S \cup T) \in \Pi$. Now, if $r \geq k$, then $S \cup T$ is the required solution S'. Otherwise, the algorithm $\mathcal{A}$ solving Dynamic Π-Deletion runs in $\mathcal{O}^*(f(k))$ time. Hence, the problem is FPT when parameterized by k. ◀

Finally, we move on to kernelization results.

▶ **Corollary 2.6.** *Let Π be a hereditary property whose membership testing problem is polynomial-time solvable. Suppose Π-Deletion parameterized by the solution size l admits a kernel with $p(l)$ vertices and $q(l)$ edges.*

- *If Π includes all independent sets, then Dynamic Π-Deletion admits a kernel with $2p(r) \leq 2p(k)$ vertices and $q(r) \leq q(k)$ edges.*
- *If Π includes all cliques, then Dynamic Π-Deletion admits a kernel with $2p(r) \leq 2p(k)$ vertices and $q(r) + p(r)^2 \leq q(k) + p(k)^2$ edges.*

Proof. Consider an instance (G, G', S, k, r) of Dynamic Π-Deletion. If $G' - S \in \Pi$ or $r \geq k$, the output of the kernelization algorithm is $(K_1, \emptyset, K_1, 0, 0)$ which is a trivial YES instance of Dynamic Π-Deletion. Suppose $G' - S \notin \Pi$ and $r < k$. Let (H', r') be the kernel of the instance (H, r) of Π-Deletion obtained from Theorem 2.4. Then, $(H'', H', \emptyset, |E(H')|, r')$ is the kernel of (G, G', S, k, r) where $H'' = I_{|V(H')|}$ if Π includes all independent sets and $H'' = K_{|V(H')|}$ if Π includes all cliques. Hence, the claimed bounds on the kernel size follow. ◀

A property Π is interesting if the number of graphs in Π and the number of graphs not in Π are unbounded. Any induced hereditary property that is interesting either contains all independent sets or contains all cliques. Thus, all the above results hold for such properties. In particular, the results of this section hold for the dynamic variants of classical problems like Vertex Cover and Feedback Vertex Set.

3 Dynamic Vertex Cover

A vertex cover is a set of vertices that has at least one endpoint from every edge. Given a graph G and an integer l, Vertex Cover is the problem of determining if G has a vertex cover of size l. Its dynamic variant, Dynamic Vertex Cover, is defined as follows.

Dynamic Vertex Cover	**Parameter:** k, r
Input: Graphs G, G' on the same vertex set, a vertex cover S of G and integers k, r such that $d_e(G, G') \leq k$.	
Question: Does there exist a vertex cover S' of G' such that $d_v(S, S') \leq r$?	

In [1] it is claimed that Dynamic Vertex Cover is W[1]-hard with respect to $k + r$ as the parameter by a reduction from Independent Set parameterized by the solution size. However, the reduction is incorrect and the fixed-parameter intractability does not follow. Clearly, Dynamic Vertex Cover is Dynamic Π-Deletion where Π is the set of all independent sets. Vertex Cover parameterized by the solution size l admits a kernel with at most $2l$ vertices [4] and the current best FPT algorithm runs in $\mathcal{O}^*(1.2738^l)$ time [3].

By Theorem 2.4 and Corollaries 2.5 and 2.6, these results extend to Dynamic Vertex Cover as well. Also, as Vertex Cover is NP-hard, its dynamic version is NP-hard too by Theorem 2.1 and Corollary 2.2. In particular, the following results hold.

- Dynamic Vertex Cover is NP-complete and admits algorithms with $\mathcal{O}^*(1.2738^r)$ and $\mathcal{O}^*(1.2738^k)$ running times.
- Dynamic Vertex Cover admits an $\mathcal{O}(r^2)$ kernel and an $\mathcal{O}(k^2)$ kernel.

We now improve over these results by describing a linear kernel and a faster FPT algorithm with respect to k as the parameter. First, we describe the linear kernelization.

▶ **Theorem 3.1.** Dynamic Vertex Cover *admits a kernel with at most* $2k$ *vertices and at most* k *edges.*

Proof. Consider an instance (G, G', S, k, r) of Dynamic Vertex Cover. By Lemma 2.3, it suffices to search for a solution S' that contains S. As $d_e(G, G') \leq k$, we have $|E(G')\backslash E(G)| \leq k$. From Theorem 2.4 and Corollary 2.6, it suffices to output a linear kernel of the Vertex Cover instance $(G' - S, r)$. We apply the following standard preprocessing on $G' - S$.

▶ **Reduction Rule 3.2.** *Delete isolated vertices.*

▶ **Reduction Rule 3.3.** *If there is a vertex* v *of degree* 1 *add* $N(v)$ *into the solution, decrease* r *by 1 and delete* $N[v]$ *from the graph.*

Let H' denote the resultant graph on which these rules are no longer applicable and r' denote the corresponding budget. Then, $(G' - S, r)$ is a YES instance of Vertex Cover if and only if (H', r') is a YES instance of Vertex Cover. As the minimum degree of H' is at least 2, we have $2|E(H')| \geq 2|V(H')|$. As $G' - S$ has at most k edges, it follows that $|E(H')| \leq k$. Thus, $|V(H')| \leq k$ and from Corollary 2.6 the kernel of (G, G', S, k, r) is $(I_{|V(H')|}, H', \emptyset, k = |E(H')|, r')$. ◀

Next, we describe an algorithm faster than $\mathcal{O}^*(1.2738^k)$.

▶ **Theorem 3.4.** Dynamic Vertex Cover *can be solved in* $\mathcal{O}^*(1.1740^k)$ *time.*

Proof. Consider an instance (G, G', S, k, r) of Dynamic Vertex Cover. By Lemma 2.3, it suffices to search for a solution S' that contains S. From Theorem 2.4 and Corollary 2.6, it suffices to solve the Vertex Cover instance $(G' - S, r)$. We first apply Reduction Rules 3.2 and 3.3 on $H = G' - S$ as long as they are applicable. Then, $|V(H)| \leq |E(H) \leq k$. A minimum vertex cover of a graph on n vertices can be found in $\mathcal{O}^*(1.2002^n)$ time [22]. Thus, an $\mathcal{O}^*(1.2002^k)$ algorithm follows. We will describe a faster branching algorithm where the measure used to bound the number of nodes of the search tree is the number of edges in H and the leaves of the tree are instances corresponding to the empty graph or a graph with maximum degree at most 2. To this end, we apply the following additional rule exhaustively.

▶ **Reduction Rule 3.5.** *If there is a triangle on vertices* u, v, w *such that* $|N(u)| = 2$ *then include* v, w *into the solution and delete* u, v, w *from the graph.*

We eliminate all other triangles in the graph by applying following branching strategy.

▶ **Branching Rule 3.6.** *Let* u, v, w *be the vertices of a triangle.*

- *Branch 1: Include* u *into the solution and delete it from the graph.*
- *Branch 2: Include* $N(u)$ *into the solution and delete* $N[u]$ *from the graph.*

As the degree of a vertex in a triangle is at least 3, the measure drops by at least 3 in the first branch and by at least 6 in the second. When this rule is no longer applicable, we have a triangle-free graph. Now, we state our final branching rule.

▶ **Branching Rule 3.7.** *Let u be a vertex of degree at least three.*
- *Branch 1: Include vertex u into the solution and delete it from the graph.*
- *Branch 2: Include $N(u)$ into the solution and delete $N[u]$ from the graph.*

In the first branch, the measure drops by at least 3. As the graph is triangle-free, no two neighbours of u are adjacent. Further, all vertices have degree at least 2. Therefore, the measure drops by at least $2|N(u)| \geq 6$ in the second branch. As no new edges are added to the graph in any rule, the measure never increases after the application of a reduction or branching rule. All reduction rules can be applied in polynomial time and at each branching rule, we only spend polynomial time to find a vertex to branch on. When k is zero or the maximum degree of the graph is at most 2, finding a minimum vertex cover is polynomial-time solvable. If the vertex cover budget exceeds the permissible value at a branch, that branch is aborted. The initial measure is upper bounded by k and the worst case branching vector is $(3, 6)$. This leads to the recurrence $T(k) \leq T(k-3) + T(k-6)$ whose solution is 1.1740^k. Thus, the algorithm runs in $\mathcal{O}^*(1.1740^k)$ time. ◀

The treewidth of a graph is a parameter that quantifies the closeness of the graph to a tree (see [6] for the precise definition). If the treewidth of the input graph is upper bounded by tw, then a minimum vertex cover can be obtained in $\mathcal{O}^*(2^{tw})$ time [6]. The following result relates the treewidth of a graph to the number of its vertices and edges.

▶ **Lemma 3.8** ([17]). *If G is a graph on n vertices and m edges, then the treewidth of G is upper bounded by $\frac{m}{5.769} + \mathcal{O}(\log n)$. Moreover, a tree decomposition of the corresponding width can be constructed in polynomial time.*

Since the graph H on which a minimum vertex cover is desired has at most k edges and k vertices, its treewidth tw is bounded by $\frac{k}{5.769} + \mathcal{O}(\log k)$. Then, we have the following result.

▶ **Theorem 3.9.** Dynamic Vertex Cover *can be solved in $\mathcal{O}^*(1.1277^k)$ time.*

Though this algorithm is faster than the branching algorithm described earlier, it requires exponential space while the algorithm in Theorem 3.4 requires only polynomial space.

4 Dynamic Connected Vertex Cover

A connected vertex cover of a graph is a vertex cover that induces a connected subgraph and Dynamic Connected Vertex Cover is defined as follows.

Dynamic Connected Vertex Cover	**Parameter:** k, r
Input: Graphs G, G' on the same vertex set, a connected vertex cover S of G and integers k, r such that $d_e(G, G') \leq k$.	
Question: Does there exist a connected vertex cover S' of G' such that $d_v(S, S') \leq r$?	

The problem is NP-complete, W[2]-hard when parameterized by r and admits an $\mathcal{O}^*(4^k)$ algorithm by a reduction to finding a minimum weight Steiner tree [1]. We describe an $\mathcal{O}^*(2^k)$ algorithm by a reduction to finding a group Steiner tree. Given a graph G, an integer p and a family $\mathcal{F}$ of subsets of $V(G)$, the Group Steiner Tree problem is the task of determining whether G contains a tree on at most p vertices that contains at least one vertex

from each set in $\mathcal{F}$. This problem is known to admit an algorithm with $\mathcal{O}^*(2^{|\mathcal{F}|})$ running time [20]. We use this algorithm as a subroutine in our algorithm for Dynamic Connected Vertex Cover. First, we show a lemma on the property of a solution to an instance of Dynamic Connected Vertex Cover analogous to Lemma 2.3.

▶ **Lemma 4.1.** *Consider an instance (G, G', S, k, r) of* Dynamic Connected Vertex Cover. *If S' is a connected vertex cover of G' with $d_v(S, S') = r'$, then $S' \cup S$ is also a connected vertex cover of G' with $d_v(S, S' \cup S) \leq r'$.*

Proof. Assume G' is connected, otherwise the given instance is a NO instance. As a set that contains a vertex cover is also a vertex cover, it follows that $T = S' \cup S$ is a vertex cover of G'. As S' is a vertex cover of G', $S \setminus S'$ is an independent set in G'. As G' is connected, every vertex in $S \setminus S'$ is adjacent to some vertex in S'. Then, as $G'[S']$ is connected and $S' \subseteq T$, it follows that $G'[T]$ is connected too. Further, as $T \setminus S = S' \setminus S$ it follows that $d_v(S, T) = |T \setminus S| + |S \setminus T| = |T \setminus S| = |S' \setminus S| \leq r'$. ◀

Now, we prove the main result of this section.

▶ **Theorem 4.2.** Dynamic Connected Vertex Cover *can be solved in $\mathcal{O}^*(2^k)$ time.*

Proof. Consider an instance (G, G', S, k, r) of Dynamic Connected Vertex Cover. By Lemma 4.1, we can assume that the required solution S' contains S. Observe that $G'[S]$ is not necessarily connected and the edges in G' that are not covered by S are those edges in $E' = (E(G') \setminus E(G)) \cap E(G' - S)$. Now, we show a reduction to finding a group Steiner tree. Contract each connected component of $G'[S]$ to a single vertex. Let H denote the resulting graph and let $X = V(H) \setminus V(G')$. Construct an instance $(H, |X| + r, \mathcal{F})$ of Group Steiner Tree where $\mathcal{F} = \{\{u, v\} \mid uv \in E'\} \cup \{\{x\} \mid \ x \in X\}$. We claim that (G, G', S, k, r) is a YES instance of Dynamic Connected Vertex Cover if and only if $(H, |X| + r, \mathcal{F})$ is a YES instance of Group Steiner Tree.

Suppose G' has a connected vertex cover S' such that $d_v(S, S') \leq r$ and $S \subseteq S'$. As $G'[S']$ is connected, it follows that $H[X \cup (S' \cap V(G' - S))]$ is also connected. Moreover, as $|S' \cap V(G' - S)| \leq r$, it follows that the spanning tree of $H[X \cup (S' \cap V(G' - S))]$ is of size at most $|X| + r$. Hence $(H, |X| + r, \mathcal{F})$ is a YES instance of Group Steiner Tree. Conversely, suppose $(H, |X| + r, \mathcal{F})$ is a YES instance of Group Steiner Tree. Let T denote the solution tree of H. Then, $X \subseteq V(T)$ and $|V(T) \setminus X| = |V(T) \cap V(G' - S)| \leq r$. Define $S' = S \cup (V(T) \cap V(G' - S))$. The size of S' is at most $|S| + r$. Further, $G'[S']$ is connected as S' is obtained from the vertices of T. Also, for every edge in E', T contains at least one of its endpoints. Thus, S' is the desired connected vertex cover of G'. As the sum of the number of connected components of $G'[S]$ and the size of E' is upper bounded by $k + 1$, it follows that $|\mathcal{F}| \leq k + 1$. Thus, the Group Steiner Tree algorithm of [20] runs in $\mathcal{O}^*(2^k)$ time. ◀

5 Dynamic Feedback Vertex Set

A feedback vertex set is a set of vertices whose deletion results in a forest and Dynamic Feedback Vertex Set is defined as follows.

Dynamic Feedback Vertex Set	**Parameter:** k, r
Input: Graphs G, G' on the same vertex set, a feedback vertex set X of G and integers k, r such that $d_e(G, G') \leq k$.	
Question: Does there exist a feedback vertex set X' of G' such that $d_v(X, X') \leq r$?	

Clearly, Dynamic Feedback Vertex Set is Dynamic Π-Deletion where Π is the set of all forests. Given a graph G and an integer l, the Feedback Vertex Set problem is the task of determining if G has a feedback vertex set of at most l vertices. As it is a classical NP-complete problem, its dynamic variant is NP-complete too by Theorem 2.1 and Corollary 2.2. Feedback Vertex Set is known to admit an $\mathcal{O}^*(3.592^l)$ algorithm [18] and a kernel with $\mathcal{O}(l^2)$ vertices [21]. Also, a randomized algorithm that solves the problem in $\mathcal{O}^*(3^l)$ time is known [7]. By Theorem 2.4 and Corollaries 2.5 and 2.6, all these results extend to Dynamic Feedback Vertex Set. In particular, the following results hold.

- Dynamic Feedback Vertex Set can be solved in $\mathcal{O}^*(3.592^r)$ time and in $\mathcal{O}^*(3.592^k)$ time.
- Dynamic Feedback Vertex Set admits randomized algorithms with $\mathcal{O}^*(3^r)$ and $\mathcal{O}^*(3^k)$ running times.
- Dynamic Feedback Vertex Set admits an $\mathcal{O}(r^2)$ kernel and an $\mathcal{O}(k^2)$ kernel.

We now improve these bounds by describing a linear kernel and a faster randomized FPT algorithm with respect to k as the parameter. First, we describe the linear kernelization.

▶ **Theorem 5.1.** Dynamic Feedback Vertex Set *admits a kernel with at most* $4k$ *vertices and at most* $3k$ *edges.*

Proof. Consider an instance (G, G', X, k, r) of Dynamic Feedback Vertex Set. Observe that if G' is obtained from G by only deleting edges, then X is feedback vertex set of G' too. Also, edges in $E(G') \setminus E(G)$ that have an endpoint in X do not affect the solution. Moreover, from Lemma 2.3, it suffices to search for a feedback vertex set of G' that contains X. Let H be the subgraph of G' induced on $V(G') \setminus X$. From Theorem 2.4, (G, G', X, k, r) is a YES instance of Dynamic Feedback Vertex Set if and only if (H, r) is a YES instance of Feedback Vertex Set. From Corollary 2.6, it suffices to output a linear kernel of the instance (H, r). We primarily exploit the fact that H is obtained by adding at most k edges to a forest. This implies that $|E(H)| \leq |V(H)| + k - 1$. Let $\tilde{E}$ be the set of edges in G' whose both endpoints are in $V(G) \setminus X$ and $U = V(\tilde{E})$. We apply the following reduction rule to $G - X$. Note that $G - X$ is a subgraph of H as $E(H) = E(G - X) \cup \tilde{E}$.

▶ **Reduction Rule 5.2.** *If there is a vertex* v *of degree at most* 1 *such that* $v \notin U$*, then delete* v *from the graph.*

This rule preserves the size of a minimum feedback vertex set as v has degree at most 1 in H too and no minimal feedback vertex set of H contains it. As the number of vertices with degree at least 3 is upper bounded by the number of leaves in a forest, it follows that the number of vertices of degree at least 3 is at most $2k$ on the resulting graph G'' on which this rule is not applicable. Consider the graph H'' obtained from G'' by adding $\tilde{E}$. We once again delete vertices of degree at most 1 (if any) and then apply following reduction rule exhaustively.

▶ **Reduction Rule 5.3.** *If there is a vertex* v *of degree* 2*, then delete* v *and add an edge between its two neighbours.*

Once again this rule preserves the size of a minimum feedback vertex set as any minimal feedback vertex set of H'' that contains v can be modified into another minimal feedback vertex set of at least the same size that does not contain v. Note that the application of Reduction Rules 5.2 and 5.3 ensure that $|E(H'')| \leq |V(H'')| + k - 1$ is satisfied. When neither of the reduction rules are applicable on H'', the minimum degree of H'' is at least 3 and $|E(H'')| \leq |V(H'')| + k - 1$. This implies that $1.5|V(H'')| \leq |E(H'')| \leq |V(H'')| + k - 1$

and hence $|V(H'')| \leq 2k-2$, $|E(H'')| \leq 3k-3$. Also, (H, r) is a YES instance of Feedback Vertex Set if and only if (H'', r) is a YES instance of Feedback Vertex Set. Thus, from Corollary 2.6, the kernel of (G, G', S, k, r) is $(I_{|V(H'')|}, H'', \emptyset, k = |E(H'')|, r)$. ◀

Next, we proceed to describe a randomized FPT algorithm for the problem. If the treewidth of the input graph is upper bounded by tw then there is a randomized $\mathcal{O}^*(3^{tw})$ time algorithm that computes a minimum feedback vertex set [7]. Further, for finding a minimum feedback vertex set of a graph on n vertices, there is a randomized algorithm running in $\mathcal{O}(1.6667^n)$ time [12]. The following result relates the treewidth of a graph to the number of its vertices and edges.

▶ **Lemma 5.4** ([13]). *For any $\epsilon > 0$, there exists an integer n_ϵ such that for every connected graph G on n vertices and m edges with $n > n_\epsilon$ and $1.5n \leq m \leq 2n$, the treewidth of G is upper bounded by $\frac{m-n}{3} + \epsilon n$. Moreover, a tree decomposition of the corresponding width can be constructed in polynomial time.*

This theorem along with the described linear kernelization leads to the following result.

▶ **Theorem 5.5.** Dynamic Feedback Vertex Set *admits a randomized algorithm running in $\mathcal{O}^*(1.6667^k)$ time.*

Proof. Consider an instance (G, G', X, k, r) of Dynamic Feedback Vertex Set. Let (H'', r) be the corresponding instance of Feedback Vertex Set obtained from the linear kernelization of Theorem 5.1. That is, H'' is a graph (not necessarily simple) on n vertices and m edges such that $m \leq n+k-1$ and $n \leq 2k-2$. Further, every vertex of H'' has degree at least 3 and hence $m \geq 1.5n$. If $m > 2n$, then as $m \leq n+k-1$, we have $n < k-1$. Then, a minimum feedback vertex set of H'' can be obtained in $\mathcal{O}(1.6667^k)$ using the randomized exact exponential-time algorithm described in [12]. Otherwise, $1.5n \leq m \leq 2n$. Let ϵ be a constant (to be chosen subsequently). Then, let n_ϵ be the integer obtained from Lemma 5.4 satisfying the required properties. If $n \leq n_\epsilon$, then a minimum feedback vertex set of H'' can be obtained in constant time as n_ϵ is a constant depending only on ϵ. Otherwise, the treewidth of H'' is at most $t = \frac{m-n}{3} + \epsilon n = \frac{m}{3} + n(\epsilon - \frac{1}{3})$. Then, using the randomized algorithm described in [7], a minimum feedback vertex set of H'' can be obtained in $\mathcal{O}^*(3^t)$ time. Now, by choosing ϵ to be a sufficiently small constant, t can be made arbitrarily close to $\frac{m-n}{3}$. For instance, if $\epsilon = 10^{-10}$, then t is $.\overline{3}m - .3333333332\overline{3}n$. As $\frac{m-n}{3} \leq \frac{n+k-1-n}{3} = \frac{k}{3}$, the algorithm in [7] runs in $\mathcal{O}^*(1.443^k)$ time. ◀

6 Dynamic Dominating Set

A dominating set is a set of vertices that has a non-empty intersection with the closed neighbourhood of every vertex and a set $S \subseteq V(G)$ is said to dominate another set $T \subseteq V(G)$ if $T \subseteq N[S]$. The Dynamic Dominating Set problem is defined as follows.

Dynamic Dominating Set	**Parameter:** k, r
Input: Graphs G, G' on the same vertex set, a dominating set D of G and integers k, r such that $d_e(G, G') \leq k$.	
Question: Does there exist a dominating set D' of G' such that $d_v(D, D') \leq r$?	

The problem is NP-complete and W[2]-hard when parameterized by r [9]. Also, it is FPT when parameterized by k via an $2^{\mathcal{O}(k^2)}$ algorithm but admits no polynomial kernel unless NP $\subseteq$ coNP/poly [9]. We describe a faster FPT algorithm for this parameterization. First, we show that it suffices to look for a dominating set with a specific property.

▶ **Lemma 6.1.** *Consider an instance (G, G', D, k, r) of* Dynamic Dominating Set. *If D' is a dominating set of G' with $d_v(D, D') = r'$, then $D' \cup D$ is also a dominating set of G' with $d_v(D, D' \cup D) \leq r'$.*

Proof. As a set that contains a dominating set is also a dominating set, it follows that $D'' = D' \cup D$ is a dominating set of G'. Further, $d_v(D, D'') = |D'' \setminus D| + |D \setminus D''| = |D'' \setminus D| = |D' \setminus D| \leq r'$. ◀

Now, we solve Dynamic Dominating Set by reducing it to an instance of Set Cover. In the Set Cover problem, we are given a family $\mathcal{F}$ of subsets of a universe U and a positive integer ℓ. The problem is to determine whether there exists a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most ℓ such that $U = \bigcup_{X \in \mathcal{F}'} X$.

▶ **Theorem 6.2.** Dynamic Dominating Set *admits an algorithm that runs in $\mathcal{O}^*(2^k)$ time.*

Proof. Consider an instance (G, G', D, k, r) of Dynamic Dominating Set. If G' is obtained from G by only adding edges, then D is dominating set of G'. The only kind of edge deletions that could possibly affect the solution are those that have one endpoint in D and the other endpoint in $V(G') \setminus D$. Further, as $d_e(G, G') \leq k$, $|V(G') \setminus N_{G'}[D]| \leq k$. That is, there are at most k vertices in G' that are not dominated by D. Let H be the subgraph of G' induced on $V(G') \setminus D$. Partition $V(H)$ into two sets $C = N_{G'}(D)$ and $B = V(H') \setminus C$.

We claim that (G, G', D, k, r) is a YES instance of Dynamic Dominating Set if and only if there exists a set $P \subseteq V(H)$ of cardinality at most r such that $B \subseteq N_H[P]$. If there is a set P of size at most r in $V(H)$ that dominates B, then $D' = D \cup P$ is a dominating set of G' with $d_v(D, D') \leq r$. Hence, (G, G', D, k, r) is a YES instance of Dynamic Dominating Set. Conversely, suppose there is a dominating set D' of G' with $d_v(D, D') \leq r$. Define D'' as $D' \setminus D$. Note that $|D''| \leq r$. By construction of H, B is not dominated by D and hence $B \subseteq N_H[D'']$. This implies that D'' is the required set of vertices of H that dominates B.

The problem now reduces to finding a set of at most r vertices from $B \cup C$ that dominates B in H. We construct an instance of Set Cover with $U = B$, $\mathcal{F} = \{N_H(u) \cap B \mid u \in C\} \cup \{N_H[w] \cap B \mid w \in B\}$ and $\ell = r$. Then, there exists a set P of size at most r in H which dominates B if and only if $(U, \mathcal{F}, \ell)$ is a YES instance of Set Cover. A set $X \in \mathcal{F}$ is said to be associated with a vertex v in C if $X = N_H(v) \cap B$ or with a vertex v in B if $X = N_H[v] \cap B$. If there exists a set P with desired property, then every vertex w in B is contained in open or closed neighbourhood of some vertex in P. Consider the subfamily $\mathcal{F}'$ of $\mathcal{F}$ that are associated with vertices in P. Every element of U is contained in at least one of these sets. Thus, $\mathcal{F}'$ is the required set cover. Conversely, if there exists a set cover $\mathcal{F}'$ of size at most $\ell = r$, then let P' be the set of vertices which are associated with sets in $\mathcal{F}'$. Then, $|P'| = |\mathcal{F}'| \leq r$ and every vertex in B is either in P' or is adjacent to some vertex in P'. Hence, P' is the desired set.

As any instance $(U, \mathcal{F}, \ell)$ of Set Cover can be solved in $\mathcal{O}^*(2^{|U|})$ [14] and $|U| = |B| \leq k$, the claimed running time bound follows. ◀

Finally, we show a lower bound on the running time of an algorithm that solves Dynamic Dominating Set assuming the Set Cover Conjecture which states that Set Cover cannot be solved in $\mathcal{O}^*((2-\epsilon)^{|U|})$ for any $\epsilon > 0$ [5]. We do so by a reduction from Set Cover to Dynamic Dominating Set.

▶ **Theorem 6.3.** Dynamic Dominating Set *does not admit an algorithm with $\mathcal{O}^*((2-\epsilon)^k)$ running time for any $\epsilon > 0$ assuming the Set Cover Conjecture.*

Proof. Consider an instance $(U, \mathcal{F}, \ell)$ of Set Cover where $U = \{u_1, \cdots, u_n\}$ and $\mathcal{F} = \{S_1, \cdots, S_m\}$. Without loss of generality, assume that every u_i is in at least one set S_j. Let G be the graph with vertex set $U \cup V \cup \{x\}$ where $U = \{u_1, \cdots, u_n\}$ and $V = \{s_1, \cdots, s_m\}$. The set V is a clique and the set U is an independent set in G. Further, a vertex u_i is adjacent to s_j if and only if $u_i \in S_j$ and x is adjacent to every vertex in $U \cup V$. Clearly, $D = \{x\}$ is a dominating set of G. Let G' be the graph obtained from G by deleting edges between x and U. We claim that $(U, \mathcal{F}, \ell)$ is a YES instance of Set Cover if and only if $(G, G', D = \{x\}, k = n, r = \ell)$ is a YES instance of Dynamic Dominating Set. Suppose $\mathcal{F}'$ is a set cover of size at most ℓ. Then, $D' = D \cup \{s_i \mid S_i \in \mathcal{F}'\}$ is a dominating set of G' with $d_v(D, D') \leq \ell$. Conversely, suppose G' has a dominating set D' with $d_v(D, D') \leq \ell$. From Lemma 6.1, assume that $D \subseteq D'$ and so $|D' \setminus D| \leq \ell$. For every vertex $u \in U \cap D'$, replace u by one of its neighbours in V. The resultant dominating set D'' contains D and satisfies $D'' \setminus D \subseteq V$. Now, $\{S_i \in \mathcal{F} \mid v_i \in D'' \cap V\}$ is a set cover of size at most ℓ. This leads to the claimed lower bound under the Set Cover Conjecture. ◀

7 Dynamic Connected Dominating Set

A connected dominating set is a dominating set that induces a connected subgraph and the Dynamic Connected Dominating Set problem is defined as follows.

Dynamic Connected Dominating Set	**Parameter:** k, r
Input: Graphs G, G' on the same vertex set, a connected dominating set D of G and integers k, r such that $d_e(G, G') \leq k$.	
Question: Does there exist a connected dominating set D' of G' such that $d_v(D, D') \leq r$?	

The problem is NP-complete and admits an $\mathcal{O}^*(4^k)$ algorithm by a reduction to finding a minimum weight Steiner tree [1]. We now show that it admits an $\mathcal{O}^*(2^k)$ algorithm by a reduction to finding a group Steiner tree. Analogous to the problems considered earlier, we first prove a property on the required solution.

▶ **Lemma 7.1.** *Consider an instance (G, G', D, k, r) of* Dynamic Connected Dominating Set. *If D' is a connected dominating set of G' with $d_v(D, D') = r'$, then $D' \cup D$ is also a connected dominating set of G' with $d_v(D, D' \cup D) \leq r'$.*

Proof. Assume G' is connected, otherwise the given instance is a NO instance. As a set that contains a dominating set is also a dominating set, it follows that $D'' = D' \cup D$ is a dominating set of G'. Now, $D'' \setminus D$ is $D' \setminus D$. As D' is a dominating set of G', every vertex in $D \setminus D'$ is adjacent to some vertex in D'. Then, as $G'[D']$ is connected and $D' \subseteq D''$, it follows that $G'[D'']$ is connected too. Further, $d_v(D, D'') = |D'' \setminus D| + |D \setminus D''| = |D'' \setminus D| = |D' \setminus D| \leq r' \leq r$. ◀

Now, we describe an algorithm by reducing the problem to finding a Group Steiner tree.

▶ **Theorem 7.2.** Dynamic Connected Dominating Set *admits an algorithm that runs in $\mathcal{O}^*(2^k)$ time.*

Proof. Consider an instance (G, G', D, k, r) of Dynamic Connected Dominating Set. Assume G' is connected. Clearly, the edges in $E(G') \setminus E(G)$ do not affect the solution. Partition $V(G') \setminus D$ into two sets $C = N_{G'}(D)$ and $B = V(G') \setminus C$. Contract each connected component of $G'[D]$ to a single vertex. Let H denote the resulting graph and let $X = V(H) \setminus V(G')$. Construct an instance $(H, |X| + r, \mathcal{F})$ of Group Steiner Tree where

$\mathcal{F} = \{N_{G'}[v] \mid \ v \in B\} \cup \{\{x\} \mid \ x \in X\}$. We claim that (G, G', D, k, r) is a YES instance of Dynamic Connected Dominating Set if and only if $(H, |X| + r, \mathcal{F})$ is a YES instance of Group Steiner Tree.

Suppose there exists a connected dominating set D' of G' such that $d_v(D, D') \leq r$ and $D \subseteq D'$. For every vertex u in B, there is a vertex x in $D' \cap (B \cup C)$ that is adjacent to u. As $G'[D']$ is connected, it follows that $H[X \cup (D' \cap (B \cup C))]$ is also connected. Moreover, as $|D' \cap (C \cup B)| \leq |D'| - |D| \leq r$, it follows that the spanning tree of $H[X \cup (D' \cap (C \cup B))]$ is of size at most $|X| + r$. Hence $(H, |X| + r, \mathcal{F})$ is a YES instance of Group Steiner Tree. Suppose $(H, |X| + r, \mathcal{F})$ is a YES instance of Group Steiner Tree. Let T denote the solution tree of H. Then, $X \subseteq V(T)$ and $|V(T) \setminus X| = |V(T) \cap (C \cup B)| \leq r$. Define $D' = D \cup (V(T) \cap (B \cup C))$. The size of D' is at most $|D| + r$. Now, $G'[D']$ is connected as D' is obtained from the vertices of T. Also, for every vertex u in B, T contains at least one vertex in $N_{G'}[v]$. Thus, D' is the desired connected dominating set of G'.

As $E(G) \setminus E(G')$, the sum of the number of connected components of $G'[D]$ and the size of B is upper bounded by $k + 1$. That is, $|\mathcal{F}| \leq k + 1$ and the Group Steiner Tree algorithm of [20] runs in $\mathcal{O}^*(2^k)$ time. ◀

Finally, by a reduction from Set Cover to Dynamic Connected Dominating Set, we show the following result.

▶ **Theorem 7.3.** Dynamic Connected Dominating Set *does not admit an algorithm with* $\mathcal{O}^*((2-\epsilon)^k)$ *running time for any* $\epsilon > 0$ *assuming the Set Cover Conjecture.*

Proof. We observe that the reduction described in Theorem 6.3 reduces instances of Set Cover to instances of Dynamic Connected Dominating Set. Thus, the claimed lower bound holds assuming the Set Cover Conjecture. ◀

8 Conclusion

We described FPT algorithms for the dynamic variants of several classical parameterized problems with respect to the edit parameter. The role of structural parameters like treewidth and pathwidth in this setting remains to be explored. Also, further exploration of the contrast between the parameterized complexity of a problem and its dynamic version is an interesting direction of research.

Acknowledgments. We are grateful to Saket Saurabh for the invaluable discussions and for providing several useful pointers that led to the writing of this paper. We are also thankful to the reviewers for their suggestions that strengthened the results for Dynamic Π-Deletion.

References

1 F. N. Abu-Khzam, J. Egan, M. R. Fellows, F. A. Rosamond, and P. Shaw. On the Parameterized Complexity of Dynamic Problems. *Theoretical Computer Science*, 607 (3):426–434, 2015.

2 L. Cai. Fixed-Parameter Tractability of Graph Modification Problems for Hereditary Properties. *Information Processing Letters*, 58(4):171–176, 1996.

3 J. Chen, I. A. Kanj, and W. Jia. Vertex Cover: Further Observations and Further Improvements. *Journal of Algorithms*, 41(2):280–301, 2001.

4 J. Chen, I. A. Kanj, and G. Xia. Improved Upper Bounds for Vertex Cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010.

5 M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlstrom. On Problems As Hard As CNF-SAT. In *Proceedings of the IEEE Conference on Computational Complexity*, CCC'12, pages 74–84. IEEE Computer Society, 2012.

6 M. Cygan, F. V. Fomin, K. Lukasz, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

7 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 *Proceedings of the IEEE* 52^{nd} *Annual Symposium on Foundations of Computer Science*, FOCS'11, pages 150–159, 2011.

8 R. Diestel. *Graph Theory*. Springer-Verlag Berlin Heidelberg, 2006.

9 R. G. Downey, J. Egan, M. R. Fellows, F. A. Rosamond, and P. Shaw. Dynamic Dominating Set and Turbo-Charging Greedy Heuristics. *Tsinghua Science and Technology*, 19(4):329–337, 2014.

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

11 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

12 F. V. Fomin, S. Gaspers, D. Lokshtanov, and S. Saurabh. Exact Algorithms via Monotone Local Search. In *Proceedings of the* 48^{th} *Annual ACM Symposium on Theory of Computing*, STOC'16. ACM, 2016.

13 F. V. Fomin, S. Gaspers, S. Saurabh, and A. A. Stepanov. On Two Techniques of Combining Branching and Treewidth. *Algorithmica*, 54(2):181–207, 2009.

14 F. V. Fomin, D. Kratsch, and G. J. Woeginger. Exact (exponential) Algorithms for the Dominating Set Problem. In *Proceedings of the* 30^{th} *International Workshop on Graph-Theoretic Concepts in Computer Science*, WG'04, pages 245–256. Springer Berlin Heidelberg, 2004.

15 S. Hartung and R. Niedermeier. Incremental List Coloring of Graphs, Parameterized by Conservation. *Theoretical Computer Science*, 494:86–98, 2013.

16 S. Khot and V. Raman. Parameterized Complexity of Finding Subgraphs with Hereditary Properties. *Theoretical Computer Science*, 289(2):997–1008, 2002.

17 J. Kneis, D. Mölle, S. Richter, and P. Rossmanith. A Bound on the Pathwidth of Sparse Graphs with Applications to Exact Algorithms. *SIAM Journal on Discrete Mathematics*, 23(1):407–427, 2009.

18 T. Kociumaka and M. Pilipczuk. Faster Deterministic Feedback Vertex Set. *Information Processing Letters*, 114(10):556–560, 2014.

19 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.

20 N. Misra, G. Philip, V. Raman, S. Saurabh, and S. Sikdar. FPT Algorithms for Connected Feedback Vertex Set. *Journal of Combinatorial Optimization*, 24(2):131–146, 2012.

21 S. Thomassé. A Quadratic Kernel for Feedback Vertex Set. In *Proceedings of the* 12^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'09, pages 115–119, 2009.

22 M. Xiao and H. Nagamochi. Exact Algorithms for Maximum Independent Set. In *Proceedings of the* 24^{th} *International Symposium on Algorithms and Computation*, ISAAC'13, pages 328–338. Springer Berlin Heidelberg, 2013.

A $2\ell k$ Kernel for ℓ-Component Order Connectivity*

Mithilesh Kumar[1] and Daniel Lokshtanov[2]

1 **Department of Informatics, University of Bergen, Norway**
Mithilesh.Kumar@ii.uib.no
2 **Department of Informatics, University of Bergen, Norway**
daniello@ii.uib.no

Abstract

In the ℓ-Component Order Connectivity problem ($\ell \in \mathbb{N}$), we are given a graph G on n vertices, m edges and a non-negative integer k and asks whether there exists a set of vertices $S \subseteq V(G)$ such that $|S| \leq k$ and the size of the largest connected component in $G - S$ is at most ℓ. In this paper, we give a kernel for ℓ-Component Order Connectivity with at most $2\ell k$ vertices that takes $n^{\mathcal{O}(\ell)}$ time for every constant ℓ. On the way to obtaining our kernel, we prove a generalization of the q-Expansion Lemma to weighted graphs. This generalization may be of independent interest.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Parameterized algorithms, Kernel, Component Order Connectivity, Max-min allocation, Weighted expansion

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.20

1 Introduction

In the classic Vertex Cover problem, the input is a graph G and integer k, and the task is to determine whether there exists a vertex set S of size at most k such that every edge in G has at least one endpoint in S. Such a set is called a *vertex cover* of the input graph G. An equivalent definition of a vertex cover is that every connected component of $G - S$ has at most 1 vertex. This view of the Vertex Cover problem gives rise to a natural generalization: can we delete at most k vertices from G such that every connected component in the resulting graph has at most ℓ vertices? Here we study this generalization. Formally, for every integer $\ell \geq 1$, we consider the following problem, called ℓ-Component Order Connectivity (ℓ-COC).

ℓ-Component Order Connectivity (ℓ-COC)
Input: A graph G on n vertices and m edges, and a positive integer k.
Task: determine whether there exists a set $S \subseteq V(G)$ such that $\|S\| \leq k$ and the maximum size of a component in $G - S$ is at most ℓ.

The set S is called an ℓ-COC *solution*. For $\ell = 1$, ℓ-COC is just the Vertex Cover problem. Aside from being a natural generalization of Vertex Cover, the family $\{\ell\text{-COC} : \ell \geq 1\}$ of problems can be thought of as a vulnerability measure of the graph G - how many vertices

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no. 306992 and the Beating Hardness by Pre-processing grant funded by the Bergen Research Foundation.

of G have to fail for the graph to break into small connected components? For a study of ℓ-COC from this perspective see the survey of Gross et al. [12].

From the work of Lewis and Yannakakis [16] it immediately follows that ℓ-COC is NP-complete for every $\ell \geq 1$. This motivates the study of ℓ-COC within paradigms for coping with NP-hardness, such as approximation algorithms [22], exact exponential time algorithms [11], parameterized algorithms [5, 7] and kernelization [14, 17]. The ℓ-COC problems have (for some values of ℓ) been studied within all four paradigms, see the related work section.

In this work we focus on ℓ-COC from the perspective of parameterized complexity and kernelization. Our main result is an algorithm that given an instance (G, k) of ℓ-COC, runs in polynomial time, and outputs an equivalent instance (G', k') such that $k' \leq k$ and $|V(G')| \leq 2\ell k$. This is called a *kernel* for ℓ-COC with $2\ell k$ vertices. Our kernel significantly improves over the previously best known kernel with $O(\ell k(k + \ell))$ vertices by Drange et al. [8]. Indeed, for $\ell = 1$ our kernel matches the size of the smallest known kernel for Vertex Cover [3] that is based on the classic theorem of Nemhauser and Trotter [18].

Related Work. 1-COC, better known as Vertex Cover, is extremely well studied from the perspective of approximation algorithms [22, 6], exact exponential time algorithms [10, 19, 24], parameterized algorithms [5, 4] and kernelization [3, 18]. The kernel with $2k$ vertices for Vertex Cover is considered one of the fundamental results in the field of kernelization. The 2-COC problem is also well studied, and has been considered under several different names. The problem, or rather the dual problem of finding a largest possible set S that induces a subgraph in which every connected component has order at most 2, was first defined by Yannakakis [25] under the name Dissociation Set. The problem has attracted attention in exact exponential time algorithms [13, 23], the fastest currently known algorithm [23] has running time $O(1.3659^n)$. 2-COC has also been studied from the perspective of parameterized algorithms [2, 20] (under the name Vertex Cover P_3) as well as approximation algorithms [21]. The fastest known parameterized algorithm, due to Chang et al. [2] has running time $1.7485^k n^{O(1)}$, while the best approximation algorithm, due to Tu and Zhou [21] has factor 2.

For the general case of ℓ-COC, $\ell \geq 1$, Drange et al. [8] gave a simple parameterized algorithm with running time $(\ell + 1)^k n^{O(1)}$, and a kernel with $O(k\ell(\ell + k))$ vertices. The parameterized algorithm of Drange et al. [8] can be improved to $(\ell + 0.0755)^k n^{O(1)}$ by reducing to the $(\ell + 1)$-Hitting Set problem, and applying the iterative compression based algorithm for $(\ell + 1)$-Hitting Set due to Fomin et al. [9]. The reduction to $(\ell + 1)$-Hitting Set, coupled with the simple factor $(\ell + 1)$-approximation algorithm for $(\ell + 1)$-Hitting Set [22] immediately also yields an $(\ell + 1)$-approximation algorithm for ℓ-COC. There has also been some work on ℓ-COC when the input graph is restricted to belong to a graph class, for a discussion of this work see [8].

Comparing the existing results with our work, we see that our kernel improves over the kernel of Drange et al. [8] from at most $O(k\ell(\ell + k))$ vertices to at most $2k\ell$ vertices. Our kernel is also the first kernel with a linear number of vertices for every fixed $\ell \geq 2$.

Our Methods. Our kernel for ℓ-COC hinges on the concept of a *reducible pair* of vertex sets. Essentially (*this is not the formal definition used in the paper!*), a reducible pair is a pair (X, Y) of disjoint subsets of $V(G)$ such that $N(Y) \subseteq X$, every connected component of $G[Y]$ has size at most ℓ, and every solution S to G has to contain at least $|X|$ vertices from $G[X \cup Y]$. If a reducible pair is identified, it is easy to see that one might just as well pick all of X into the solution S, since any solution has to pay $|X|$ inside $G[X \cup Y]$, and after X

is deleted, Y breaks down into components of size at most ℓ and is completely eliminated from the graph.

At this point there are several questions. (a) How does one argue that a reducible pair is in fact reducible? That is, how can we prove that any solution has to contain at least $|X|$ vertices from $X \cup Y$? (b) How big does G have to be compared to k before we can assert the existence of a reducible pair? Finally, (c) even if we can assert that G contains a reducible pair, how can we find one in polynomial time?

To answer (a) we restrict ourselves to reducible pairs with the additional property that each connected component C of $G[Y]$ can be assigned to a vertex $x \in N(C)$, such that for every $x \in X$ the total size of the components assigned to x is at least ℓ. Then x together with the components assigned to it form a set of size at least $\ell + 1$ and have to contain a vertex from the solution. Since we obtain such a connected set for each $x \in X$, the solution has to contain at least $|X|$ vertices from $X \cup Y$. Again we remark that this definition of a reducible pair is local to this section, and not the one we actually end up using.

To answer (b) we first try to use the q-Expansion Lemma (see [5]), a tool that has found many uses in kernelization. Roughly speaking the Expansion Lemma says the following: if $q \geq 1$ is an integer and H is a bipartite graph with bipartition (A, B) and B is at least q times larger than A, then one can find a subset X of A and a subset Y of B such that $N(Y) \subseteq X$, and an assignment of each vertex $y \in Y$ to a neighbor x of y, such that every vertex x in X has at least q vertices in Y assigned to it.

Suppose now that the graph does have an ℓ-COC solution S of size at most k, and that $V(G) \setminus S$ is sufficiently large compared to S. The idea is to apply the Expansion Lemma to the bipartite graph H, where the A side of the bipartition is S and the B side has one vertex for each connected component of $G - S$. We put an edge in H between a vertex v in S and a vertex corresponding to a component C of $G - S$ if there is an edge between v and C in G. If $G - S$ has at least $|S| \cdot \ell$ connected components, we can apply the ℓ-Expansion Lemma on H, and obtain a set $X \subseteq S$, and a collection $\mathcal{Y}$ of connected components of $G - X$ satisfying the following properties. Every component $C \in \mathcal{Y}$ satisfies $N(C) \subseteq X$ and $|C| \leq \ell$. Furthermore, there exists an assignment of each connected component C to a vertex $x \in N(C)$, such that every $x \in X$ has at least ℓ components assigned to it. Since x has at least ℓ components assigned to it, the total size of the components assigned to x is at least ℓ. But then, X and $Y = \bigcup_{C \in \mathcal{Y}} C$ form a reducible pair, giving an answer to question (b). Indeed, this argument can be applied whenever the number of components of $G - S$ is at least $\ell \cdot |S|$. Since each component of $G - S$ has size at most ℓ, this means that the argument can be applied whenever $|V(G) \setminus S| \geq \ell^2 \cdot |S| \geq \ell^2 k$.

Clearly this argument fails to yield a kernel of size $2\ell k$, because it is only applicable when $|V(G)| = \Omega(\ell^2 k)$. At this point we note that the argument above is extremely wasteful in one particular spot: we used the *number* of components assigned to x to lower bound the *total size* of the components assigned to x. To avoid being wasteful, we prove a new variant of the Expansion Lemma, where the vertices on the B side of the bipartite graph H have non-negative integer weights. This new Weighted Expansion lemma states that if $q, W \geq 1$ are integers, H is a bipartite graph with bipartition (A, B), every vertex in B has a non-negative integer weight which is at most W, and the total weight of B is at least $(q + W - 1) \cdot |A|$, then one can find a subset X of A and a subset Y of B such that $N(Y) \subseteq X$, and an assignment of each vertex $y \in Y$ to a neighbor x of y, such that for every vertex in X, the total weight of the vertices assigned to it is at least q. The proof of the Weighted Expansion Lemma is based on a combination of the usual, unweighted Expansion Lemma with a variant of an argument by Bezáková and Dani [1] to round the linear program for

Max-min Allocation of goods to customers.

Having the Weighted Expansion Lemma at hand we can now repeat the argument above for proving the existence of a reducible pair, but this time, when we build H, we can give the vertex corresponding to a component C of $G - S$ weight $|C|$, and apply the Weighted Expansion Lemma with $q = \ell$ and $W = \ell$. Going through the argument again, it is easy to verify that this time the existence of a reducible pair is guaranteed whenever $|V(G) \setminus S| \leq (2\ell - 1)k$, that is when $|V(G)| \geq 2\ell k$.

We are now left with question (c) - the issue of how to *find* a reducible pair in polynomial time. Indeed, the proof of existence crucially relies on the knowledge of an (optimal) solution S. To find a reducible pair we use the linear programming relaxation of the ℓ-COC problem. We prove that an optimal solution to the LP-relaxation has to highlight every reducible pair (X, Y), essentially by always setting all the variables corresponding to X to 1 and the variables corresponding to Y to 0. For Vertex Cover (i.e 1-COC), the classic Nemhauser Trotter Theorem [18] implies that we may simply include all the vertices whose LP variable is set to 1 into the solution S. For ℓ-COC with $\ell \geq 2$ we are unable to prove the corresponding statement. We are however, able to prove that if a reducible pair (X, Y) exists, then X (essentially) has to be assigned 1 and Y (essentially) has to be assigned 0. We then give a polynomial time algorithm that extracts X and Y from the vertices assigned 1 and 0 respectively by the optimal linear programming solution. Together, the arguments (b) and (c) yield the kernel with $2\ell k$ vertices. We remark that to the best of our knowledge, after the kernel for Vertex Cover [3] our kernel is the first example of a kernelization algorithm based on linear programming relaxations.

Overview of the paper. In Section 2 we recall basic definitions and set up notations. The kernel for ℓ-COC is proved in Sections 3, 4 and 5. In Section 3 we prove the necessary adjustment of the results on Max-Min allocation of Bezáková and Dani [1] that is suitable to our needs. In Section 4 we state and prove our new Weighted Expansion Lemma, and in Section 5 we combine all our results to obtain the kernel.

2 Preliminaries

Let $\mathbb{N}$ denote the set of positive integers $\{0, 1, 2, \dots\}$. For any non-zero $t \in \mathbb{N}$, $[t] := \{1, 2, \dots, t\}$. We denote a constant function $f : X \to \mathbb{N}$ such that for all $x \in X, f(x) = c$, by $f = c$. For any function $f : X \to \mathbb{N}$ and a constant $c \in \mathbb{N}$, we define the function $f + c : X \to \mathbb{N}$ such that for all $x \in X, (f + c)(x) = f(x) + c$. We use the same symbol f to denote the restriction of f over a subset of it's domain, X. For a set $\{v\}$ containing a single element, we simply write v. A vertex $u \in V(G)$ is said to be incident on an edge $e \in E(G)$ if u is one of the endpoints of e. A pair of edges $e, e' \in E(G)$ are said to be adjacent if there is a vertex $u \in V(G)$ such that u is incident on both e and e'. For any vertex $u \in V(G)$, by $N(u)$ we denote the set of neighbors of u i.e. $N(u) := \{v \in V(G) \mid uv \in E(G)\}$. For any subgraph $X \subseteq G$, by $N(X)$ we denote the set of neighbors of vertices in X outside X, i.e. $N(X) := (\bigcup_{u \in X} N(u)) \setminus X$. A pair of vertices $u, v \in V(G)$ are called *twins* if $N(u) = N(v)$. An induced subgraph on $X \subseteq V(G)$ is denoted by $G[X]$.

A *path* P is a graph, denoted by a sequence of vertices $v_1 v_2 \dots v_t$ such that for any $i, j \in [t], v_i v_j \in E(P)$ if and only if $|i - j| = 1$. A *cycle* C is a graph, denoted either by a sequence of vertices $v_1 v_2 \dots v_t$ or by a sequence of edges $e_1 e_2 \dots e_t$, such that for any $i, j \in [t]$ $u_i u_j \in E(C)$ if and only if $|i - j| = 1 \mod t$ or in terms of edges, for any $i, j \in [t]$, e_i is adjacent to e_j if and only if $|i - j| = 1 \mod t$. The *length* of a path(cycle) is the number

of edges in the path(cycle). A *triangle* is a cycle of length 3. In G, for any pair of vertices $u, v \in V(G)$ $\mathsf{dist}(u, v)$ represents the length of a shortest path between u and v. A *tree* is a connected graph that does not contain any cycle. A *rooted tree* T is a tree with a special vertex r called the root of T. With respect to r, for any edge $uv \in E(T)$ we say that v is a child of u (equivalently u is parent of v) if $\mathsf{dist}(u, r) < \mathsf{dist}(v, r)$. A *forest* is a collection of trees. A *rooted forest* is a collection of rooted trees. A *clique* is a graph that contains an edge between every pair of vertices. A vertex cover of a graph is a set of vertices whose removal makes the graph edgeless.

Fixed Parameter Tractability. A *parameterized problem* Π is a subset of $\Sigma^* \times \mathbb{N}$. A parameterized problem Π is said to be *fixed parameter tractable*(FPT) if there exists an algorithm that takes as input an instance (I, k) and decides whether $(I, k) \in \Pi$ in time $f(k) \cdot n^c$, where n is the length of the string I, $f(k)$ is a computable function depending only on k and c is a constant independent of n and k.

A *kernel* for a parameterized problem Π is an algorithm that given an instance (T, k) runs in time polynomial in $|T|$, and outputs an instance (T', k') such that $|T'|, k' \leq g(k)$ for a computable function g and $(T, k) \in \Pi$ if and only if $(T', k') \in \Pi$. For a comprehensive introduction to FPT algorithms and kernels, we refer to the book by Cygan et al. [5].

A *data reduction rule*, or simply, reduction rule, for a parameterized problem Q is a function $\phi : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ that maps an instance (I, k) of Q to an equivalent instance (I', k') of Q such that ϕ is computable in time polynomial in $|I|$ and k. We say that two instances of Q are *equivalent* if $(I, k) \in Q$ if and only if $(I', k') \in Q$; this property of the reduction rule ϕ, that it translates an instance to an equivalent one, is referred as the *safeness* of the reduction rule.

3 Max-min Allocation

We will now view a bipartite graph $G := ((A, B), E)$ as a relationship between "customers" represented by the vertices in A and "items" represented by the vertices in B. If the graph is supplied with two functions $w_a : A \to \mathbb{N}$ and $w_b : B \to \mathbb{N}$, we treat these functions as a "demand function" and a "capacity" function, respectively. That is, we consider each item $v \in B$ to have value $w_b(v)$, and every customer $u \in A$ wants to be assigned items worth at least $w_a(u)$. An edge between $u \in A$ and $v \in B$ means that the item v can be given to u.

A weight function $f : E(G) \to \mathbb{N}$ describes an assignment of items to customers, provided that the items can be "divided" into pieces and the pieces can be distributed to different customers. However this "division" should not create more value than the original value of the items. Formally we say that the weight function *satisfies* the capacity constraint $w_b(v)$ of $v \in B$ if $\sum_{uv \in E(G)} f(uv) \leq w_b(v)$. The weight function satisfies the capacity constraints if it satisfies the capacity constraints of all items $v \in B$.

For each item $u \in A$, we say that f *allocates* $\sum_{uv \in E(G)} f(uv)$ value to u. The weight function f *satisfies* the demand $w_a(u)$ of $u \in A$ if it allocates at least $w_a(u)$ value to u, and f satisfies the demand constraints if it does so for all $u \in A$. In other words, the weight function satisfies the demands if every customer gets items worth at least her demand. The weight function f *over-satisfies* a demand constraint $w_a(u)$ of u if it allocates strictly more than $w_a(u)$ to u.

We will also be concerned with the case where items are indivisible. In particular we say that a weight function $f : E(G) \to \mathbb{N}$ is *unsplitting* if for every $v \in B$ there is at most one edge $uv \in E(G)$ such that $f(uv) > 0$. The essence of the next few lemmas is that if

we have a (splitting) weight function f of items whose value is at most W, and f satisfies the capacity and demand constraints, then we can obtain in polynomial-time an unsplitting weight function f' that satisfies the capacity constraints and violates the demand constraints by at most $(W-1)$. In other words we can make a splitting distribution of items unsplitting at the cost of making each customer lose approximately the value of the most expensive item.

Allocating items to customers in such a way as to maximize satisfaction is well studied in the literature. The lemmata 1 and 2 are very similar, both in statement and proof, to the work of Bezáková and Dani [1][Theorem 3.2], who themselves are inspired by Lenstra et al. [15]. However we do not see a way to directly use the results of Bezáková and Dani [1], because we need a slight strengthening of (a special case of) their statement.

▶ **Lemma 1.** *There exists a polynomial-time algorithm that given a bipartite graph G, a capacity function $w_b : B \to \mathbb{N}$, a demand function $w_a : A \to \mathbb{N}$ and a weight function $f : E(G) \to \mathbb{N}$ that satisfies the capacity and demand constraints, outputs a function $f' : E(G) \to \mathbb{N}$ such that f' satisfies the capacity and demand constraints and the graph $G_{f'} = (V(G), \{uv \in E(G) \mid f'(uv) > 0\})$ induced on the non-zero weight edges of G is a forest.*

Proof. We start with f and in polynomially many steps, change f into the required function f'. If $G_f = (V(G), \{uv \in E(G) \mid f(uv) > 0\})$ is a forest, then we return $f' = f$. Otherwise, suppose that G_f contains a cycle $C := e_1e_2e_3 \dots e_{2s}$. Proceed as follows. Without loss of generality, suppose $c = f(e_1) = min\{f(e) \mid e \in C\}$, and note that $c > 0$. Compute the edge weight function $f^\star : E \to \mathbb{R}$ defined as follows. For $e_i \in C$, we define $f^\star(e_i) = f(e) - c$ if i is odd, and define $f^\star(e_i) = f(e) + c$ if i is even. For $e \notin C$ we define $f^\star(e_i) = f(e)$.

Every vertex of G is incident to either 0 or exactly 2 edges of C. If the vertex v is incident to two edges of C then one of these edges, say e_{2i}, has even index in C, and the other, e_{2i+1} has odd. For the edge e_{2i} we have $f^\star(e_{2i}) = f(e_{2i}) + c$ and for e_{2i+1} we have $f^\star(e_{2i+1}) = f(e_{2i+1}) - c$. Thus we conclude that for all $v \in V(G)$, $\sum_{u \in N(v)} f^\star(uv) = \sum_{u \in N(v)} f(uv)$, and that therefore $f^\star$ satisfies the capacity and demand constraints. Furthermore at least one edge that is assigned non-zero weight by f is assigned 0 by $f^\star$ and $G_{f^\star} = (V(G), \{uv \in E(G) \mid f^\star(uv) > 0\})$ has one less cycle than G_f. For a polynomial-time algorithm, repeatedly apply the process described above to reduce the number of edges with non-zero weight, as long as $G_{f^\star}$ contains a cycle. ◀

▶ **Lemma 2.** *There exists a polynomial-time algorithm with the following specifications. It takes as input a bipartite graph $G := ((A, B), E)$, a demand function $w_a : A \to \mathbb{N}$, a capacity function $w_b : B \to \mathbb{N}$, an edge weight function $f : E(G) \to \mathbb{N}$ that satisfies both the capacity and demand constraints, and a vertex $r \in A$. The algorithm outputs an unsplitting edge weight function $h : E(G) \to \mathbb{N}$ that satisfies the capacity constraints, satisfies the demands $w'_a = w_a - (W-1)$ where $W = \max_{v \in B} w_b(v)$, and additionally satisfies the demand $w_a(r)$ of r.*

Proof. Without loss of generality the graph $G_f := (V(G), \{uv \in E(G) \mid f(uv) > 0\})$ is a forest. If it is not, we may apply Lemma 1 to f, and obtain a function f' that satisfies the capacity and demand constraints, and such that $G_{f'} = (V(G), \{uv \in E(G) \mid f'(uv) > 0\})$ is a forest. We then rename f' to f. By picking a root in each connected component of G_f we may consider G_f as a rooted forest. We pick the roots as follows, if the component contains the special vertex r, we pick r as root. If the component does not contain r, but contains at least one vertex $u \in A$, we pick that vertex as the root. If the component does not contain any vertices of A then it does not contain any edges and is therefore a single vertex in B, we

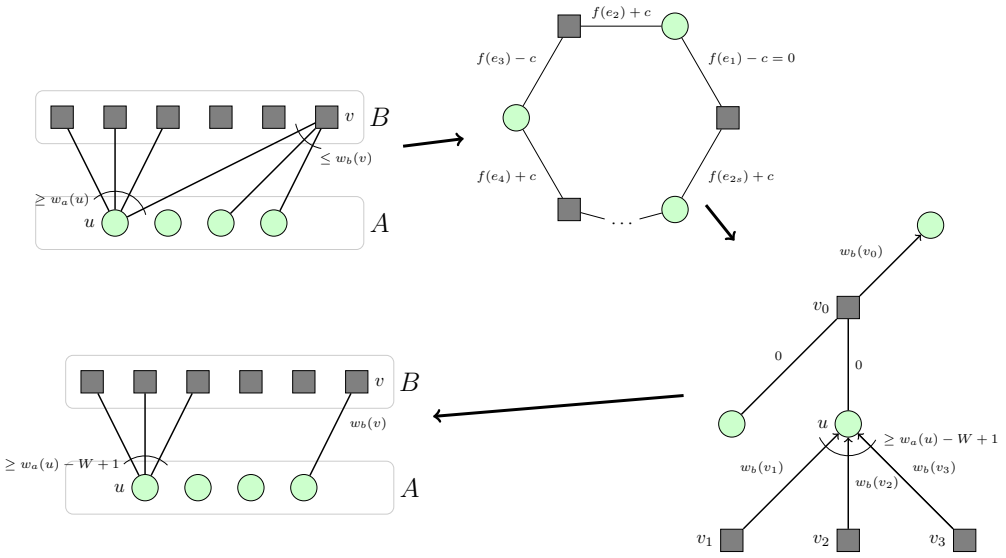


Figure 1 Proof of Lemma 1 and 2. Cyclically shift smallest weight in a non-zero weight cycle to obtain a forest. Root each tree in the forest at a vertex in A such that each vertex in B has a parent in A. Assign the value of $v \in B$ to its parent $u \in A$. In this new assignment, a non-root vertex $u \in A$ *loses* its parent $v_0 \in B$ and $f(v_0u) \leq W-1$ which explains the cost of making a splitting assignment into an unsplitting assignment.

pick that vertex as root. Thus, every item $v \in B$ that is incident to at least one edge in G_f has a unique *parent* $u \in A$ in the forest G_f. We define the new weight function h. For every edge $uv \in E(G)$ with $u \in A$ and $v \in B$ we define $h(uv)$ as follows.

$$h(uv) = w_b(v) \text{ if } u \text{ is the parent of } v \text{ in } G_f, \text{ and } h(uv) = 0 \text{ otherwise.}$$

Clearly h is unsplitting and satisfies the capacity constraints. We now prove that h also satisfies the demand constraints w'_a and satisfies the demand constraint $w_a(r)$ of r. Consider the demand constraint $w'_a(u)$ for an arbitrary customer $u \in A$. There are two cases, either u is the root of the component of G_f or it is not. If u is the root, then for every edge $uv \in E(G)$ such that $f(uv) > 0$ we have that $uv \in E(G_f)$ and consequently that u is the parent of v. Hence $h(uv) = w_b(v) \geq f(uv)$, and therefore h satisfies the demand $w_a(u)$ of u. Since $w_a(u) \geq w'_a(u)$, we have that h satisfies the demand $w'_a(u)$. Furthermore, since r is the root of its component this also proves that h satisfies the demand $w_a(r)$.

Consider now the case that u is not the root of its component in G_f. Then u has a unique parent in G_f, call this vertex $v^\star \in B$. We first prove that $f(uv^\star) \leq w_b(v^\star) - 1$. Indeed, since $v^\star$ is incident to the edge $uv^\star$ we have that $v^\star$ has a parent $u^\star$ in G_f, and that $u^\star \neq u$ because $v^\star$ is the parent of u. We have that $f(u^\star v^\star) + f(uv^\star) \leq w_b(v^\star)$ and that $f(u^\star v^\star) \geq 1$, because $u^\star v^\star$ is an edge in G_f. It follows that $f(uv^\star) \leq w_b(v^\star) - 1$. We now proceed to proving that h satisfies the demand $w'_a(u)$.

For every edge $uv \in E(G) \setminus \{uv^\star\}$ such that $uv \in E(G)$ such that $f(uv) > 0$ we have that $uv \in E(G_f)$ and consequently that u is the parent of v. Hence we have that $h(uv) = w_b(v) \geq f(uv)$. Furthermore $h(uv^\star) = 0$ while $f(uv^\star) \leq w_b(v^\star) - 1 \leq W - 1$. Therefore h satisfies the demand $w'_a(u)$. ◀

4 The Weighted Expansion Lemma

Our kernelization algorithm will use "q-expansions" in bipartite graphs, a well known tool in kernelization [5]. We begin by stating the definition of a q-expansion and review the facts about them that we will use.

▶ **Definition 3** (q-expansion)**.** Let $G := ((A, B), E)$ be a bipartite graph. We say that A has q-expansion into B if there is a family of sets $\{V_a \mid V_a \subseteq N(a), |V_a| \geq q, a \in A\}$ such that for any pair of vertices $a_i, a_j \in A$,$i \neq j$, $V_{a_i} \cap V_{a_j} = \emptyset$.

▶ **Definition 4** (Twin graph)**.** For a bipartite graph $G := ((A, B), E)$ with a weight function $w_b : B \to \mathbb{N}$, the twin graph $T_{AB} := (A, B')$ of G is obtained as follows: B' contains $|w_b(v)|$ twins of every vertex $v \in B$ i.e. $B' := \{v_1, v_2, \ldots v_{w_b(v)} \mid v \in B\}$ and edges in T_{AB} such that for all $v \in B$ and $i \in [w_b(v)], N(v_i) = N(v)$ i.e. $E(T_{AB}) := \{av_i | a \in A, v_i \in B', v \in B, av \in E(G)\}$.

▶ **Lemma 5** ([5])**.** *Let G be a bipartite graph with bipartition (A, B). Then there is a q-expansion from A into B if and only if $|N(X)| \geq q|X|$ for every $X \subseteq A$. Furthermore, if there is no q-expansion from A into B, then a set $X \subseteq A$ with $|N(X)| < q|X|$ can be found in polynomial-time.*

▶ **Lemma 6** (Expansion Lemma [5])**.** *Let $q \geq 1$ be a positive integer and G be a bipartite graph with vertex bipartition (A, B) such that $|B| \geq q|A|$, and there are no isolated vertices in B. Then there exist nonempty vertex sets $X \subseteq A$ and $Y \subseteq B$ such that there is a q-expansion of X into Y, and no vertex in Y has a neighbor outside X, i.e. $N(Y) \subseteq X$. Furthermore, the sets X and Y can be found in time polynomial in the size of G.*

▶ **Lemma 7** (Folklore)**.** *There exists a polynomial-time algorithm that given a bipartite graph $G := ((A, B), E)$ and an integer q decides (and outputs in case yes) if there exist sets $X \subseteq A, Y \subseteq B$ such that there is a q-expansion of X into Y.*

Proof. We describe a recursive algorithm. If $A = \emptyset$ or $B = \emptyset$, then output NO and terminate. Otherwise, construct the twin graph T_{BA} with weight function $w : A \to \mathbb{N}$ where for all $u \in A, w(u) = q$ and let M be a maximum matching in T_{BA}. Consider the graph $G' := (A, B)$ with edge set $E(G') := \{uv, u \in A, v \in B \mid u_i v \in M\}$. Let $A' \subseteq A$ such that for all $u \in A', d_{G'}(u) \geq q$ and let $B' \subseteq B$ such that $B' := \bigcup_{u \in A'} N_{G'}(u)$. If $N(B') \subseteq A'$, then return (A', B') and terminate. Otherwise, recurse on $G[A' \cup (B \setminus N_G(A \setminus A'))]$.

If there are no sets X, Y such that there is a q-expansion of X into Y, then for any pair of sets $A' \subseteq A, B' \subseteq B$ either $N(B') \setminus A' \neq \emptyset$ or $|B'| < q|A'|$. Since at each recursive step, the size of the graph with which the algorithm calls itself decreases, eventually either A' becomes empty or $B \setminus N_G(A \setminus A')$ becomes empty. Hence, the algorithm outputs no. Now we need to show that if there exist sets (A^*, B^*) such that there is a q-expansion of A^* into B^*, then at each recursive call, we have that $A^* \subseteq A$ and $B^* \subseteq B$. At the start of the algorithm, $A^* \subseteq A$ and $B^* \subseteq B$. Since $N(B^*) \subseteq A^*$ and for all $u \in A^*$ $d_G(u) \geq q$, we have that $A^* \cup B^* \subseteq V(G')$. If $N(B') \subseteq A'$, then the algorithm of Lemma 6 when run on G', q will output (A^*, B^*). Note that $B^* \subseteq B'$. At the recursive step, $A^* \subseteq A'$ and since $B^* \cap N_G(A \setminus A') = \emptyset$, we have that $B^* \subseteq B' \setminus N_G(A \setminus A')$. Hence, $G[A^* \cup B^*]$ is a subgraph of $G[A' \cup (B \setminus N_G(A \setminus A'))]$ which concludes the correctness of the algorithm. Since at each recursive call the size of the graph decreases by at least 1, the total time taken by the above algorithm is polynomial in n. ◀

One may think of a q-expansion in a bipartite graph with bipartition (A, B) as an allocation of the items in B to each customer in A such that every customer gets at least q items. For our kernel we will need a generalization of q-expansions to the setting where the items in B have different values, and every customer gets items of total value at least q.

▶ **Definition 8** (Weighted q-expansion)**.** Let $G := ((A, B), E)$ be a bipartite graph with capacity function $w_b : B \to \mathbb{N}$. Then, a weighted q-expansion in G is an edge weight function $f : E(G) \to \mathbb{N}$ that satisfies the capacity constraints w_b and also satisfies the demand constraints $w_a = q$. For an integer $W \in \mathbb{N}$, the q-expansion f is called a *W-strict* q-expansion if f allocates at least $q + W - 1$ value to at least one vertex r in A, and in this case we say that f is W-strict at r. Further, a q-expansion f is *strict* (at r) if it is 1-strict (at r). If f is unsplitting we call f an *unsplitting* q-expansion.

▶ **Lemma 9.** *There exists a polynomial-time algorithm that given a bipartite graph $G := ((A, B), E)$, an integer q and a capacity function $w_b : B \to \mathbb{N}$ outputs (if it exist) two sets $X \subseteq A$ and $Y \subseteq B$ along with a weighted q-expansion in $G[X \cup Y]$ such that $N(Y) \subseteq X$.*

Proof. Construct the twin graph $T_{AB} := (A, B')$ of G. Run the algorithm of Lemma 7 with input T_{AB}, q that outputs sets $X \subseteq A$ and $Y' \subseteq B'$ such that X has q-expansion into Y' and $N(Y') \subseteq X$. Consider the set $Y := \{v \in B \mid v_i \in Y'\}$. Define a weight function $f : E(G[X \cup Y]) \to \mathbb{N}$ as follows: for all $uv \in E(G[X \cup Y])$ $f(uv) = |\{v_i \in Y' | v_i \text{ matched to } u\}|$.

Clearly, $N(Y) \subseteq X$. Now we claim that f is a weighted q-expansion in $G[X \cup Y]$ with capacity function w_b and demand function $w_a = q$. For any vertex $u \in A$, there are at least q vertices in Y' are matched to u. Hence for all $u \in A$, we have that $\sum_{v \in N(u)} f(uv) \geq q = w_a$. At the same time, for any vertex $v \in B$, there are at most $w_b(v)$ copies of v in Y'. Therefore, for all $v \in Y$ we have $\sum_{u \in N(v)} f(uv) \leq w_b(v)$. ◀

▶ **Lemma 10.** *There exists a polynomial-time algorithm that given a weighted q-expansion $f : E(G) \to \mathbb{N}$ in $G := ((A, B), E)$, a capacity function $w_b : B \to \mathbb{N}$ and an integer W such that $W = \max_{e \in E(G)} f(e)$ outputs an unsplitting W-strict weighted $(q - W + 1)$-expansion in G.*

Proof. Run the algorithm of Lemma 2 with inputs $G, f, w_a = q, w_b, W$ and a vertex $u \in A$. In case f is strict, u is the vertex r that makes f strict. Let the function $h : E(G) \to \mathbb{N}$ be the output of Lemma 2. Now h is an unsplitting edge weight function that satisfies the capacity constraints, satisfies the demands $q - W + 1$, and additionally satisfies the demand q of u. Hence, h is the required unsplitting weighted W-strict $(q - W + 1)$-expansion in G. ◀

▶ **Lemma 11** (Weighted Expansion Lemma)**.** *Let $q, W \geq 1$ be positive integers and G be a bipartite graph with vertex bipartition (A, B) and $w_b : B \to \{1, \ldots, W\}$ be a capacity function such that $\sum_{v \in B} w_b(v) \geq (q + W - 1) \cdot |A|$, and there are no isolated vertices in B. Then there exist nonempty vertex sets $X \subseteq A$ and $Y \subseteq B$ such that $N(Y) \subseteq X$ and there is an unsplitting weighted W-strict q-expansion of X into Y. Furthermore, the sets X and Y can be found in time polynomial in the size of G.*

Proof. Construct the twin graph T_{AB} from G and w_b, the bipartition of T_{AB} is (A, B'). Now, obtain using the Expansion Lemma 6 with $q' = q + W - 1$ on T_{AB} sets $X \subseteq A$ and $Y' \subseteq B'$, such that $N(Y') \subseteq X$ and there is a $(q + W - 1)$-expansion from X to Y' in T_{AB}.

Let $Y := \{v \in B \mid v_i \in Y'\}$ (here the $v_i \in Y'$ are as in Definition 4). Then $N(Y) \subseteq X$ and the $(q + W - 1)$-expansion from X to Y' in T_{AB} immediately yields a weighted $(q + W - 1)$-expansion f from X to Y in G. Applying Lemma 10 on $G[X \cup Y]$ using the weighted $(q + W - 1)$-expansion f proves the statement of the lemma. ◀

5 Obtaining the Linear Kernel

▶ **Definition 12.** For a graph G and a pair of vertex-disjoint sets $X, Y \subseteq V(G)$, we define the weighted graph $\tilde{G}_{XY}$ as follows: $V(\tilde{G}_{XY}) := X \cup \tilde{Y}$ such that there is a bijection $h : cc(G[Y]) \to \tilde{Y}$ where $cc(G[Y])$ is the set of connected components of $G[Y]$. $E(\tilde{G}_{XY}) := \{xc \mid x \in X, c \in \tilde{Y}, c = h(C) \text{ and } x \in N_G(C)\}$. We also define a weight function $w : \tilde{Y} \to \mathbb{N}$ such that for all $c \in \tilde{Y}, w(c) = |h^{-1}(c)|$.

▶ **Definition 13** (Reducible Pair)**.** For a graph G, a pair of vertex-disjoint sets (X, Y) where $X, Y \subseteq V(G)$ is called a (strict) reducible pair if $N(Y) \subseteq X$, the size of every component in $G[Y]$ is at most ℓ, and there exists a (strict) weighted $(2\ell - 1)$-expansion in $\tilde{G}_{XY}$.

▶ **Definition 14.** A reducible pair (X, Y) is called minimal if there is no reducible pair (X', Y') such that $X' \subset X$ and $Y' \subseteq Y$.

▶ **Lemma 15.** *There exists a polynomial-time algorithm that given an ℓ-COC instance (G, k) together with a vertex-disjoint set pair $A, B \subseteq V(G)$ outputs (if it exists) a reducible pair (X, Y) where $X \subseteq A$ and $Y \subseteq B$.*

Proof. Construct $\tilde{G}_{AB} := (A, \tilde{B})$ and run the algorithm of Lemma 9 with input $\tilde{G}_{AB}, w, q = 2\ell - 1$ which outputs sets $X \subseteq A$ and $Y' \subseteq \tilde{B}$ (if it exists) along with a weighted $(2\ell - 1)$-expansion of X into Y' such that $N(Y') \subseteq X$. Now from Y' we obtain the set $Y := \bigcup_{y \in Y'} h^{-1}(y)$. Clearly, $N(Y) \subseteq X$ and hence, (X, Y) is the desired reducible pair. ◀

▶ **Lemma 16.** *Given an ℓ-COC instance (G, k), if $|V(G)| \geq 2\ell k$ and (G, k) is a* YES*-instance, then there exists a reducible pair (X, Y).*

Proof. Without loss of generality, we can assume that G is a connected graph. Let S be an ℓ-COC solution of size at most k. Clearly, $|V \setminus S| \geq (2\ell - 1)k$. We define $A := S$ and $B := V \setminus S$ and construct $\tilde{G}_{AB} = (A, \tilde{B})$. We have the weight function $w_b : \tilde{B} \to \mathbb{N}$ such that for all $v \in \tilde{B}, w_b(v) = |h^{-1}(v)| \leq \ell$, as the size of components in $G[V \setminus S]$ is at most ℓ. We have that $\sum_{v \in \tilde{B}} w_b(v) \geq (2\ell - 1)|A|$ and there are no isolated vertices in $\tilde{B}$. Hence, (A, B) is the desired reducible pair. ◀

▶ **Lemma 17.** *Let (X, Y) be a reducible pair. Then, there exists a partition of $X \cup Y$ into $C_1, ..., C_{|X|}$ such that (i) for all $u_i \in X$, we have $u_i \in C_j$ if and only if $i = j$, (ii) for all $i \in [|X|]$, $|C_i| \geq \ell + 1$, (iii) for every component C in $G[Y]$, there exists a unique C_i such that $V(C) \subseteq C_i$ and $u_i \in N(C)$ and (iv) if (X, Y) is a strict reducible pair, then there exists C_j such that $|C_j| \geq 2\ell + 1$.*

Proof. Construct $\tilde{G}_{XY} := (X, \tilde{Y})$. Run the algorithm of Lemma 10 with input $\tilde{G}_{XY}, q = 2\ell - 1$, and $W = \ell$(as the capacity of any vertex in $\tilde{Y}$ is at most ℓ) which outputs an unsplitting weighted ℓ-expansion f' in $\tilde{G}_{XY}$. In polynomial time, we modify f' such that if there is a vertex $v \in \tilde{Y}$ such that $\forall u \in N(v), f'(uv) = 0$, we choose a vertex $u \in N(v)$ and set $f'(uv) = w_b(v)$. For each $u_i \in X$ define $C_i := u_i \bigcup_{f'(u_i v) \neq 0} h^{-1}(v)$. Since f' is unsplitting, the collection $C_1, \ldots, C_{|X|}$ forms a partition of $X \cup Y$. By the definition of C_i, we have that for any $u_i \in X$, $u_i \in C_j$ if and only if $i = j$. For any component C in $G[Y]$, $h(C)$ is matched to a unique vertex $u_i \in X$ by f', we have that $V(C) \subseteq C_i$. As f' is a weighted ℓ-expansion, $|C_i| = 1 + \sum_{f'(u_i v) \neq 0} |h^{-1}(v)| = 1 + \sum_{f'(u_i v) \neq 0} f'(u_i v) \geq 1 + \ell$. Let (X, Y) be strict at $u_j \in X$. Then, we can use Lemma 10 to obtain the expansion f' such that it is strict at u_j. Hence, $|C_j| = 1 + \sum_{f'(u_j v) \neq 0} |h^{-1}(v)| = 1 + \sum_{f'(u_j v) \neq 0} f'(u_j v) > 1 + \ell + (\ell - 1)$ which implies $|C_j| \geq 2\ell + 1$. This concludes the proof of the lemma. ◀

▶ **Lemma 18.** *Let (X, Y) be a reducible pair. If (G, k) is a* YES*-instance for ℓ-COC, then there exists an ℓ-COC solution S of size at most k such that $X \subseteq S$ and $S \cap Y = \emptyset$.*

Proof. By Lemma 17 we have that there are $C_1, \ldots, C_{|X|} \subseteq X \cup Y$ vertex disjoint sets of size at least $\ell + 1$ such that for all $i \in [|X|]$, $G[C_i]$ is a connected set. Let S' be an arbitrary solution. Then, S' must contain at least one vertex from each C_i. Let $S := S' \setminus (X \cup Y) \cup X$. We have that $|S| \leq |S'| - |X| + |X| = |S'|$. As any connected set of size $\ell + 1$ that contains a vertex in Y also contains a vertex in X and $X \subseteq S$, S is also an ℓ-COC solution. ◀

Now we encode an ℓ-COC instance (G, k) as an Integer Linear Programming instance. We introduce $n = |V(G)|$ variables, one variable x_v for each vertex $v \in V(G)$. Setting the variable x_v to 1 means that v is in S, while setting $x_v = 0$ means that v is not in S. To ensure that S contains a vertex from every connected set of size $\ell + 1$, we can introduce constraints $\sum_{v \in C} x_v \geq 1$ where C is a connected set of size $\ell + 1$. The size of S is given by $\sum_{v \in V(G)} x_v$. This gives us the following ILP formulation:

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V(G)} x_v, & \\ \text{subject to} & \sum_{v \in C} x_v \geq 1 & \text{for every connected set } C \text{ of size } \ell + 1 \\ & 0 \leq x_v \leq 1 & \text{for every } v \in V(G) \\ & x_v \in \mathbb{Z} & \text{for every } v \in V(G). \end{array}$$

Note that there are $n^{\mathcal{O}(\ell)}$ connected sets of size at most ℓ in a graph on n vertices. Hence, providing an explicit ILP requires $n^{\mathcal{O}(\ell)}$ time which forms the bottleneck for the runtime of the kernelization algorithm that follows. We consider the Linear Programming relaxation of above ILP obtained by dropping the constraint that $x \in \mathbb{Z}$. By an optimal LP solution S_L with weight L we mean the set of values assigned to each variable, and optimal value is L. For a set of vertices $X \in V(G)$, $X = 1$ ($X = 0$) denotes that every variable corresponding to vertices in X is set to 1 (0).

▶ **Lemma 19.** *Let S_L be an optimal LP solution for G such that $x_v = 1$ for some $v \subseteq V(G)$. Then, $S_L - x_v$ is an optimal LP solution for $G - v$ of value $L - 1$.*

Proof. Clearly, $S_L - x_v$ is feasible solution for $G - v$ of value $L - 1$. Suppose it is not optimal. Let $S_{L'}$ be an optimal LP solution for $G - v$ such that $L' < L - 1$. Then, $S_{L'} \cup x_v$ with $x_v = 1$ is an optimal LP solution for G with value $< L - 1 + 1 = L$ contradicting that the optimal solution value of LP for G is L. ◀

From now on by running LP after setting $x_v = 1$ for some vertex v, we mean running the LP algorithm for $G - v$ and including $x_v = 1$ in the obtained solution to get a solution for G.

▶ **Lemma 20.** *Let (X, Y) be a strict reducible pair. Then every optimal LP solution sets at least one variable corresponding to a vertex in X to 1.*

Proof. By Lemma 18, we have that every connected set of size $\ell + 1$ in $G[X \cup Y]$ contains a vertex in X. Hence, from any LP solution S_L, a feasible LP solution can be obtained by setting $X = 1$ and $Y = 0$. Since, we have at least $|X|$ many vertex disjoint LP constraints, for each $v_i \in X$, we have $\sum_{u \in C_i} x_u = 1$. By Lemma 17, there is a set $C_j \subseteq X \cup Y$ such that $|C_j| \geq 2\ell + 1$. If $x_{v_j} \neq 1$, then there is a vertex $w \in C_j$ such that $x_w > 0$. Let $w \in C \subset C_j$ where $G[C]$ is a connected component in $G[Y]$. Since $|C| \leq \ell$, there is a connected set C' of size at least $\ell + 1$ in $G[C_j] - C$. But now $\sum_{u \in C'} x_u < 1$ contradicting that S_L is feasible. ◀

▶ **Lemma 21.** *Let (X, Y) be a minimal reducible pair. If for any vertex $v \in X$, an optimal LP solution sets $x_v = 1$, then it also sets $X = 1$ and $Y = 0$.*

Proof. We prove the lemma by contradiction. Let $X' \subset X$ be the largest subset of X such that $X' = 1$. Consider $\tilde{G}_{XY}$. Let $Y' \subseteq \tilde{Y}$ be the set of vertices such that $N(Y') \subseteq X'$. Let $Z := \bigcup_{v \in Y'} h^{-1}(v)$. By the minimality of (X, Y), we have that $\sum_{v \in Y'} w(v) < (2\ell - 1)|X'|$. Hence, $\sum_{v \in \tilde{Y} \setminus Y'} w(v) > (2\ell - 1)|X \setminus X'|$. Clearly, the weighted $(2\ell - 1)$-expansion in the reducible pair (X, Y) when restricted to $(X \setminus X', Y \setminus Z)$ provides a weighted $(2\ell-1)$-expansion of $X \setminus X'$ into $Y \setminus Z$. This implies that $(X \setminus X', Y \setminus Z)$ is a strict reducible pair in $G-(X' \cup Z)$. By Lemma 19, we have that the LP solution restricted to $G - (X' \cup Z)$ is optimal. Since $(X \setminus X', Y \setminus Z)$ is a strict reducible pair, by Lemma 20, there is a vertex $u \in X \setminus X'$ such that $x_u = 1$, but this contradicts the maximality of X'. Therefore, if for any vertex $v \in X$, an LP solution sets $x_v = 1$, then it sets $X = 1$ and $Y = 0$. ◀

▶ **Lemma 22.** *There exists a polynomial time algorithm that given an integer ℓ and ℓ-COC instance (G, k) on at least $2\ell k$ vertices either finds a reducible pair (X, Y) or concludes that (G, k) is a* NO*-instance.*

Proof. If (G, k) is a YES-instance of ℓ-COC, then by Lemma 16, there exists a reducible pair (X, Y). We use the following algorithm to find one:

Step 1. Run the LP algorithm. Let $A = 1$ and $B = 0$ in the LP solution.

Step 2. If both A and B are non-empty, then run the algorithm of Lemma 15 with input $(G, k), A, B$. If it outputs a reducible pair (X, Y), then return (X, Y) and terminate. Otherwise, go to step 3.

Step 3. Now we do a linear search for a vertex in X. For each vertex $v \in V(G)$, do the following: in the original LP introduce an additional constraint that sets the value of the variable x_v to 1 i.e. $x_v = 1$ and run the LP algorithm. If the optimal value of the new LP is the same as the optimal value of the original LP, then let $A = 1$ and $B = 0$ be the sets of variables set to 1 and 0 respectively in the optimal solution of the new LP and go to step 2.

Step 4. Output a trivial NO-instance.

Step 1 identifies the set of variables set to 1 and 0 by the LP algorithm. By Lemma 21, we have that if there is a minimal reducible pair (X, Y) in G, then $X \subseteq A$ and $Y \subseteq B$. So, in Step 2 if the algorithm succeeds in finding one, we return the reducible pair and terminate otherwise we look for a potential vertex in X and set it to 1. If (X, Y) exists, then for at least one vertex, setting $x_v = 1$ would set $X = 1$ and $Y = 0$ (by Lemma 21) without changing the LP value and we go to Step 2 to find it. If for each choice of $v \in V(G)$, the LP value changes when x_v is set to 1, we can conclude that there is no reducible pair and output a trivial NO instance. Since, we need to do this search at most n times and each step takes only polynomial time, the total time taken by the algorithm is polynomial in the input size. ◀

▶ **Theorem 23.** *For every constant $\ell \in \mathbb{N}$, ℓ-*Component Order Connectivity *admits a kernel with at most $2\ell k$ vertices that takes $n^{\mathcal{O}(\ell)}$ time.*

References

1 Ivona Bezáková and Varsha Dani. Allocating indivisible goods. *SIGecom Exchanges*, 5(3):11–18, 2005.

2 Maw-Shang Chang, Li-Hsuan Chen, Ling-Ju Hung, Peter Rossmanith, and Ping-Chen Su. Fixed-parameter algorithms for vertex cover p3. *Discrete Optimization*, 19:12–22, 2016. doi:10.1016/j.disopt.2015.11.003.

3 Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001. doi:10.1006/jagm.2001.1186.

4 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010.

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

6 Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.

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

8 Pål Grønås Drange, Markus Sortland Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. In *Algorithms and Computation – 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*, pages 285–297, 2014. doi:10.1007/978-3-319-13075-0_23.

9 Fedor V. Fomin, Serge Gaspers, Dieter Kratsch, Mathieu Liedloff, and Saket Saurabh. Iterative compression and exact algorithms. *Theor. Comput. Sci.*, 411(7-9):1045–1053, 2010.

10 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5), 2009.

11 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

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

13 Frantisek Kardos, Ján Katrenic, and Ingo Schiermeyer. On computing the minimum 3-path vertex cover and dissociation number of graphs. *Theor. Comput. Sci.*, 412(50):7009–7017, 2011. doi:10.1016/j.tcs.2011.09.009.

14 Stefan Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113, 2014.

15 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990.

16 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.

17 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization–preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond*, pages 129–161. Springer, 2012.

18 George L. Nemhauser and Leslie E. Trotter Jr. Properties of vertex packing and independence system polyhedra. *Math. Program.*, 6(1):48–61, 1974. doi:10.1007/BF01580222.

19 J. M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7(3):425–440, 1986.

20 Jianhua Tu. A fixed-parameter algorithm for the vertex cover p_3 problem. *Inf. Process. Lett.*, 115(2):96–99, 2015.

21 Jianhua Tu and Wenli Zhou. A factor 2 approximation algorithm for the vertex cover p3 problem. *Inf. Process. Lett.*, 111(14):683–686, July 2011. doi:10.1016/j.ipl.2011.04.009.

22 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

23 Mingyu Xiao and Shaowei Kou. Faster computation of the maximum dissociation set and minimum 3-path vertex cover in graphs. In *Frontiers in Algorithmics – 9th International Workshop, FAW 2015, Guilin, China, July 3-5, 2015, Proceedings*, pages 282–293, 2015. doi:10.1007/978-3-319-19647-3_26.

24 Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. In *Algorithms and Computation – 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, pages 328–338, 2013.

25 Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM J. Comput.*, 10(2):310–327, 1981. doi:10.1137/0210022.

Structural Parameterizations of Feedback Vertex Set

Diptapriyo Majumdar

The Institute of Mathematical Sciences, Chennai, HBNI, India
diptapriyom@imsc.res.in

Abstract

A feedback vertex set in an undirected graph is a subset of vertices whose removal results in an acyclic graph. It is well-known that the problem of finding a minimum sized (or k-sized in case of decision version of) feedback vertex set (FVS) is polynomial time solvable in (sub)-cubic graphs, in pseudo-forests (graphs where each component has at most one cycle) and mock-forests (graphs where each vertex is part of at most one cycle). In general graphs, it is known that the problem is NP-complete, and has an $\mathcal{O}^*((3.619)^k)$ fixed-parameter algorithm and an $\mathcal{O}(k^2)$ kernel where k, the solution size, is the parameter. We consider the parameterized and kernelization complexity of feedback vertex set where the parameter is the size of some structure in the input. In particular, we show that

- FVS is fixed-parameter tractable, but is unlikely to have polynomial sized kernel when parameterized by the number of vertices of the graph whose degree is at least 4. This answers a question asked in an earlier paper.
- When parameterized by k, the number of vertices, whose deletion results in a pseudo-forest, we give an $\mathcal{O}(k^6)$ vertices kernel improving from the previously known $\mathcal{O}(k^{10})$ bound.
- When parameterized by the number k of vertices, whose deletion results in a mock-d-forest, we give a kernel consisting of $\mathcal{O}(k^{3d+3})$ vertices and prove a lower bound of $\Omega(k^{d+2})$ vertices (under complexity theoretic assumptions). Mock-d-forest for a constant d is a mock-forest where each component has at most d cycles.

1998 ACM Subject Classification G.2.2 Graph Algorithms, I.1.2 Analysis of Algorithms

Keywords and phrases Parameterized Complexity, Kernelization, Feedback Vertex Set, Structural Parameterization

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.21

1 Introduction

In the early years of parameterized complexity and algorithms, problems were almost always parameterized by solution size. Recent research has focused on other parameterizations based on structural properties of the input [16, 9, 15, 8], above or below guaranteed optimum values [14]. Such 'non-standard' parameters are known to be small in practice. Also once a problem is shown to be fixed-parameter tractable (and/or having a polynomial kernel) with respect to a parameter, it is a natural question whether it has a fixed-parameter algorithm or polynomial kernel with respect to a smaller parameter. Similarly, when a problem is W-hard or has no polynomial kernel then it is interesting to ask whether it is fixed-parameter tractable or admits a polynomial kernel when it is parameterized by a *structurally larger* parameter. We study such ecology of parameterization for Feedback Vertex Set.

Feedback Vertex Set in an undirected graph G asks whether G has a subset S of at most k vertices such that $G \setminus S$ is a forest, for a given integer k. The set S is called a feedback vertex set of the graph. The problem is known to be NP-complete even on

bipartite graphs [13] and in graphs of degree at most 4 [20], but is polynomial time solvable in sub-cubic graphs [22, 6, 5], asteroidal triple free graphs [19] and chordal bipartite graphs [17]. The problem is easy (polynomial time) to solve in pseudo-forests (graphs in which each component has at most one cycle), in mock-forests (graphs where each vertex is part of at most one cycle), in cliques and disjoint union of cliques. This is also one of the well-studied problems in parameterized complexity and when parameterized by solution size, it has an algorithm with running time $\mathcal{O}^*(3.619^k)$ [18][1] and a kernel with $\mathcal{O}(k^2)$ vertices and edges [21].

Some parameterizations by the size of some structure in the input have already been explored. Feedback Vertex Set parameterized by the size of maximum induced matching (also maximum independent set and vertex clique cover) has been shown to be $W[1]$-Hard but contained in XP (See [16, 1]). Bodlaender et al. [2] proved that Feedback Vertex Set parameterized by deletion distance to a cluster graph (disjoint union of cliques) has no polynomial kernel unless NP $\subseteq$ coNP/poly. Jansen et al. [16] give a survey of results known for such structural parameterization of feedback vertex set and show that

- Feedback Vertex Set parameteirzed by Deletion Distance to chordal graph is fixed-parameter tractable;
- Feedback Vertex Set parameterized by Deletion Distance to Pseudo-forest has an $\mathcal{O}(f^{10})$ kernel and a kernel lower bound of $\Omega(f^4)$ where f is the size of the deletion distance to pseudo-forest of the input graph.
- Feedback Vertex Set parameterized by Deletion Distance to mock-forest has no polynomial kernel unless NP $\subseteq$ coNP/poly.

Our Results: Jansen et al. suggested in [16], *"An interesting question in this direction is whether* Feedback Vertex Set *is XP or FPT when parameterized by the vertex deletion distance to sub-cubic graphs or alternatively, parameterized by the number of vertices of degree more than* 3". While the first question remains open, our first result is an answer to the latter question (FVS-High-Degree defined below). We answer it positively by providing a fixed-parameter algorithm running in time $\mathcal{O}^*(2^k)$. We also prove that this problem has no polynomial kernel unless NP $\subseteq$ coNP/poly.

FVS-High-Degree **Parameter:** k
Input: An undirected graph G such that $|\{u \in V(G) | deg_G(u) > 3\}| \leq k$ and $\ell \in \mathbb{N}$.
Question: Does G have a feedback vertex set of size at most ℓ?

Our next result is an improved kernel for the following problem for which an $\mathcal{O}(k^{10})$ vertex kernel and a conditional lower bound of $\Omega(k^4)$ were given by Jansen et al. [16].

FVS-Pseudo-Forest **Parameter:** k
Input: An undirected graph G, $S \subseteq V(G)$ of size at most k such that $G[V(G) \setminus S]$ is a graph in which every component has at most one cycle and an integer ℓ.
Question: Does G have a feedback vertex set of size at most ℓ?

We give a kernel on $\mathcal{O}(k^6)$ vertices, narrowing the gap between upper and lower bound for the size of the kernel. Note that every feedback vertex set is also a pseudo-forest deletion set, but not all pseudo-forest deletion sets are feedback vertex sets. So, in this problem, our parameter is smaller than the solution size.

[1] $\mathcal{O}^*$ notation suppresses polynomial factors.

Finally, we consider a variation of mock-forests (called mock-d-forest) where each component has at most d cycles, where d is a constant, and consider the kernelization complexity of FVS parameterized by the deletion distance to mock-d-forests. It is easy to see that FVS is fixed-parameter tractable when parameterized by the deletion distance to mock-d-forest (or any mock-forest) as any mock-forest has tree-width at most 2. Also, it is easy to see that any pseudo-forest deletion set is also a mock-d-forest deletion set. But not all mock-d-forest deletion sets are pseudo-forest deletion sets. So, our parameter for this problem is not just smaller than solution size, it is even smaller than the parameter for FVS-Pseudo-Forest problem. But, it is larger than the size of the mock-forest deletion set for which case there is no polynomial kernel.

FVS-Mock-d-Forest where $d \geq 2$ and d is a constant **Parameter:** k
Input: An undirected graph G, $S \subseteq V(G)$ of size at most k such that $G[V(G) \setminus S]$ is a mock-forest where every component has at most d cycles and an integer ℓ.
Question: Does G have a feedback vertex set of size at most ℓ?

When d is not bounded, then we know that this problem has no polynomial kernel unless NP $\subseteq$ coNP/poly [16]. Here, we provide a $\mathcal{O}(k^{3d+3})$ vertex kernel for this problem when d is a constant. And we also prove that a kernel consisting of $O(k^{d+2-\epsilon})$ is unlikely for any $\epsilon > 0$ unless NP $\subseteq$ coNP/poly. We assume that for FVS-Pseudo-Forest, the deletion set to pseudo-forest is given with the input. But, this is not a serious assumption as there are constant factor approximation algorithms [10, 12] for computing a minimum vertex deletion set to a pseudo-forest of a graph.

We organise our paper as follows. In Section 2, we introduce the notations. In Section 3, we provide the FPT Algorithm for FVS-High-Degree and prove that it has no polynomial kernel unless NP $\subseteq$ coNP/poly. In Section 4, we provide the improved polynomial kernel for FVS-Pseudo-Forest. In Section 5, we provide the polynomial kernel and a conditional kernel lower bound of $\Omega(k^{d+2})$ for FVS-Mock-d-Forest.

2 Preliminaries and Notations

By $[r]$, we mean the set $\{1, 2, \ldots r\}$. Throughout the paper we denote the *feedback vertex set number* (the size of a minimum feedback vertex set) by $fvs(G)$. Let S be a set of vertices. By $\binom{S}{r}$, we denote the family of subsets of S containing *exactly* r vertices. By $\binom{S}{\leq r}$, we denote the family of subsets of S containing *at most* r vertices. We call a pair of vertices (u, v) a *double edge* if there are at least 2 edges between u and v. Otherwise we call (u, v) a *non-double-pair*. For an edge (u, v) the *multiplicity* of (u, v) is the number of edges present between u and v. Let $G = (V, E)$ be a tree or a pseudo-forest or a mock-forest. Then a set of vertices $V' \subseteq V(G)$ is a *degree-2-path* if V' induces an acyclic path and every vertex of the path has degree exactly 2 in G. A degree-2-path is maximal if no proper superset of V' is a degree-2-path. Let G be a graph where we contract an edge (u, v). Then we denote $G' = G/(u, v)$ as the graph created by contraction of edge (u, v). Let uv be the contracted vertex as a result of contraction. Then, $N_{G'}(uv) = N_G(u) \cup N_G(v)$. We denote $G[B]$ by the graph induced on the vertex set $B \subseteq V(G)$. We say $G[B]$ is a *double-clique* if there are at least 2 edges between every pair of vertices in B.

We give the definitions of fixed-parameter tractability, kernelization, polynomial parameter transformation and its related facts.

2.1 Definitions and Properties

▶ **Definition 1** (Fixed-Parameter Tractability)**.** Let $L \subseteq \sum^* \times \mathbb{N}$ is a parameterized language. L is said to be fixed-parameter tractable (or FPT) if there exists an algorithm $\mathcal{B}$, a constant c and a computable function f such that $\forall x, \forall k$, $\mathcal{B}$ on input (x, k) runs in at most $f(k).|x|^c$ time and outputs $(x, k) \in L$ iff $\mathcal{B}([x, k]) = 1$. We call the algorithm $\mathcal{B}$ as fixed-parameter algorithm.

▶ **Definition 2** (Slice-Wise Polynomial (XP))**.** Let $L \subseteq \sum^* \times \mathbb{N}$ is a parameterized language. L is said to be Slice-Wise Polynomial (or in XP) if there exists an algorithm $\mathcal{B}$, a constant c and computable functions f, g such that $\forall x, \forall k$, $\mathcal{B}$ on input (x, k) runs in at most $f(k).|x|^{g(k)+c}$ time and outputs $(x, k) \in L$ iff $\mathcal{B}([x, k]) = 1$. We call the algorithm $\mathcal{B}$ as XP Algorithm.

▶ **Definition 3** (Kernelization)**.** Let $L \subseteq \sum^* \times \mathbb{N}$ be a parameterized language. Kernelization is a procedure that replaces the input instance (I, k) by a reduced instance (I', k') such that

- $k' \leq f(k)$, $|I'| \leq g(k)$ for some function f, g depending only on k.
- $(I, k) \in L$ if and only if $(I', k') \in L$.

The reduction from (I, k) to (I', k') must be computable in $poly(|I| + k)$ time. If $g(k) = k^{\mathcal{O}(1)}$ then we say that L admits a *polynomial kernel*.

▶ **Definition 4** (Soundness/Safeness of Reduction Rule)**.** A reduction rule that replaces an instance (I, k) of a parameterized language L by a reduced instance (I', k') is said to be sound or safe if $(I, k) \in L$ if and only if $(I', k') \in L$.

▶ **Definition 5** (Polynomial parameter transformation (PPT))**.** Let P_1 and P_2 be two parameterized languages. We say that P_1 is polynomial parameter reducible to P_2 if there exists a polynomial time computable function (or algorithm) $f : \sum^* \times \mathbb{N} \to \sum^* \times \mathbb{N}$, a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that $(x, k) \in P_1$ if and only if $f(x, k) \in P_2$ and $k' \leq p(k)$ where $f((x, k)) = (x', k')$. We call f to be a polynomial parameter transformation from P_1 to P_2.

The following proposition gives the use of the polynomial parameter transformation for obtaining kernels for one problem from another.

▶ **Proposition 6** ([3])**.** *Let $P, Q \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems and assume that there exists a PPT from P to Q. Furthermore, assume that classical version of P is NP-hard and Q is in NP. Then if Q has a polynomial kernel then P has a polynomial kernel.*

2.2 Initial Preprocessing Rules

For the algorithms in Section 3, 4, 5, we use the following well known reduction rules. See Chapter 3, 4 of [6] for safeness of these Reduction Rules. Here ℓ is the size of the solution (fvs) being sought.

▶ **Reduction Rule 7.** *If there exists $u \in V(G)$ such that u has a self loop, then $G' \leftarrow G \setminus \{u\}, \ell' \leftarrow \ell - 1$.*

▶ **Reduction Rule 8.** *If there exists a vertex v such that $deg_G(v) \leq 1$, then $G' \leftarrow G \setminus v, \ell' \leftarrow \ell$.*

▶ **Reduction Rule 9.** *If there exists a vertex v such that $N_G(v) = \{u, w\}$, then delete the vertex v and add an edge (u, w) into G.*

Note that Reduction Rule 9 can create parallel edges.

▶ **Reduction Rule 10.** *If there exists an edge (u, v) whose multiplicity is more than 2, then reduce its multiplicity to 2.*

3 Feedback Vertex Set Parameterized by number of vertices of degree more than 3

FVS-High-Degree	**Parameter:** $\lvert\{u \in V(G) \lvert deg_G(u) > 3\}\rvert \leq k$
Input: An undirected graph $G = (V, E)$ and an integer ℓ.	
Question: Does G have a feedback vertex set of size at most ℓ?	

Note that our input can be a multigraph. We prove that this problem is fixed-parameter tractable and has no polynomial kernel unless $NP \subseteq coNP/poly$. First, we provide an FPT Algorithm to answer a question asked in [16]. Then, we prove that this problem has no polynomial kernel. Let $S = \{u \in V(G) | deg_G(u) > 3\}$. Throughout this section and in Sections 4, and 5, we use F to denote $G \setminus S$.

3.1 Fixed-Parameter Algorithm

Now, we provide the FPT Algorithm for this problem. We first make the graph minimum degree three. Then, we run over all possible subsets of S and for every subset S' of S, we reduce that instance to a polynomial time solvable problem.

▶ **Theorem 11.** *There exists an algorithm that runs in $\mathcal{O}(2^k \cdot n^{\mathcal{O}(1)})$ time for* FVS-High-Degree *problem.*

Proof. First we apply Reduction Rules 7, 8, 9, 10 in sequence and keep updating ℓ appropriately. Then the algorithm works as follows (see Algorithm 1 for pseudo-code).

We guess a subset $S' \subseteq S$ that intersects S with an ℓ sized feedback vertex set we are seeking for. If $G[S \setminus S']$ has a cycle, then we move on to the next guess. Otherwise, let $S'' = S \setminus S'$ and $G[S'']$ is a forest. Now, let $F = G \setminus S$. $\ell' = \ell - |S'|$. Now, we have to find a *minimum feedback vertex set* D of $G \setminus S'$ such that $S'' \cap D = \emptyset$. Note that every vertex in F has degree at most three in G and also in $G \setminus S'$. Now, we subdivide every edge $(u, v) \in E(F)$, by adding a new vertex $e_{u,v}$ and we add $e_{u,v}$ to S''. We get $T' = S'' \cup \{e_{u,v} | (u, v) \in E(F)\}$. u and v are the only two neighbours of $e_{u,v}$. Hence we have that for every vertex $u \in V(F)$, u has no neighbour in F. Let the graph we have currently is G''. Let $R' = V(G'') \setminus T'$. Note that R' is an independent set. Our goal is to find a feedback vertex set of G'' of at most ℓ' vertices that is disjoint from T'. Now, we pre-process G'' using the following reduction rules (also available in [18]) so that every vertex in R' has exactly three neighbours in T' and all such neighbours appear in different components of $G''[T']$. We need to apply these rules in sequence. Safeness of first two of them are easy to see.

▶ **Reduction Rule 12.** *If there exists $u \in R'$ such that $deg_{G''}(u) \leq 1$, then delete u.*

▶ **Reduction Rule 13.** *If there exists $u \in R'$ such that $G''[T' \cup \{u\}]$ has a cycle, then delete u from G'' and reduce ℓ' by 1.*

Notice that Reduction Rule 13 is also applicable when a vertex $u \in R'$ has exactly two neighbours in T' that are in same component of $G[T']$.

▶ **Reduction Rule 14.** *If there exists a vertex $u \in R'$ such that u has exactly 2 neighbours in T' and those two neighbours in different components of $G''[T']$, then move u to T'.*

▶ **Lemma 15** ($\star$[2]). *Reduction Rules 12, 13 and 14 are safe and can be implemented in polynomial time.*

When Reduction Rules 12, 13, 14 are not applicable, then our goal is to solve the following problem.

Special Disjoint Feedback Vertex Set
Input: An undirected graph $G = (V, E), S_1 \cup S_2 = V(G), S_1 \cap S_2 = \emptyset$, $G[S_1]$ is a forest, S_2 is an independent set and every vertex of S_2 has exactly 3 neighbours and all are in different components of $G[S_1]$.
Goal: Find a minimum feedback vertex set W of G such that $W \cap S_1 = \emptyset$.

The following Lemma is due to Kociumaka and Pilipczuk [18] which uses matroid techniques.

▶ **Lemma 16** ([18]). *Let (G, S_1, S_2) be an instance of* Special Disjoint Feedback Vertex Set. *Then there exists a polynomial time algorithm that finds a minimum feedback vertex set W of G such that $W \subseteq S_2$.*

Now, if $|W| \leq \ell'$, then we output Yes. Otherwise we repeat the above steps for another subset of S. There are at most $2^{|S|}$ many such subset S' of S and after guessing subset, the problem is polynomial time solvable. If for every subset of S, it is seen that $|W| > \ell'$, then we output No. Therefore, we have an algorithm that runs in time $\mathcal{O}^*(2^k)$. ◀

3.2 Kernelization Lower Bound

Now, to justify that FVS-High-Degree has no polynomial kernel unless NP $\subseteq$ coNP/poly, we use the following theorem and the construction that is used by Jansen et al. [16].

▶ **Theorem 17** ([11]). *Let ϕ be a boolean formula in CNF form with n variables and m clauses. CNF-SAT parameterized by n has no polynomial kernel unless* NP $\subseteq$ coNP/poly.

Jansen et al. [16] provided a polynomial parameter transformation from CNF-SAT parameterized by number of variables, n to Feedback Vertex Set parameterized by deletion distance to Mock-Forest. In that construction, the size of the deletion distance to Mock-Forest is at most $4n$. In the same construction, the number of vertices of the graph whose degree is at least 4 is $2n$. For details, see Section 6 in [16]. So, the same transformation is also a polynomial parameter transformation from CNF-SAT parameterized by number of variables to FVS-High-Degree. Thus, we have the following corollary.

▶ **Corollary 18.** FVS-High-Degree *has no polynomial kernel unless* NP $\subseteq$ coNP/poly.

4 Improved Polynomial Kernel for Parameterization by Deletion Distance to Pseudo-Forest

FVS-Pseudo-Forest	**Parameter:** k
Input: An undirected graph G, $S \subseteq V(G)$ of size at most k such that $G[V(G) \setminus S]$ is a graph in which every component has at most one cycle, and an integer ℓ.	
Question: Does G have a feedback vertex set of size at most ℓ?	

[2] Due to lack of space, the proofs of Lemmas and Observations marked $\star$ will appear in the full version.

Algorithm 1: FVS-Param-High-Degree-Vertices

input : $G = (V, E)$ and $\ell \in \mathbb{N}$
output : Yes if $\exists C \subseteq V(G), |C| \leq \ell$ such that $G \setminus C$ is a forest, No otherwise

```
S ← {u ∈ V(G) | deg_G(u) ≥ 4};
ℓ' ← ℓ;
for every S' ⊆ S do
    if G[S \ S'] is a forest then
        S'' ← S \ S';
        ℓ' ← ℓ − |S'|;
        F = G \ S;
        T = ∅;
        for each (u, v) ∈ E(F) do
            T ← T ∪ {e_{u,v}};
        T' ← T ∪ S'';
        E' = E(G[S'']) ∪ {(u, e_{u,v}) | (u, v) ∈ E(F)};
        G'' = (T', E');
        Apply Reduction Rules 12, 13, 14 in this sequence and keep updating ℓ'
         appropriately.;
        When Reduction Rules 12, 13, 14 are not applicable, run algorithm for
         Lemma 16 and get W;
        if |W| ≤ ℓ' then
            Return Yes
Return No;
```

Throughout the section for input (G, S, ℓ), we use F to denote $G[V(G) \setminus S]$. An $\mathcal{O}(k^{10})$ vertex kernel is provided by Jansen et al. [16]. We provide here an improved kernel. We first apply the Reduction Rules 7, 8, 9, 10. It is easy to see that these reduction rules can be applied in polynomial time. When Reduction Rules 7, 8, 9, 10 are not applicable, then every vertex of the graph has degree at least three and there are at most two edges between every pair of vertices. In particular, every vertex in $V(F)$ has at least one neighbour in S. We partition the vertices of F into H_1, H_2, H_3. We also partition the components of F into F_1, F_2, F_3, F_4. Formal notations are given as follows.

- $H_1 = \{u \in V(F) | deg_F(u) \leq 1\}$.
- $H_2 = \{u \in V(F) | deg_F(u) = 2\}$.
- $H_3 = \{u \in V(F) | deg_F(u) \geq 3\}$.
- F_1 – set of connected component of F that is a tree.
- F_2 – set of connected component of F that contains a vertex from H_1 and also contains a cycle. Let c_2 be the number of such components.
- F_3 – set of connected component of F that are induced cycles consisting of two vertices. Let c_3 be the number of such components.
- F_4 – set of connected component of F that are induced cycles of length at least three. Let c_4 be the number of such components.

Let $\mathcal{P}$ be the collection of *maximal degree-2-paths* in $F_1 \cup F_2$. Let $\hat{M}$ be a maximum matching in $G[\mathcal{P} \cup F_4]$. Also let $\hat{c} = c_2 + c_4$. We will use these notations in the rest of the section.

4.1 General Reduction Rules

Our first step is to device some reduction rules to bound the number of vertices in H_1. By pseudo-forest property, the number of vertices in H_3 becomes bounded. Now, to bound the number of vertices in H_2, we need to bound the number of edges in M, the number of maximal degree-2-paths in $\mathcal{P}$ and c_4. By pseudo-forest property, the number of maximal degree-2-paths in $\mathcal{P}$ also becomes bounded once $|H_1|$ and $|H_3|$ are bounded. In order to define such reduction rules, we need to use the fact crucially that the minimum degree of G is at least 3. In particular, for every vertex $v \in H_1$, either there exists $x \in S$ such that (x, v) is double-edge or there exists $x, y \in N_G(v) \cap S$. For every vertex $v \in H_1$, there exists $x \in S$ such that $(x, v) \in E(G)$. The reduction rules described in this subsection help to bound H_1 and also $M \cap (E(F_1) \cup E(F_2))$. These reduction Rules also appear in [4] in different form.

▶ **Reduction Rule 19.** *Let $x \in S$. Then $G' \leftarrow G \setminus \{x\}, \ell' \leftarrow \ell - 1$ if any of the following happens.*

- *There are at least $|S| + 1$ vertices in H_1 that are connected to x by a double-edge.*
- *There are at least $|S| + \hat{c} + 1$ vertices in $F_1 \cup F_2 \cup F_4$ that are matched by $\hat{M}$ and are connected to x by a double-edge.*
- *$N_G(x)$ contains both end points of at least $|S| + \hat{c} + 1$ edges in $\hat{M}$.*

▶ **Reduction Rule 20.** *Let $x, y \in S$ such that (x, y) is not a double-edge. Then, make (x, y) into a double-edge if one of the following happens.*

- *$|N_G(x) \cap N_G(y) \cap H_1| \geq |S| + 2$.*
- *$N_G(x) \cup N_G(y)$ contains both end points of at least $|S| + \hat{c} + 2$ edges of $\hat{M}$.*

Even though Reduction Rule 20 does not reduce the size of the graph, it helps to capture some constraints and also helps to apply some other reduction rules (for example Reduction Rule 21).

▶ **Reduction Rule 21.**

- *If there exists a vertex $u \in F$ such that $deg_F(u) = 0$, and there is no double edge attached to u and if $N_G(u) \cap S$ forms a double clique, then $G' \leftarrow G \setminus \{u\}$.*
- *If there exists $u \in F$ such that $deg_F(u) = 1$, and there is no double edge attached to u and $N_G(u) \cap S$ forms a double clique, then $G' \leftarrow G/(u, v)$ where $\{v\} = N_G(u) \cap F$. However, the multiple edges created because of contraction should not be deleted.*
- *If there exists $(u, v) \in M$ such that $N_G(u) \cap N_G(v) \cap S = \emptyset$, and no double edge is attached to either of u or v and $G[(N_G(u) \cup N_G(v)) \cap S]$ forms a double clique, then $G' \leftarrow G/(u, v)$. However, multiple edges or self loops created because of contraction should not be deleted.*

See Figure 1 for illustration.

4.2 Bounding $|H_1 \cup H_3|$

Now, we proceed to bound the number of vertices in F that have degree at most one and at least three. We know that F is a pseudo-forest. We need to use some structural properties of a pseudo-forest and also in applicability of the Reduction Rules 7, 8, 9, 10 19, 20 21. Here is an observation about pseudo-forest.

▶ **Observation 22 ($\star$).** *Let $G = (V, E)$ be a pseudo-forest and let $V_1 = \{v \in V(G) | deg_G(v) \leq 1\}$ and $V_3 = \{v \in V(G) | deg_G(v) \geq 3\}$. Then, $|V_3| \leq |V_1|$.*

Using the Observation 22, we have the following lemma.

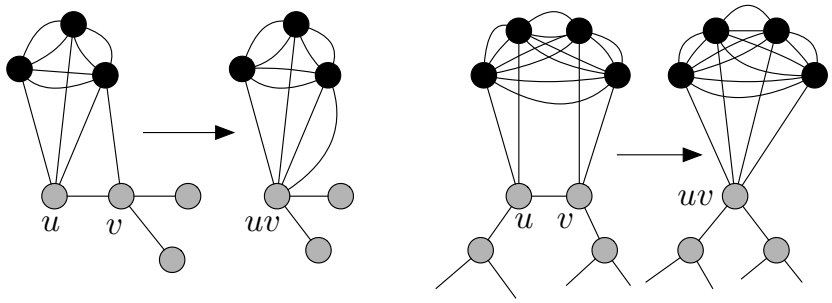


Figure 1 Illustration of Reduction Rule 21.

▶ **Lemma 23.** *When Reduction Rules 7, 8, 9, 10, 19, 20 and 21 are not applicable,* $|H_1 \cup H_3| = 2k^2 + 2(k+1)\binom{k}{2}$.

Proof. We know that $H_1 \cup H_3 \subseteq V(F_1 \cup F_2)$. Since Reduction Rules 7, 8, 9 are not applicable, every vertex in H_1 has at least two neighbours in S. As Reduction Rule 21 is not applicable, for every vertex $v \in H_1$, we associate either $z \in S$ when (x, z) is a double-edge. Otherwise we associate $(x, y) \in \binom{S}{2}$ for v, when $x, y \in N_G(v) \cap S$ and (x, y) is not a double-edge. As Reduction Rule 19 is not applicable, for every $z \in S$, there are at most $|S|$ vertices in H_1 that are connected by a double-edge. As Reduction Rule 20 is not applicable, for every $(x, y) \in \binom{S}{2}$ and (x, y) is not a double-edge, $N_G(x) \cap N_G(y)$ contain at most $|S| + 1$ vertices of H_1. Then $|H_1| \leq k^2 + (k+1)\binom{k}{2}$. By Observation 22, we know that $|H_3| \leq |H_1| \leq k^2 + (k+1)\binom{k}{2}$. So, $|H_1 \cup H_3| \leq 2k^2 + 2(k+1)\binom{k}{2}$. ◀

4.3 Bounding the number of components in F_3 and F_4

Now, what remains is to get an upper bound on $|H_2|$. Towards that, we need to bound $\hat{M}$ which requires to use an upper bound on the number of induced cycles, i.e. the number of components in F_4, i.e. c_4. We also need to bound c_3 to bound the number of vertices in H_2. In addition, we also need to use some facts from the earlier Subsection 4.1. By definition, for any component of F_3 and F_4, no vertex has exactly one neighbour in F. In particular the graph induced on the set of components of F_3 and F_4 is a two regular graph. To get an upper bound on number of such components, we need to do a little more work. We recall the following concept due to Jansen et al. [16].

▶ **Definition 24.** Let C be a connected component in $F_3 \cup F_4$ and let $X \subseteq N_G(C) \cap S$. We say that C *can be resolved with respect to* X if there exists $u \in C$ such that $C \setminus \{u\}$ is acyclic and for every connected component C' in $C \setminus \{u\}$, $|N_G(C') \cap X| \leq 1, |N_G(X) \cap C'| \leq 1$ and $G[(C \setminus \{u\}) \cup X]$ has no cycle.

The idea is that if a component C can be resolved with respect to its neighbourhood in S, then we can just delete that component and reduce the budget by 1. Every connected component of F_3 and F_4 are just induced cycles. When Reduction Rules 8, 9, 10 are not applicable, we show that the components in F_3 and F_4 have the following properties. A variation of the following lemma is provided in [16]. Here we provide an improved version of their lemma by using more facts that are useful for our purpose.

▶ **Lemma 25** ($\star$)**.** *Let C be a connected component in $F_3 \cup F_4$ and Reduction Rules 8, 9 be not applicable. Then, if there exists $X \subseteq N_G(C) \cap S$ such that C can not be resolved with respect to X then the followings statements are true.*

- *If $C \in F_3$, then there exists $X' \subseteq X, |X'| \leq 4$ such that C can not be resolved with respect to X'.*
- *If $C \in F_4$, then there exists $X' \subseteq X, |X'| \leq 3$ such that C can not be resolved with respect to X'.*

Now, the idea behind the proof of Lemma 25 is that if for some $X \subseteq S, |X| \leq 3$ (or $|X| \leq 4$), there are a large number of components that can not be resolved with respect to X, then any minimum feedback vertex set must intersect X. Therefore, we have the following definition (also available in [16]).

▶ **Definition 26.** Let (G, S, ℓ) be an instance of FVS-Pseudo-Forest. We say that $X \subseteq S, |X| \leq 4$ (respectively $|X| \leq 3$), be such that at least t connected components in F_3 (respectively F_4) can not be resolved with respect to X, then we say that X *is saturated by t unresolvable components* in F_3 (respectively F_4).

▶ **Lemma 27 (⋆).**
- *Let (G, S, ℓ) be an instance of* FVS-Pseudo-Forest *and $A \subseteq S, |A| \leq 3$ and A is saturated by $|S| + 4$ components in F_4, then any minimum feedback vertex set of G must intersect A.*
- *Let (G, S, ℓ) be an instance of* FVS-Pseudo-Forest *and $A \subseteq S, |A| \leq 4$ and A is saturated by $|S| + 7$ components in F_3, then any minimum feedback vertex set of G must intersect A.*

Now, we have the following Reduction Rule that follows from Lemma 27.

▶ **Reduction Rule 28.**
- *Let C be a connected component of F_3. If for each $A \subseteq \binom{S \cap N_G(C)}{\leq 4}$, component C can be resolved with respect to A or A is saturated by at least $|S| + 8$ non-resolvable components in F_3, then delete C and reduce ℓ by 1*
- *Let C be a connected component of F_4. If for each $A \subseteq \binom{S \cap N_G(C)}{\leq 3}$, component C can be resolved with respect to A or A is saturated by at least $|S| + 5$ non-resolvable components in F_4, then delete C and reduce ℓ by 1.*

▶ **Lemma 29 (⋆).** *Reduction Rules 19, 20, 21 and 28 are safe and can be implemented in polynomial time.*

Now, we have the following lemma when the above reduction rules are not applicable.

▶ **Lemma 30.** *Recall that the number of components in F_3, F_4 are c_3, c_4 respectively. When Reduction Rule 7, 8, 9, 10 28 are not applicable, then*
- $c_3 \leq (k+7) \sum_{i=1}^{4} \binom{k}{i}$.
- $c_4 \leq (k+4) \sum_{i=1}^{3} \binom{k}{i}$.

Proof. Assume that the conditions hold. We prove the statement in the given order.
- Since Reduction Rule 28 is not applicable, for each component C in F_3, there is a set $A \in \binom{N_G(C) \cap S}{\leq 4} \subseteq \binom{|S|}{4}$ such that C can be resolved with respect to A and A is saturated by at most $|S| + 7$ components. Then, for each component C in F_3, we choose one such set A and charge C to A. Clearly we can charge to every set A at most $|S| + 7$ times, otherwise some set A would be saturated by $|S| + 8$ components. Hence the number of components in F_3 is at most $(|S| + 7) \sum_{i=1}^{4} \binom{|S|}{i} \leq (k + 7) \sum_{i=1}^{4} \binom{k}{i}$.

- Similarly we can show that the number of components in F_4 is $(|S|+4)\sum_{i=1}^{3}\binom{|S|}{i} \leq (k+4)\sum_{i=1}^{3}\binom{k}{i}$. ◀

The following is an easy consequence of the Lemma 30.

▶ **Corollary 31.** *When Reduction Rule 7, 8, 9, 10 28 are not applicable, then, the number of vertices in F_3 is at most $2(k+7)\sum_{i=1}^{4}\binom{k}{i}$.*

4.4 Bounding $|H_2|$ and Putting Things together

Now, use the results in the earlier section and proceed to get an upper bound on the number of vertices in H_2. We need few more structural properties of pseudo-forests for that. As any component in F_2 has at least one vertex who has exactly one neighbour in F, number of components in F_2, i.e. $c_2 \leq |H_1|$. So, we have the following lemma.

▶ **Lemma 32.** *Recall that the number of components in F_2 is c_2. Then $c_2 \leq |H_1|$.*

Proof. Note that any component in F_2 must have at least one vertex from H_1. Therefore, the number of components in F_2 is at most $|H_1|$. ◀

Recall that in order to bound $|H_2|$, we also need an upper bound on the number of degree-2-paths in $\mathcal{P}$. The following is a structural property of pseudo-forest which helps to do so.

▶ **Observation 33 ($\star$).** *Let $G=(V,E)$ be a pseudo-forest where every component has at least one vertex of degree 1. Let $V_1=\{v\in V(G)|deg_G(v)\leq 1\}$ and $V_3=\{v\in V(G)|deg_G(v)\geq 3\}$ and $\mathcal{P}$ be the set of maximal degree-2-paths in G. Then, $|\mathcal{P}|\leq |V_3|+|V_1|$.*

▶ **Lemma 34.** *If Reduction Rule 7, 8, 9, 10, 19, 20, 21, 28 are not applicable, then the number of vertices in $|H_2|=\mathcal{O}(k^6)$.*

Proof. By Corollary 31, we have that $|V(F_3)|=\mathcal{O}(k^5)$. Recall that by Lemma 32, we have that $c_2=\mathcal{O}(k^3)$. Also by Lemma 30, we have that $c_4=\mathcal{O}(k^4)$. So, $\hat{c}=c_2+c_4=\mathcal{O}(k^4)$. Recall that $\hat{M}$ be the maximum matching in $\mathcal{P}\cup F_4$. As Reduction Rule 9 is not applicable, every vertex in H_2 has at least one neighbour in S. As Reduction Rule 21 is not applicable, for every $(u,v)\in\hat{M}$, we associate $z\in N_G(u)\cap N_G(v)\cap S$ when $N_G(u)\cap N_G(v)\cap S\neq\emptyset$ or (u,z) is a double-edge. Otherwise, we associate $x\in N_G(u)\cap S, y\in N_G(v)\cap S$ such that (x,y) is not a double-edge. For any $z\in S$, define $Matched(z)=\{(u,v)\in\hat{M}|(u,z)$ is a double-edge or $u,v\in N_G(z)\}$. As Reduction Rule 19 is not applicable, for every $z\in S$, $|Matched(z)|\leq 2|S|+2\hat{c}$. Similarly as Reduction Rule 20 is not applicable, for every $x,y\in\binom{S}{2}$ and (x,y) is not a double-edge, $N_G(x)\cup N_G(y)$ contain both end points of at most $|S|+\hat{c}+1$ edges of $\hat{M}$. Since $\hat{c}=\mathcal{O}(k^4)$, we have that $|\hat{M}|\leq|S|(2|S|+2\hat{c})+\binom{S}{2}(|S|+\hat{c}+1)\leq 2k^2+2k\cdot\mathcal{O}(k^4)+\binom{k}{2}\cdot\mathcal{O}(k^4)=\mathcal{O}(k^6)$. A *maximal degree-2-path* can also have only one vertex which is not matched by $\hat{M}$. Recall that $\mathcal{P}$ be the collection of all *maximal degree-2-paths* in $F_1\cup F_2$. Using Observation 33, we get that $|\mathcal{P}|\leq|H_1|+|H_3|=\mathcal{O}(k^3)$. So, the number of vertices in H_2 that are not matched by $\hat{M}$ is at most $|\mathcal{P}|+|\hat{M}|=\mathcal{O}(k^6)$. So, $|H_2|=\mathcal{O}(k^6)$. ◀

The following is the main theorem of this section and this is an easy consequence of Lemma 23 and Lemma 34.

▶ **Theorem 35.** FVS-Pseudo-Forest *has a kernel consisting of $\mathcal{O}(k^6)$ vertices.*

5 Kernelization of Feedback Vertex Set Parameterized by Deletion distance to bounded Mock Forest

Now we consider the Feedback Vertex Set problem parameterized by the size of a deletion set whose deletion results in a *mock-d-forest*. Recall that a graph is called mock-d-forest when every vertex is contained in at most one cycle and every connected component has at most d cycles. Formal definition of the problem is given below.

FVS-Mock-d-Forest for $d \geq 2$ and d is a constant **Parameter:** k
Input: An undirected graph G, $S \subseteq V(G)$ of size at most k such that $G[V(G) \setminus S]$ is a graph of which every vertex participates in at most most one cycle, every component has at most d cycles for some constant d and an integer ℓ.
Question: Does G have a feedback vertex set of size at most ℓ?

When d is not bounded, then there is no polynomial kernel unless NP $\subseteq$ coNP/poly. In this section, we first provide a polynomial kernel for this problem when d is a constant and $d \geq 2$. After that we provide a lower bound for this problem.

5.1 Polynomial Kernel for FVS-Mock-d-Forest

Our kernelization algorithm follows along the line of the kernel for FVS-Pseudo-Forest in the earlier section. Here, we need to use some special properties of mock-d-forest. We use $F = G \setminus S$ throughout the section. Let $\mathcal{P}_F$ be the collection of maximal acyclic degree-2-paths in F. Let M_F be a maximum matching in $\mathcal{P}_F$ and set of induced cycles in F. Let $\hat{c}$ be the total number of cycles in F. We partition $V(F)$ into three parts as follows.

- $F_1 = \{u \in V(F) | deg_F(u) \leq 1\}$.
- $F_2 = \{u \in V(F) | deg_F(u) = 2\}$.
- $F_3 = \{u \in V(F) | deg_F(u) \geq 3\}$.

Our first step is to bound the number of vertices in F_1. An upper bound on F_1 along with some properties of pseudo-forest, we get an upper bound on the number of vertices of F_3. Then, we have to bound the number of edges in M_F and the number of maximal acyclic degree-2-paths in $\mathcal{P}_F$. Now, we are ready to state the Reduction Rules. Our Reduction Rules in this section are generalisations of the Reduction Rules in Section 4. We apply the following two reduction rules that are more general variant of Reduction Rules 19, 20.

▶ **Reduction Rule 36.** *Let $x \in S$. Then $G' \leftarrow G \setminus \{x\}, \ell' \leftarrow \ell - 1$ if one of the following conditions is satisfied.*

- *There are at least $|S| + 1$ vertices in F_1 that are connected to x by a double-edge.*
- *There are at least $|S| + \hat{c} + 1$ vertices in F_2 that are matched by M_F and are connected to x by a double-edge.*
- *$N_G(x)$ contains both end points of at least $|S| + \hat{c} + 1$ edges in M_F.*

▶ **Reduction Rule 37.** *Let $(x, y) \in \binom{S}{2}$. Then make (x, y) into a double-edge if one of the following conditions is satisfied.*

- *$N_G(x) \cap N_G(y)$ contains at least $|S| + 1$ vertices from F_1.*
- *$N_G(x) \cap N_G(y)$ contains both end points of at least $|S| + \hat{c} + 2$ edges of M_F.*

We apply Reduction Rules 7, 8, 9, 10, 36, 37, 21 in the this order (Recall that we did similar in Section 4).

▶ **Lemma 38 ($\star$).** *When Reduction Rules 7 8, 9, 10, 36, 37, 21 are not applicable, then $|F_1| = \mathcal{O}(k^3)$.*

We need to bound the number of vertices in F_2 and F_3. To have that, we need to bound the number of components in F. We need to do little more work for that. In particular, we need the following definition which is a generalisation of Definition 24.

▶ **Definition 39.** Let C be a connected component in F and let $X \subseteq N_G(C) \cap S$. We say that C *can be resolved with respect to* X if there exists $\{u_1, u_2, \ldots, u_d\} \subseteq C$ such that $C \setminus \{u_1, \ldots, u_d\}$ is acyclic and for every connected component C' in $C \setminus \{u_1, \ldots, u_d\}$, $|N_G(C') \cap X| \leq 1, |N_G(X) \cap C'| \leq 1$ and $G[(C \setminus \{u_1, \ldots, u_d\}) \cup X]$ has no cycle.

The following lemma is a generalisation of Lemma 25. It is useful to bound the number of components in F.

▶ **Lemma 40** ($\star$). *Let C be a connected component of F having exactly d cycles and let $X \subseteq N_G(C) \cap S$ such that C can not be resolved with respect to X. Then, there exists $X' \subseteq X, |X'| \leq 3d$ such that C can not be resolved with respect to X'.*

The following definition is a generalisation of Definition 26 in Section 4.

▶ **Definition 41.** Let $A \subseteq S, |A| \leq 3d$ be such that there are t components in F that can not be resolved with respect to A. Then, we say that A *is saturated by t components* in F.

The following lemma is a property of mock-forest. We will need this to prove the safeness of the Reduction Rule 43.

▶ **Lemma 42** ($\star$). *Let (G, S, ℓ) be an instance of* FVS-Mock-d-Forest] *and $A \subseteq S, |A| \leq 3d$ and A is saturated by $|S| + \binom{3d}{2} + 1$ components in F, then any minimum feedback vertex set of G must intersect A.*

Now, we have just one more reduction rule to get an upper bound on the number of components in F. And Lemma 45 is a consequence of inapplicability of Reduction Rule 43.

▶ **Reduction Rule 43.** *Let C be a connected component in F that contains some cycle. If for each $A \in \binom{N_G(C) \cap X}{\leq 3d}$, component C can be resolved with respect to A or A is saturated by $|S| + \binom{3d}{2} + 2$ components, then remove C and reduce ℓ by the number of cycles in C.*

▶ **Lemma 44** ($\star$). *Reduction Rules 36, 37 and 43 are safe and can be implemented in polynomial time.*

▶ **Lemma 45.** *Let (G, S, ℓ) be an irreducible instance with respect to Reduction Rule 43, then number of components in F is at most $\mathcal{O}(|S|^{3d+1})$.*

Proof. Consider any component $C \in F$. Reduction Rule 43 is not applicable, therefore, there exists $A \subseteq S, |A| \leq 3d$ such that C can not be resolved with respect to A. Also, for the same reason, A can be saturated by at most $|S| + \binom{3d}{2} + 1$ components. Therefore, the number of components is at most $(|S| + \binom{3d}{2})\binom{|S|}{3d} \leq 9d^2.|S|^{3d+1} = \mathcal{O}(d^2.|S|^{3d+1})$. ◀

We have bounded the number of components in F. We already have bounded the number of vertices in F_1. We are left to bound $|F_3 \cup F_2|$. We need graph theoretic properties of mock-d-forest to get an upper bound on $|F_3|$. Recall that in Section 4, we used observations about pseudo-forest. Similarly, in this section, we use observations about mock-forest when there are at most d cycles in a mock-forest.

▶ **Observation 46** ($\star$). *Let $G = (V, E)$ be a mock forest with c components where every component has at most d cycles. $V_1 = \{v \in V(G) | deg_G(v) \leq 1\}, V_2 = \{v \in V(G) | deg_G(v) = 2\}, V_3 = \{v \in V(G) | deg_G(v) \geq 3\}$. Then $|V_3| \leq |V_1| + 2cd - 2c$.*

Using Lemma 38 and Observation 46, we have the following Lemma.

▶ **Lemma 47.** *Let c be the number of components in F. Then, $|F_3| = \mathcal{O}(k^{3d+1})$.*

Proof. By Lemma 38, we know that $|F_1| = \mathcal{O}(k^3)$. Now, by Observation 46, we know that $|F_3| \leq |F_1| + 2c(d-1)$. Recall that $c = \mathcal{O}(k^{3d+1})$. Now, $c(d-1) \leq \hat{c} = \mathcal{O}(k^{3d+1})$. So, $|F_3| = \mathcal{O}(k^{3d+1})$. ◀

Now, what remains is to bound the number of vertices in F_2. For that, we need to bound M_F and also the number of maximal acyclic degree-2-paths in $\mathcal{P}_F$. Using structural properties of mock-d-forest, we have the following lemma that bounds the number of maximal acyclic degree-2-paths in F, i.e. $\mathcal{P}_F$.

▶ **Lemma 48 (⋆).** *$|\mathcal{P}_F| = \mathcal{O}(k^{3d+1})$ where c' is the number of components in F that have at least two cycles.*

Using the above observations and lemmas we have the following lemma.

▶ **Lemma 49 (⋆).** *$|F_2| = \mathcal{O}(|S|^{3d+3})$.*

Combining Lemma 38, 47, 49, we get the following theorem.

▶ **Theorem 50.** FVS-Mock-d-Forest *has a kernel consisting of $\mathcal{O}(k^{3d+3})$ vertices.*

5.2 Kernel Lower Bound for FVS-Mock-d-Forest

We provide a polynomial parameter transformation from $(d+2)$-CNF-SAT parameterized by the number of variables to Feedback Vertex Set parameterized by deletion distance to Mock-d-Forest where $d \geq 2$. A polynomial parameter transformation from CNF-SAT to FVS-Mock-Forest when every clause has exactly r literals where r is a power of 2 is already known [16]. We modify the construction for a polynomial parameter transformation from $(d+2)$-CNF-SAT to FVS-Mock-d-Forest where d is not necessarily a power of 2.

Let the clause C_i have $d_i \leq d+2$ literals. We provide a clause gadget of height j_i where $2^{j_i-1} < d_i \leq 2^{j_i}$. We create d^2 many copies for this gadget. In this gadget, the terminal vertices are the corresponding vertices of literals (See figure 2). For clause C_q with its r'th copy, we name literals as $y_{q,r,1}, \ldots, y_{q,r,d_i}$. And we create a variable gadget for variable x_i as a cycle of 3 vertices. Let $\{t_i, f_i, e_i\}$ are those vertices. We define $S = \bigcup_{i=1}^{n} \{t_i, f_i, e_i\}$. Let $y_{q,r,j}$ be the j'th literal of clause C_q. Let the variable corresponding to that variable is x_i. Then, if the literal $y_{q,r,j}$ is $\bar{x_i}$, then we connect $y_{q,r,j}$ with f_i. Otherwise we connect $y_{q,r,j}$ with t_i. We do the same for every $r \in [d^2]$. We set $\ell = d^2 \sum_{i=1}^{m} (d_i - 2)$.

▶ **Lemma 51 (⋆).** *Let ϕ be a $(d+2)$-CNF formula. Let G_ϕ be the graph constructed from ϕ using the construction above. Then ϕ is satisfiable if and only if (G_ϕ, S, ℓ) is* Yes-Instance. *Thus there is a polynomial parameter transformation from $(d+2)$-CNF-SAT to* FVS-Mock-d-Forest.

▶ **Theorem 52.** *[[7]] d-*CNF-SAT *parameterized by n, the number of variables, has no kernel of size $\mathcal{O}(n^{d-\epsilon})$ for any $d \geq 3, \epsilon > 0$ unless* NP $\subseteq$ coNP/poly.

Using Lemma 51 and Theorem 52, we have the following theorem.

▶ **Theorem 53.** FVS-d-Mock-Forest *has no kernel consisting of $\mathcal{O}(k^{d+2-\epsilon})$ vertices for every $d \geq 2, \epsilon > 0$ unless* NP $\subseteq$ coNP/poly.

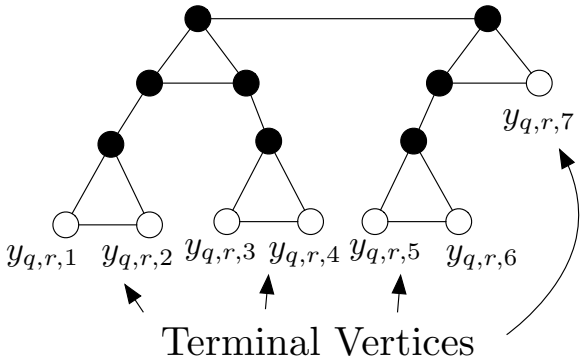


Figure 2 Illustration of Clause Gadget Construction for 7 literals.

6 Conclusion

We have given a kernel with $\mathcal{O}(k^6)$ vertices for FVS-Pseudo-Forest improving from an earlier $\mathcal{O}(k^{10})$ bound [16], and narrowing the gap with the $\Omega(k^4)$ conditional lower bound. Bridging the gap further is an interesting problem. We proved that FVS-High-Degree is fixed-parameter tractable. Status of Feedback Vertex Set parameterized by deletion distance to a (sub)-cubic graph (a related problem) remains open, and we do not even know an XP algorithm for the problem. We considered FVS parameterized by deletion distance to mock-d-forest and proved an upper bound of $\mathcal{O}(k^{3d+3})$ and lower bound $\Omega(k^{d+2})$ under complexity theoretic assumptions. Narrowing this gap is another interesting future direction.

References

1 E. Balas and C. S. Yu. On graphs with polynomially solvable maximum-weight clique problem. *Networks*, 19(2):247–253, 1989.

2 H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014.

3 H. L. Bodlaender, S. Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011.

4 H. L. Bodlaender and T. C. van Dijk. A cubic kernel for feedback vertex set and loop cutset. *Theory Comput. Syst.*, 46(3):566–597, 2010.

5 Y. Cao, J. Chen, and Y. Liu. On feedback vertex set: New measure and new structures. *Algorithmica*, 73(1):63–86, 2015.

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

7 H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014.

8 R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

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

10 F. V. Fomin, D. Lokshtanov, N. Misra, and S. Saurabh. Planar F-Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *Proceedings of FOCS*, pages 470–479, 2012.

11 L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

12 T. Fujito. A unified approximation algorithm for node-deletion problems. *Discrete Applied Mathematics*, 86(2-3):213–231, 1998.

13 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

14 G. Gutin, E. J. Kim, M. Lampis, and V. Mitsou. Vertex cover problem parameterized above and below tight bounds. *Theory Comput. Syst.*, 48(2):402–410, 2011.

15 B. M. P. Jansen and S. Kratsch. Data reduction for graph coloring problems. *Inf. Comput.*, 231:70–88, 2013.

16 B. M. P. Jansen, V. Raman, and M. Vatshelle. Parameter ecology for feedback vertex set. *Tsinghua Science and Technology*, 19(4):387–409, 2014.

17 T. Kloks, C. Liu, and S. Poon. Feedback vertex set on chordal bipartite graphs. *CoRR*, abs/1104.3915v2, 2011.

18 T. Kociumaka and M. Pilipczuk. Faster deterministic feedback vertex set. *Inf. Process. Lett.*, 114(10):556–560, 2014.

19 D. Kratsch, H. Müller, and I. Todinca. Feedback vertex set on at-free graphs. *Discrete Applied Mathematics*, 156(10):1936–1947, 2008.

20 R. Rizzi. Minimum weakly fundamental cycle bases are hard to find. *Algorithmica*, 53(3):402–424, 2009.

21 S. Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2), 2010.

22 S. Ueno, Y. Kajitani, and S. Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex degree exceeding three. *Discrete Mathematics*, 72(1-3):355–360, 1988.

Randomised Enumeration of Small Witnesses Using a Decision Oracle

Kitty Meeks

School of Mathematics and Statistics, University of Glasgow, Glasgow , UK
kitty.meeks@glasgow.ac.uk

Abstract

Many combinatorial problems involve determining whether a universe of n elements contains a witness consisting of k elements which have some specified property. In this paper we investigate the relationship between the decision and enumeration versions of such problems: efficient methods are known for transforming a decision algorithm into a search procedure that finds a single witness, but even finding a second witness is not so straightforward in general. In this paper we show that, if the decision version of the problem belongs to FPT, there is a randomised algorithm which enumerates all witnesses in time $f(k) \cdot poly(n) \cdot N$, where N is the total number of witnesses and f is a computable function. This also gives rise to an efficient algorithm to count the total number of witnesses when this number is small.

1998 ACM Subject Classification F.2.0 Analysis of Algorithms and Problem Complexity, G.2.1 Combinatorial Algorithms

Keywords and phrases enumeration algorithms, parameterized complexity, randomized algorithms, color coding

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.22

1 Introduction

Many well-known combinatorial decision problems involve determining whether a universe U of n elements contains a witness W consisting of *exactly* k elements which have some specified property. Specifically, we are concerned with problems for which any decision algorithm can be called with input universe $X \subset U$ in order to determine whether there is a witness W for the *original* problem (i.e. with universe U) such that $W \subseteq X$; we will call such problems *self-contained k-witness problems*. Thus the well-studied problems k-Clique, k-Cycle and k-Path are all self-contained k-witness problems, but others such as k-Vertex Cover and k-Dominating Set are not (as we need to preserve information about the relationship of any potential witness to the entire universe U).

While the basic decision versions of self-contained k-witness problems are of interest, it is often not sufficient for applications to output simply "yes" or "no": we need to *find* a witness. The issue of finding a single witness using an oracle for the decision problem has previously been investigated by Björklund, Kaski, and Kowalik [5], motivated by the fact that the fastest known parameterised algorithms for a number of widely studied problems (such as graph motif [4] and k-path [3]) are non-constructive in nature. Moreover, for some problems (such as k-Clique or Independent Set [2] and **p-**Even Subgraph [15]) the only known FPT decision algorithm relies on a Ramsey theoretic argument which says the answer must be "yes" provided that the input graph avoids certain easily recognisable structures.

Following the first approach used in [5], we assume the existence of a deterministic "oracle" (a black-box decision procedure), as follows.

ORA(X)
Input: $X \subseteq U$
Output: 1 if some witness is entirely contained in X; 0 otherwise.

A naïve approach easily finds a single witness using $\Theta(n)$ calls to **ORA**: we successively delete elements of the universe, following each deletion with an oracle call, and if the oracle answers "no" we reinsert the last deleted element and continue. Assuming we start with a yes-instance, this process will terminate when only k elements remain, and these k elements must form a witness. In [5], ideas from combinatorial group testing are used to make a substantial improvement on this strategy for the extraction of a single witness: rather than deleting a single element at a time, large subsets are discarded (if possible) at each stage. This gives an algorithm that extracts a witness with only $2k\left(\log_2\left(\frac{n}{k}\right)+2\right)$ oracle queries.

However, neither of these approaches for finding a single witness can immediately be extended to find *all* witnesses, a problem which is of interest even if an efficient decision algorithm does output a single witness; indeed, it is not even obvious how to find a second witness. Both approaches for finding a first witness rely on the fact that we can safely delete some subset of elements from our universe provided we know that what is left still contains at least one witness; if we need to look for a second witness, the knowledge that at least one witness will remain is no longer sufficient to guarantee we can delete a given subset. Of course, for any self-contained k-witness problem we can check all possible subsets of size k, and hence enumerate all witnesses, in time $O(n^k)$; indeed, if *every* set of k vertices is in fact a witness then we will require this amount of time simply to list them all. However, we can seek to do much better than this when the number of witnesses is small by making use of a decision oracle.

The enumeration problem becomes straightforward if we have an *extension oracle*,[1] defined as follows.

EXT-ORA(X, Y)
Input: $X \subseteq U$ and $Y \subseteq X$
Output: 1 if there exists a witness W with $Y \subseteq W \subseteq X$; 0 otherwise.

The existence of an efficient procedure **EXT-ORA**(X, Y) for a given self-contained k-witness problem allows us to use standard backtracking techniques to devise an efficient enumeration algorithm. We explore a binary search tree of depth $O(n)$, branching at level i of the tree on whether the i^{th} element of U belongs to the solution. Each node in the search tree then corresponds to a specific pair (X, Y) with $Y \subseteq X \subseteq U$; we can call **EXT-ORA**(X, Y) to determine whether any descendant of a given node corresponds to a witness. Pruning the search tree in this way ensures that no more than $O(n \cdot N)$ oracle calls are required, where N is the total number of witnesses.

Note that, with only the basic decision oracle, we can determine whether there is a witness that does *not* contain some element x (we simply call **ORA**$(U \setminus \{x\})$), but we cannot determine whether there is a witness which *does* contain x. However, as we will show in Section 3, there are natural self-contained k-witness problems for which there is no fpt-algorithm for the extension decision problem unless FPT=W[1]. This motivates the development of enumeration algorithms that do not rely on such an oracle.

The main result of this paper is just such an algorithm; specifically, we prove the following theorem.

[1] Such an oracle is sometimes called an *interval* oracle, as in the enumeration procedure described by Björklund, Kaski, Kowalik and Lauri [6] which builds on earlier work by Lawler [19].

▶ **Theorem 1.1.** *There is a randomised algorithm to enumerate all witnesses of size k in a self-contained k-witness problem exactly once, whose expected number of calls to a deterministic decision oracle is at most $2^{O(k)} \log^2 n \cdot N$, where N is the total number of witnesses. Moreover, if an oracle call can be executed in time $g(k) \cdot n^{O(1)}$ for some computable function g, then the expected total running time of the algorithm is*

$$2^{O(k)} \cdot g(k) \cdot n^{O(1)} \cdot N.$$

The key tool we use to obtain this algorithm is a colour coding method, using a family of k-perfect hash functions. This technique was introduced by Alon, Yuster and Zwick in [1] and has been widely used in the design of parameterised algorithms for decision and approximate counting (see for example [14, Chapters 13 and 14] and [11, Chapter 8]), but to the best of the author's knowledge has not yet been applied to enumeration problems.

Theorem 1.1 is proved in Section 4, before some implications of our enumeration algorithm for the complexity of related counting problems are discussed in Section 5. We begin in Section 2 with some background on relevant complexity theoretic notions, before discussing the hardness of the extension version of some self-contained k-witness problems in Section 3.

2 Parameterised enumeration

There are two natural measures of the size of a self-contained k-witness problem, namely the number of elements n in the universe and the number of elements k in each witness, so the running time of algorithms is most naturally discussed in the setting of parameterised complexity. There are two main complexity issues to consider in the present setting: first of all, as usual, the running time, and secondly the number of oracle calls required.

For general background on the theory of parameterised complexity, we refer the reader to [11, 14]. The theory of parameterised enumeration has been developed relatively recently [12, 8, 7], and we refer the reader to [8] for the formal definitions of the different classes of parameterised enumeration algorithms. To the best of the author's knowledge, this is the first occurrence of a randomised parameterised enumeration algorithm in the literature, and so we introduce randomised analogues of the four types of parameterised enumeration algorithms introduced in [8] (for a problem with total input size n and parameter k, and with $f : \mathbb{N} \to \mathbb{N}$ assumed to be a computable function throughout):

- an expected-total-fpt algorithm enumerates all solutions and terminates in expected time $f(k) \cdot n^{O(1)}$;
- an expected-delay-fpt algorithm enumerates all solutions with expected delay at most $f(k) \cdot n^{O(1)}$ between the times at which one solution and the next are output (and the same bound applies to the time before outputting the first solution, and between outputting the final solution and terminating);
- an expected-incremental-fpt algorithm enumerates all solutions with expected delay at most $f(k) \cdot (n+i)^{O(1)}$ between outputting the i^{th} and $(i+1)^{th}$ solution;
- an expected-output-fpt algorithm enumerates all solutions and terminates in expected time $f(k) \cdot (n+N)^{O(1)}$, where N is the total number of solutions enumerated.

Under these definitions, Theorem 1.1 says that, if the decision version of a self-contained k-witness problem belongs to FPT, then there is an expected-output-fpt algorithm for the corresponding enumeration problem.

3 Hardness of the extension problem

Many combinatorial problems have a very useful property, often referred to as *self-reducibility*, which allows a search or enumeration problem to be reduced to (smaller instances of) the corresponding decision problem in a very natural way (see [8, 18, 21]). A problem is self-reducible in this sense if the existence of an efficient decision procedure (equivalent to **ORA**(X)) implies that there is an efficient algorithm to solve the extension decision problem (equivalent to **EXT-ORA**(X)). While many self-contained k-witness problems do have this property, we will demonstrate that there exist self-contained k-witness problems that do not (unless FPT=W[1]), and so an enumeration procedure that makes use only of **ORA**(X) and not **EXT-ORA**(X) is desirable.

In order to demonstrate this, we show that there exist self-contained k-witness problems whose decision versions belong to FPT, but for which the corresponding extension decision problem is W[1]-hard. We will consider the following problem, which is clearly a self-contained k-witness problem.

p-Clique or Independent Set
Input: A graph $G = (V, E)$ and $k \in \mathbb{N}$.
Parameter: k.
Question: Is there a k-vertex subset of V that induces either a clique or an independent set?

This problem is known to belong to FPT [2]: all sufficiently large input graphs are yes-instances by Ramsey's Theorem. We now turn our attention to the extension version of the problem, defined as follows.

p-Extension Clique or Independent Set
Input: A graph $G = (V, E)$, a subset $U \subseteq V$ and $k \in \mathbb{N}$.
Parameter: k.
Question: Is there a k-vertex subset S of V, with $U \subseteq S$, that induces either a clique or an independent set?

It is straightforward to adapt the hardness proof for **p-**Multicolour Clique or Independent Set [20, Proposition 3.7] to show that **p-**Extension Clique or Independent Set is W[1]-hard.

▶ **Proposition 3.1.** **p-**Extension Clique or Independent Set *is* W[1]*-hard.*

Proof. We prove this result by means of a reduction from the W[1]-complete problem **p-**Clique. Let (G, k) be the input to an instance of **p-**Clique. We now define a new graph G', obtained from G by adding one new vertex v, and an edge from v to every vertex $u \in V(G)$. It is then straightforward to verify that $(G', \{v\}, k + 1)$ is a yes-instance for **p-**Extension Clique or Independent Set if and only if G contains a clique of size k. ◀

This demonstrates that **p-**Extension Clique or Independent Set is a problem for which there exists an efficient decision procedure but no efficient algorithm for the extension version of the decision problem (unless FPT=W[1]). The reduction given here can easily be adapted to demonstrate that the following problem has the same property.

p-Induced Regular Subgraph
Input: A graph $G = (V, E)$ and $k \in \mathbb{N}$.
Parameter: k.
Question: Is there a k-vertex subset of V that induces a subgraph in which every vertex has the same degree?

Indeed, the same method can be applied to any problem in which putting a restriction on the degree of one of the vertices in the witness guarantees that the witness induces a clique (or some other induced subgraph for which it is W[1]-hard to decide inclusion in an arbitrary input graph).

4 The randomised enumeration algorithm

In this section we describe and analyse our randomised witness enumeration algorithm, thus proving Theorem 1.1.

As mentioned above, our algorithm relies on a colour coding technique. A family $\mathcal{F}$ of hash functions from $[n]$ to $[k]$ is said to be *k-perfect* if, for every subset $A \subset [n]$ of size k, there exists $f \in \mathcal{F}$ such that the restriction of f to A is injective. We will use the following bound on the size of such a family of hash functions, proved in [1].

▶ **Theorem 4.1.** *For all $n, k \in \mathbb{N}$ there is a k-perfect family $\mathcal{F}_{n,k}$ of hash functions from $[n]$ to $[k]$ of cardinality $2^{O(k)} \cdot \log n$. Furthermore, given n and k, a representation of the family $\mathcal{F}_{n,k}$ can be computed in time $2^{O(k)} \cdot n \log n$.*

Our strategy is to solve a collection of $2^{O(k)} \cdot \log n$ *colourful enumeration problems*, one corresponding to each element of a family $\mathcal{F}$ of k-perfect hash functions. In each of these problems, our goal is to enumerate all witnesses that are *colourful* with respect to the relevant element f of $\mathcal{F}$ (those in which each element is assigned a distinct colour by f). Of course, we may discover the same witness more than once if it is colourful with respect to two distinct elements in $\mathcal{F}$, but it is straightforward to check for repeats of this kind and omit duplicate witnesses from the output. It is essential in the algorithm that we use a deterministic construction of a k-perfect family of hash functions rather than the randomised construction also described in [1], as the latter method would allow the possibility of witnesses being omitted (with some small probability).

The advantage of solving a number of colourful enumeration problems is that we can split the problem into a number of sub-problems with the only requirement being that we preserve witnesses in which every element has a different colour (rather than all witnesses). This makes it possible to construct a number of instances, each (roughly) half the size of the original instance, such that every colourful witness survives in at least one of the smaller instances. More specifically, for each k-perfect hash function we explore a search tree: at each node, we split every colour-class randomly into (almost) equal-sized parts, and then branch to consider each of the 2^k combinations that includes one (nonempty) subset of each colour, provided that the union of these subsets still contains at least one witness (as determined by the decision oracle). This simple pruning of the search tree will not prevent us exploring "dead-ends" (where we pursue a particular branch due to the presence of a non-colourful witness), but turns out to be sufficient to make it unlikely that we explore very many branches that do not lead to colourful witnesses.

We describe the algorithm in pseudocode (Algorithm 1), making use of two subroutines. In addition to our oracle **ORA**(X), we also define a procedure **RANDPART**(X) which we use, while exploring the search tree, to obtain a random partition of a subset of the universe.

RANDPART(X)
Input: $X \subseteq U$
Output: A partition (X_1, X_2) of X with $||X_1| - |X_2|| \leq 1$, chosen uniformly at random from all such partitions of X.

Algorithm 1: Randomised algorithm to enumerate all k-element witnesses in the universe U, using a decision oracle.

```
1  Construct a family F = {f_1, f_2, ..., f_|F|} of k-perfect hash functions from U to [k];
2  for 1 ≤ r ≤ |F| do
3      Initialise an empty FIFO queue Q;
4      if ORA(U) = 1 then
5          Insert U into Q;
6      end if
7      while Q is not empty do
8          Remove the first element A from Q;
9          if |A| = k then
10             if A is not colourful with respect to f_s for any s ∈ {1, ..., r − 1} then
11                 Output A;
12             end if
13         else
14             for 1 ≤ i ≤ k do
15                 Set A_i to be the set of elements in A coloured i by f_r;
16                 Set (A_i^(1), A_i^(2)) = RANDPART(A_i);
17             end for
18             for each j = (j_1, ..., j_k) ∈ {1, 2}^k do
19                 if |A_i^(j_ℓ)| > 0 for each 1 ≤ ℓ ≤ k then
20                     Set A_j = A_i^(j_1) ∪ ··· ∪ A_i^(j_k);
21                     if ORA(A_j) = 1 then
22                         Add A_j to Q;
23                     end if
24                 end if
25             end for
26         end if
27     end while
28 end for
```

We prove the correctness of the algorithm in Section 4.1, and bound the expected running time in Section 4.2.

4.1 Correctness of the algorithm

In order to prove that our algorithm does indeed output every witness exactly once, we begin by showing that we will identify a given k-element subset X during the iteration corresponding to the hash-function $f \in \mathcal{F}$ if and only if X is a colourful witness with respect to f.

▶ **Lemma 4.2.** *Let X be a set of k vertices in the universe U. In the iteration of Algorithm 1 corresponding to $f \in \mathcal{F}$, we will execute 10 to 12 with $A = X$ if and only if:*
1. *X is a witness, and*
2. *X is colourful with respect to f.*

Proof. We first argue that we only execute lines 10 to 12 with $A = X$ if X is a witness

and is colourful with respect to f. We claim that, throughout the execution of the iteration corresponding to f, every subset B in the queue Q has the following properties:

1. there is some witness W such that $W \subseteq B$, and
2. B contains at least one vertex receiving each colour under f.

Notice that we check the first condition before adding any subset A to Q (lines 4 and 27), and we check the second condition for any $A \neq U$ in line 25 (U necessarily satisfies condition 2 by construction of $\mathcal{F}$), so these two conditions are always satisfied. Thus, if we execute lines 10 to 12 with $A = X$, these conditions hold for X; note also that we only execute these lines with $A = X$ if $|X| = k$. Hence, as there is a witness $W \subseteq X$ where $|W| = |X| = k$, we must have $X = W$ and hence X is a witness. Moreover, as X must contain at least one vertex of each colour, and contains exactly k elements, it must be colourful.

Conversely, suppose that $W = \{w_1, \ldots, w_k\}$ is a witness such that $f(w_i) = i$ for each $1 \leq i \leq k$; we need to show that we will at some stage execute lines 10 to 12 with $A = W$. We argue that at the start of each execution of the while loop, if W has not yet been output, there must be some subset B in the queue such that $W \subseteq B$. This invariant clearly holds before the first execution of the loop (U will have been inserted into Q, as U contains at least one witness W). Now suppose that the invariant holds before starting some execution of the while loop. Either we execute lines 10 to 12 with $A = W$ on this iteration (in which case we are done), or else we proceed to line 19. Now, for $1 \leq i \leq k$, set j_i to be either 1 or 2 in such a way that $w_i \in A_i^{(j_i)}$. The subset $A_{\mathbf{j}}$, where $\mathbf{j} = (j_1, \ldots, j_k)$ will then pass both tests for insertion into Q, and $W \subseteq A_{\mathbf{j}}$ by construction, so the invariant holds when we exit the while loop. Since the algorithm only terminates when Q is empty, it follows that we must eventually execute lines 10 to 12 with $A = W$. ◀

The key property of k-perfect families of hash functions then implies that the algorithm will identify every witness; it remains only to ensure that we avoid outputting any witness more than once. This is the purpose of lines 10 to 12 in the pseudocode. We know from Lemma 4.2 that we find a given witness W while considering the hash-function f if and only if W is colourful with respect to f: thus, in order to determine whether we have found the witness in question before, it suffices to verify whether it is colourful with respect to any of the colourings previously considered. (The most obvious strategy for avoiding repeats would be to maintain a list of all the witnesses we have output so far, and check for membership of this list; however, in general there might be as many as $\binom{n}{k}$ witnesses, so both storing this list and searching it would be costly.) Hence we see that every witness is output exactly once, as required.

4.2 Expected running time

We know from Theorem 4.1 that a family $\mathcal{F}$ of k-perfect hash functions from U to $[k]$, with $|\mathcal{F}| = 2^{O(k)} \log n$, can be computed in time $2^{O(k)} n \log n$; thus line 1 can be executed in time $2^{O(k)} n \log n$ and the total number of iterations of the outer for-loop (lines 2 to 34) is at most $2^{O(k)} \log n$.

Moreover, it is clear that each iteration of the while loop (lines 7 to 33) makes at most 2^k oracle calls. If an oracle call can be executed in time $g(k) \cdot n^{O(1)}$ for some computable function g, then the total time required to perform each iteration of the while loop is at most $\max\{|\mathcal{F}|, kn + 2^k \cdot g(k) \cdot n^{O(1)}\} = 2^{O(k)} \cdot g(k) \cdot n^{O(1)}$.

Thus it remains to bound the expected number of iterations of the while loop in any iteration of the outer for-loop; we do this in the next lemma.

▶ **Lemma 4.3.** *The expected number of iterations of the while-loop in any given iteration of the outer for-loop is at most $N\,(1 + \lceil \log n \rceil)$, where N is the total number of witnesses in the instance.*

Proof. We fix an arbitrary $f \in \mathcal{F}$, and for the remainder of the proof restrict our attention to the iteration of the outer for-loop corresponding to f.

We can regard this iteration of the outer for-loop as the exploration of a search tree, with each node of the search tree indexed by some subset of U. The root is indexed by U itself, and every node has up to 2^k children, each child corresponding to a different way of selecting one of the two randomly constructed subsets for each colour. A node may have strictly fewer than 2^k children, as we use the oracle to prune the search tree (line 27), omitting the exploration of branches indexed by a subset of U that does not contain any witness (colourful or otherwise). Note that the search tree defined in this way has depth at most $\lceil \log n \rceil$: at each level, the size of each colour-class in the indexing subset is halved (up to integer rounding).

In this search tree model of the algorithm, each node of the search tree corresponds to an iteration of the while-loop, and vice versa. Thus, in order to bound the expected number of iterations of the while-loop, it suffices to bound the expected number of nodes in the search tree.

Our oracle-based pruning method means that we can associate with every node v of the search tree some representative witness W_v (not necessarily colourful), such that W_v is entirely contained in the subset of U which indexes v. (Note that the choice of representative witness for a given node need not be unique.) We know that in total there are N witnesses; our strategy is to bound the expected number of nodes, at each level of the search tree, for which any given witness can be the representative.

For a given witness W, we define a random variable $X_{W,d}$ to be the number of nodes at depth d (where the root has depth 0, and children of the root have depth 1, etc.) for which W could be the representative witness. Since every node has some representative witness, it follows that the total number of nodes in the search tree is at most

$$\sum_{W \text{ a witness}} \sum_{d=0}^{\lceil \log n \rceil} X_{W,d}.$$

Hence, by linearity of expectation, the expected number of nodes in the search tree is at most

$$\sum_{W \text{ a witness}} \sum_{d=0}^{\lceil \log n \rceil} \mathbb{E}\left[X_{W,d}\right] \quad \leq \quad N \sum_{d=0}^{\lceil \log n \rceil} \max_{W \text{ a witness}} \mathbb{E}\left[X_{W,d}\right].$$

In the remainder of the proof, we argue that $\mathbb{E}[X_{W,d}] \leq 1$ for all W and d, which will give the required result.

Observe first that, if W is in fact a colourful witness with respect to f, then $X_{W,d} = 1$ for every d: given a node whose indexing set contains W, exactly one of its children will be indexed by a set that contains W. So we will assume from now on that W intersects precisely ℓ colour classes, where $\ell < k$.

If a given node is indexed by a set that contains W, we claim that the probability that W is contained in the set indexing at least one of its children is at most $\frac{1}{2}^{k-\ell}$. For this to happen, it must be that for each colour i, all elements of W having colour i are assigned to the same set in the random partition. If c_i elements in W have colour i, the probability of this happening for colour i is at most $\left(\frac{1}{2}\right)^{c_i - 1}$ (the first vertex of colour i can be assigned to either set, and each subsequent vertex has probability at most $\frac{1}{2}$ of being assigned to this

same set). Since the random partitions for each colour class are independent, the probability that the witness W survives is at most

$$\prod_{W\cap f^{-1}(i)\neq\emptyset}\left(\frac{1}{2}\right)^{c_i-1}=\left(\frac{1}{2}\right)^{k-|\{i:W\cap f^{-1}(i)\neq\emptyset\}|}=\left(\frac{1}{2}\right)^{k-\ell}.$$

Moreover, if W is contained in the set indexing at least one of the child nodes, it will be contained in the indexing sets for exactly $2^{k-\ell}$ child nodes: we must select the correct subset for each colour-class that intersects W, and can choose arbitrarily for the remaining $k-\ell$ colour classes. Hence, for each node indexed by a set that contains W, the *expected* number of children which are also indexed by sets containing W is at most $\left(\frac{1}{2}\right)^{k-\ell}\cdot 2^{k-\ell}=1$.

We now prove by induction on d that $\mathbb{E}\left[X_{W,d}\right]\leq 1$ (in the case that W is not colourful). The base case for $d=0$ is trivial (as there can only be one node at depth 0), so suppose that $d>0$ and that the result holds for smaller values. Then, if $\mathbb{E}[Y|Z=s]$ is the conditional expectation of Y given that $Z=s$,

$$\begin{aligned}\mathbb{E}[X_{W,d}]&=\sum_{t\geq 0}\mathbb{E}[X_{W,d}|X_{W,d-1}=t]\,\mathbb{P}[X_{W,d-1}=t]\\&\leq\sum_{t\geq 0}t\,\mathbb{P}[X_{W,d-1}=t]\\&=\mathbb{E}[X_{W,d-1}]\\&\leq 1,\end{aligned}$$

by the inductive hypothesis, as required. Hence $\mathbb{E}[X_{W,d}]\leq 1$ for *any* witness W, which completes the proof. ◀

By linearity of expectation, it then follows that the expected total number of executions of the while loop will be at most $|\mathcal{F}|\cdot N\left(1+\lceil\log n\rceil\right)$, and hence that the expected number of oracle calls made during the execution of the algorithm is at most $2^{O(k)}\log^2 n\cdot N$. Moreover, if an oracle call can be executed in time $g(k)\cdot n^{O(1)}$ for some computable function g, then the expected total running time of the algorithm is

$$2^{O(k)}\cdot g(k)\cdot n^{O(1)}\cdot N,$$

as required.

5 Application to counting

There is a close relationship between the problems of counting and enumerating all witnesses in a self-contained k-witness problem, since any enumeration algorithm can very easily be adapted into an algorithm that simply counts the witnesses. However, in the case that the number N of witnesses is large, an enumeration algorithm necessarily takes time at least $O(N)$, whereas we might hope for much better if our goal is simply to determine the total number of witnesses.

The family of self-contained k-witness problems studied here includes subgraph problems, whose parameterised complexity from the point of view of counting has been a rich topic for research in recent years [13, 16, 17, 9, 10, 20, 15]. Many such counting problems, including those whose decision problem belongs to FPT, are known to be #W[1]-complete (see [14] for background on the theory of parameterised counting complexity). In this section we demonstrate how our enumeration algorithm can be adapted to give an efficient (randomised)

algorithm to solve the counting version of a self-contained k-witness problem *when the total number of witnesses is small*. This complements the fact that a simple random sampling algorithm can be used for *approximate* counting when the number of witnesses is very large [20, Lemma 3.4], although there remain many situations which are not covered by either result.

▶ **Theorem 5.1.** *Let Π be a self-contained k-witness problem, and suppose that $0 < \delta \leq \frac{1}{2}$ and $M \in \mathbb{N}$. Then there exists a randomised algorithm which makes at most $2^{O(k)} \log^2 n \, M \log(\delta^{-1})$ calls to a deterministic decision oracle for Π, and*

1. *if the number of witnesses in the instance of Π is at most M, outputs with probability at least $1 - \delta$ the exact number of witnesses in the instance;*
2. *if the number of witnesses in the instance of Π is strictly greater than M, always outputs "More than M."*

Moreover, if there is an algorithm solving the decision version of Π in time $g(k) \cdot n^{O(1)}$ for some computable function g, then the expected running time of the randomised algorithm is bounded by $2^{O(k)} \cdot g(k) \cdot n^{O(1)} \cdot M \cdot \log(\delta^{-1})$.

Proof. Note that our randomised enumeration algorithm can very easily be adapted to give a randomised counting algorithm which runs in the same time as the enumeration algorithm but, instead of listing all witnesses, simply outputs the total number of witnesses when it terminates. We may compute explicitly the expected running time of our randomised enumeration algorithm (and hence its adaptation to a counting algorithm) for a given self-contained k-witness problem Π in terms of n, k and the total number of witnesses, N. We will write $T(\Pi, n, k, N)$ for this expected running time.

Now consider an algorithm A, in which we run our randomised counting algorithm for at most $2T(\Pi, n, k, M)$ steps; if the algorithm has terminated within this many steps, A outputs the value returned, otherwise A outputs "FAIL". Since our randomised counting algorithm is always correct (but may take much longer than the expected time), we know that if A outputs a numerical value then this is precisely the number of witnesses in our problem instance. If the number of witnesses is in fact at most M, then the expected running time of the randomised counting algorithm is bounded by $T(\Pi, n, k, M)$, so by Markov's inequality the probability that it terminates within $2T(\Pi, n, k, M)$ steps is at least $1/2$. Thus, if we run A on an instance in which the number of witnesses is at most M, it will output the exact number of witnesses with probability at least $1/2$.

To obtain the desired probability of outputting the correct answer, we repeat A a total of $\lceil \log(\delta^{-1}) \rceil$ times. If any of these executions of A terminates with a numerical answer that is at most M, we output this answer (which must be the exact number of witnesses by the argument above); otherwise we output "More than M."

If the total number of witnesses is in fact less than or equal to M, we will output the exact number of witnesses unless A outputs "FAIL" every time it is run. Since in this case A outputs "FAIL" independently with probability at most $1/2$ each time we run it, the probability that we output "FAIL" on every one of the $\lceil \log(\delta^{-1}) \rceil$ repetitions is at most $(1/2)^{\lceil \log(\delta^{-1}) \rceil} \leq 2^{\log \delta} = \delta$. Finally, note that if the number of witnesses is strictly greater than M, we will always output "More than M" since every execution of A must in this case return either "FAIL" or a numerical answer greater than M.

The total running time is at most $O\left(\log(\delta^{-1}) \cdot T(\Pi, n, k, M)\right)$ and hence, using the bound on the running time of our enumeration algorithm from Theorem 1.1, is bounded by $2^{O(k)} \cdot g(k) \cdot n^{O(1)} \cdot M \cdot \log(\delta^{-1})$, as required. ◀

6 Conclusions and open problems

Many well-known combinatorial problems satisfy the definition of the self-contained k-witness problems considered in this paper. We have shown that, given access to a deterministic oracle for the decision version of a self-contained k-witness problem (answering the question "does this subset of the universe contain at least one witness?"), there is a randomised algorithm which is guaranteed to enumerate all witnesses and whose expected number of oracle calls is at most $2^{O(k)} \log^2 n \cdot N$, where N is the total number of witnesses. Moreover, if the decision problem belongs to FPT (as is the case for many self-contained k-witness problems), our enumeration algorithm is an expected-output-fpt algorithm.

This result also has implications for counting the number of witnesses. In particular, if the total number of witnesses is small (at most $f(k) \cdot n^{O(1)}$ for some computable function f) then our enumeration algorithm can easily be adapted to give an fpt-algorithm that will, with high probability, determine exactly the number of witnesses in an instance of a self-contained k-witness problem. This in fact satisfies the conditions for a FPTRAS (Fixed Parameter Tractable Randomised Approximation Scheme, as defined in [2]), but without requiring the full flexibility that this definition requires: with probability $1 - \delta$ we will output the exact number of witnesses, rather than just an answer that is within a factor of $1 \pm \epsilon$ of this quantity.

While the enumeration problem can be solved in a more straightforward fashion for self-contained k-witness problems that have certain additional properties, we demonstrated that several self-contained k-witness problems do not have these properties, unless FPT=W[1]. A natural line of enquiry arising from this work would be the characterisation of those self-contained k-witness problems that do have the additional properties, namely those for which an fpt-algorithm for the decision version gives rise to an fpt-algorithm for the extension version of the decision problem.

Another key question that remains open after this work is whether the existence of an fpt-algorithm for the decision version of a self-contained k-witness problem is sufficient to guarantee the existence of an (expected-)delay-fpt or (expected-)incremental-fpt algorithm for the enumeration problem. Finally, it would be interesting to investigate whether the randomised algorithm given here can be derandomised.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

2 V. Arvind and Venkatesh Raman. Approximation algorithms for some parameterized counting problems. In *ISAAC 2002*, volume 2518 of *LNCS*, pages 453–464. Springer-Verlag Berlin Heidelberg, 2002.

3 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. arXiv:1007.1161 [cs.DS], 2010.

4 Andreas Björklund, Petteri Kaski, and Łukasz Kowalik. Probably Optimal Graph Motifs. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *LIPIcs*, pages 20–31. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2013. doi:10.4230/LIPIcs.STACS.2013.20.

5 Andreas Björklund, Petteri Kaski, and Łukasz Kowalik. Fast witness extraction using a decision oracle. In *Algorithms – ESA 2014*, volume 8737 of *LNCS*, pages 149–160. Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-662-44777-2_13.

6 Andreas Björklund, Petteri Kaski, Łukasz Kowalik, and Juho Lauri. Engineering motif search for large graphs. In *2015 Proc. of the Seventeenth Workshop on Algorithm*

Engineering and Experiments (ALENEX), pages 104–118. SIAM, 2015. doi:10.1137/1.9781611973754.10.

7 Nadia Creignou, Raïda Ktari, Arne Meier, Julian-Steffen Müller, Frédéric Olive, and Heribert Vollmer. Parameterized enumeration for modification problems. In *Language and Automata Theory and Applications*, volume 8977 of *LNCS*, pages 524–536. Springer International Publishing, 2015. doi:10.1007/978-3-319-15579-1_41.

8 Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. In *Mathematical Foundations of Computer Science 2013*, volume 8087 of *LNCS*, pages 290–301. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-40313-2_27.

9 Radu Curticapean. Counting matchings of size k is #W[1]-hard. In *Automata, Languages, and Programming*, volume 7965 of *LNCS*, pages 352–363. Springer Berlin Heidelberg, 2013.

10 Radu Curticapean and Dániel Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2014*, 2014.

11 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer London, 2013.

12 Henning Fernau. On parameterized enumeration. In *Computing and Combinatorics*, volume 2387 of *LNCS*, pages 564–573. Springer Berlin Heidelberg, 2002. doi:10.1007/3-540-45655-4_60.

13 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.

14 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

15 Mark Jerrum and Kitty Meeks. The parameterised complexity of counting even and odd induced subgraphs. arXiv:1410.3375 [math.CO], to appear in *Combinatorica*, 2014.

16 Mark Jerrum and Kitty Meeks. The parameterised complexity of counting connected subgraphs and graph motifs. *Journal of Computer and System Sciences*, 81(4):702–716, 2015. doi:10.1016/j.jcss.2014.11.015.

17 Mark Jerrum and Kitty Meeks. Some hard families of parameterised counting problems. *ACM Transactions on Computation Theory*, 7(3), June 2015. doi:10.1145/2786017.

18 Samir Khuller and Vijay V. Vazirani. Planar graph coloring is not self-reducible, assuming P $\neq$ NP. *Theoretical Computer Science*, 88(1):183–189, 1991. doi:10.1016/0304-3975(91)90081-C.

19 Eugene L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):401–405, 1972. doi:10.1287/mnsc.18.7.401.

20 Kitty Meeks. The challenges of unbounded treewidth in parameterised subgraph counting problems. *Discrete Applied Mathematics*, 198:170–194, 2016. doi:10.1016/j.dam.2015.06.019.

21 C. P. Schnorr. Optimal algorithms for self-reducible problems. In *Proc. of the 3rd ICALP*, pages 322–337. Edinburgh University Press, 1976.

Backdoors for Linear Temporal Logic*

Arne Meier[1], Sebastian Ordyniak[2], Ramanujan Sridharan[3], and Irena Schindler[4]

1 **Leibniz Universität Hannover, Germany**
meier@thi.uni-hannover.de
2 **TU Wien, Austria**
sordyniak@gmail.com
3 **TU Wien, Austria**
ramanujan@ac.tuwien.ac.at
4 **Leibniz Universität Hannover, Germany**
schindler@thi.uni-hannover.de

Abstract

In the present paper, we introduce the backdoor set approach into the field of temporal logic for the global fragment of linear temporal logic. We study the parameterized complexity of the satisfiability problem parameterized by the size of the backdoor. We distinguish between backdoor detection and evaluation of backdoors into the fragments of Horn and Krom formulas. Here we classify the operator fragments of globally-operators for past/future/always, and the combination of them. Detection is shown to be fixed-parameter tractable (FPT) whereas the complexity of evaluation behaves differently. We show that for Krom formulas the problem is paraNP-complete. For Horn formulas, the complexity is shown to be either fixed parameter tractable or paraNP-complete depending on the considered operator fragment.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Linear Temporal Logic, Parameterized Complexity, Backdoor Sets

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.23

1 Introduction

Temporal logic is one of the most important formalisms in the area of program verification and validation of specification consistency. Most notable are the seminal contributions of Kripke [21], Pnueli [30], Emerson, Clarke, and Halpern [14, 7] to name a few. There exist several different variants of temporal logic from which, best known are the computation tree logic CTL, the linear temporal logic LTL, and the full branching time logic CTL*. In this paper, we will consider the global fragment of LTL for formulas in *separated normal form* (SNF) which has been introduced by Fisher [16]. This normal form is a generalization of the conjunctive normal form from propositional logic to linear temporal logic with future and past modalities interpreted over the flow of time, i.e., the frame of the integers $(\mathbb{Z}, <)$. In SNF the formulas are divided into a past, a present, and a future part. Technically this normal form is not a restriction since one can always translate an arbitrary LTL formula to a satisfiability-equivalent formula in SNF in linear time in the original formula [16]. In fact, the restriction to SNF normal form is crucial for us, because it is known that syntactical restrictions of arbitrary LTL formulas such as Horn or Krom do not lead to tractability [4].

* The first and last author gratefully acknowledge the support by the German Research Foundation DFG for their grant ME 4279/1-1.

Table 1 Results overview. The term "any" refers to any combination of $\boxtimes, \Box_F, \Box_P$, whereas "above" denotes that the lower bound from the cell above applies.

Problem	Operators	HORN	KROM
Detection	any	FPT (Thm. 5)	FPT (Thm. 6)
Evaluation	$\boxtimes$	FPT (Thm. 9)	paraNP-c. (Thm. 10)
	$\Box_F, \Box_P$	paraNP-c. (Thm. 11)	paraNP-c. (above)
	one of $\Box_F, \Box_P$	open	paraNP-c. (Cor. 12)
LTL-SAT	$\boxtimes, \Box_F, \Box_P$	P [2]	NP-c. [2]
	$\boxtimes$	P [2]	NL [2]

LTL and its two main associated computational problems LTL model checking and LTL satisfiability have been deeply investigated in the past. In this work we focus on the LTL satisfiability problem, i.e., given an LTL formula the question is whether there is a temporal interpretation that satisfies the formula. Sistla and Clarke classified the computational complexity of the satisfiability problem to be PSPACE-complete [34]. Then, later, several restrictions of the unrestricted problem have been considered. These approaches considered operator fragments [27], Horn formulas [4], temporal operator fragments, temporal depth, and number of propositional variables [8], the use of negation [26], an XOR fragment [11], an application of Post's lattice [3], and the SNF fragment [2].

In contrast to LTL satisfiability where the search for fruitful parameterization has so far been rather unsuccessful [25], various important parameterizations have been identified for SAT [35, 5, 28]. One very prominent and well-studied structural parameterization for SAT are so-called backdoor sets. Backdoors are small sets of variables of a SAT instance that represent "clever reasoning shortcuts" through the search space. Backdoor sets have been widely used in the areas of propositional satisfiability [36, 31, 9, 33, 20, 10, 19], and also for material discovery [24], abductive reasoning [29], argumentation [13], planning [22, 23], and quantified Boolean formulas [32]. A backdoor set is defined with respect to some fixed *base class* for which the computational problem under consideration is polynomial-time tractable. For instance, in the case of the propositional satisfiability problem, a backdoor set B for a given CNF formula ϕ into the base class of Horn formulas is a set of variables such that for every assignment of the variables in B it holds that the reduced formula, i.e., the formula obtained after applying the assignment to ϕ, is Horn. Given such a backdoor set one can decide the satisfiability of ϕ in time $O(2^{|B|}p(|\phi|))$ by enumerating the $2^{|B|}$ assignments of the variables in B and for each such assignment solving the remaining formula in time $p(|\phi|)$, where p is a polynomial given by the base class. Hence, once a small backdoor set is identified the satisfiability check is *fixed-parameter tractable* for the parameter backdoor size. Since the backdoor set is usually not provided with the input, it is crucial that small backdoor sets to a given base class can be found efficiently. When employing the backdoor approach one therefore usually considers two subtasks the so-called *detection* and *evaluation* problem, where the former is the task to identify a small backdoor set and the later concerns the solution of the problem using the backdoor set.

Our Contribution. In this paper, we introduce a notion of backdoors for the global fragment of LTL formulas that are given in SNF. Namely, we consider backdoor sets to the base classes that have recently been identified by Artale et al. [2]. These base classes are defined by both restrictions on the allowed temporal operators (i.e., to a subset of $\{\boxtimes, \Box_P, \Box_F\}$) and

restrictions on the clauses to be either HORN or KROM. We show that surprisingly a notion of backdoor sets very similar to the strong backdoor sets employed for SAT [18] can also be successfully applied to LTL formulas. Whereas the detection of these backdoor sets can be achieved via efficient fpt-algorithms for all the considered fragments (using algorithms similar to the algorithms employed in the context of SAT), the evaluation of these backdoor sets turns out to be much more involved. In particular, we obtain tractability of the evaluation problem for HORN formulas using only the always operator. In fact, LTL restricted to only the always operator, is already quite interesting, since it allows one to express "Safety" properties of a system. For almost all of the remaining cases we show that the evaluation problem is paraNP-hard. Moreover, the techniques used to show these results are very different from and more involved than the techniques employed for SAT, i.e., in the context of SAT the backdoor set evaluation problem is trivial. Our results are summarized in Table 1.

2 Preliminaries

Parameterized Complexity. A good introduction into the field of parameterized complexity is given by Downey and Fellows [12]. A *parameterized problem* Π is a tuple (Q, κ) such that the following holds. $Q \subseteq \Sigma^*$ is a language over an alphabet Σ, and $\kappa\colon \Sigma^* \to \mathbb{N}$ is a computable function; then κ also is called the *parameterization (of* Π*)*.

If there is a deterministic Turing machine M and a computable function $f\colon \mathbb{N} \to \mathbb{N}$ s.t. for every instance $x \in \Sigma^*$ (i) M decides correctly if $x \in Q$, and (ii) M has a runtime bounded by $f(\kappa(x)) \cdot |x|^{O(1)}$, then we say that M is an *fpt-algorithm for* Π and that Π is *fixed-parameter tractable* (or in the class FPT). If M is non-deterministic, then Π belongs to the class paraNP. One way to show paraNP-hardness of a parameterized problem (Q, κ) is to show that Q is NP-hard for a specific, fixed value of κ, i.e., there exists a constant $\ell \in \mathbb{N}$ such that $(Q, \kappa)_\ell := \{x \mid x \in Q \text{ and } \kappa(x) = \ell\}$ is NP-hard.

Temporal Logic. We assume familiarity with standard notions of propositional logic. Let PROP be a finite set of propositions and $\bot/\top$ abbreviate the constants false/true. The syntax of the global fragment of LTL is defined by the following EBNF:

$$\varphi ::= \bot \mid \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box_{\mathrm{P}}\, \varphi \mid \Box_{\mathrm{F}}\, \varphi \mid \boxdot \varphi,$$

where $p \in \mathsf{PROP}$. Here $\Box_{\mathrm{P}}\, \varphi$ can be read as "φ holds in every point in the past", $\Box_{\mathrm{F}}\, \varphi$ as "φ holds in every point in the future", and $\boxdot \varphi$ as "φ holds always". We also will make use of well-known shortcuts such as $\rightarrow, \leftrightarrow$. Now we define the semantics of these formulas. Here, we interpret LTL formulas over the flow of time $(\mathbb{Z}, <)$ (for further information on this approach, see, e.g., Gabbay et al. [17]). Note that all our results will also apply one-to-one if the formulas are evaluated over the set of natural numbers instead of the set of all integers.

▶ **Definition 1** (Temporal Semantics)**.** Let PROP be a finite set of propositions. A *temporal interpretation* $\mathfrak{M} = (\mathbb{Z}, <, V)$ is a mapping from propositions to moments of time, i.e., $V\colon \mathsf{PROP} \to \mathcal{P}(\mathbb{Z})$. The satisfaction relation $\models$ is then defined as follows where $n \in \mathbb{Z}$, $\varphi, \psi \in$ LTL

$\mathfrak{M}, n \models \top$ always, and $\mathfrak{M}, n \models \bot$ never,
$\mathfrak{M}, n \models p$ iff $n \in V(p)$,
$\mathfrak{M}, n \models \neg\varphi$ iff $\mathfrak{M}, n \not\models \varphi$
$\mathfrak{M}, n \models \varphi \vee \psi$ iff $\mathfrak{M}, n \models \varphi$ or $\mathfrak{M}, n \models \psi$
$\mathfrak{M}, n \models \varphi \wedge \psi$ iff $\mathfrak{M}, n \models \varphi$ and $\mathfrak{M}, n \models \psi$

Table 2 Considered normal forms. Restrictions refer to equation (2).

class	description	restrictions on n, m
CNF	no restrictions on (2)	–
HORN	at most one positive temporal literal	$m \leq 1$
KROM	binary clauses	$n + m \leq 2$

$\mathfrak{M}, n \models \Box_{\mathrm{F}} \varphi$ iff for all $k > n$ it holds $\mathfrak{M}, k \models \varphi$
$\mathfrak{M}, n \models \Box_{\mathrm{P}} \varphi$ iff for all $k < n$ it holds $\mathfrak{M}, k \models \varphi$
$\mathfrak{M}, n \models \boxtimes \varphi$ iff for all $k \in \mathbb{Z}$ it holds $\mathfrak{M}, k \models \varphi$

We say that φ is *satisfiable* if there is a temporal interpretation $\mathfrak{M}$ such that $\mathfrak{M}, 0 \models \varphi$. Then $\mathfrak{M}$ is also referred to as a *(temporal) model (of φ)*. Sometimes we also directly write $\mathfrak{M}(p)$ instead of $V(p)$.

As shown by Fisher et al. every LTL formula considered over the frame $(\mathbb{Z}, <)$ has a satisfiability-equivalent formula in the separated normal form SNF [15] (and can also be constructed in linear time [16]). We follow the notation of SNF formulas by Artale et al. [2] and directly restrict them to the relevant global fragment of this study:

$$\lambda ::= \bot \mid p \mid \Box_{\mathrm{F}} \lambda \mid \Box_{\mathrm{P}} \lambda \mid \boxtimes \lambda, \tag{1}$$
$$\varphi ::= \lambda \mid \neg\lambda \mid \varphi \wedge \varphi \mid \boxtimes(\neg\lambda_1 \vee \cdots \vee \neg\lambda_n \vee \lambda_{n+1} \vee \cdots \lambda_{n+m}), \tag{2}$$

where λ is called a *temporal literal* and φ is said to be in *clausal normal form*.

Note that the operator name G instead of $\Box_{\mathrm{F}}$ often occurs in literature. Yet, in contrast to $\mathsf{G}\varphi$, for $\Box_{\mathrm{F}} \varphi$ it is not required that φ holds in the present world. We distinguish fragments of LTL by adding superscripts and subscripts as follows. If $O \subseteq \{\Box_{\mathrm{F}}, \Box_{\mathrm{P}}, \boxtimes\}$ is an operator subset then LTL^{O} is the fragment of LTL consisting of formulas that are allowed to only use temporal operators from O for temporal literals, i.e., it is a constraint on the allowed operators in equation (1) from above. We also consider restrictions of the clausal normal form in (2): $\boxtimes(\neg\lambda_1 \vee \cdots \vee \neg\lambda_n \vee \lambda_{n+1} \vee \cdots \lambda_{n+m})$. Table 2 lists the relevant cases for this study. If $\alpha \in \{\mathrm{CNF}, \mathrm{HORN}, \mathrm{KROM}\}$ then LTL_α is the set of formulas where the subformulas of the type $\boxtimes(\neg\lambda_1 \vee \cdots \vee \neg\lambda_n \vee \lambda_{n+1} \vee \cdots \lambda_{n+m})$ (3), obey the normal form α.

The following lemma shows a log-space constructible normal form which prohibits deep nesting of temporal operators of the investigated formulas.

▶ **Lemma 2** ([2, Lemma 2]). *Let $\mathcal{L} \in \{\mathrm{LTL}_\alpha^{\Box_{\mathrm{F}},\Box_{\mathrm{P}}}, \mathrm{LTL}_\alpha^{\Box_{\mathrm{F}}}, \mathrm{LTL}_\alpha^{\Box_{\mathrm{P}}}, \mathrm{LTL}_\alpha^{\boxtimes}\}$ be a formula class for $\alpha \in \{\mathrm{CNF}, \mathrm{HORN}, \mathrm{KROM}\}$. For any formula $\varphi \in \mathcal{L}$, one can construct, in log-space, a satisfiability-equivalent $\mathcal{L}$-formula $\Psi \wedge \boxtimes \Phi$, where Ψ is a conjunction of propositional variables from Φ, and Φ is a conjunction of clauses of the form (3) containing only $\Box_{\mathrm{F}}, \Box_{\mathrm{P}}$ for $\mathrm{LTL}_\alpha^{\Box_{\mathrm{F}},\Box_{\mathrm{P}}}$, $\Box_{\mathrm{F}}$ for $\mathrm{LTL}_\alpha^{\Box_{\mathrm{F}}}$, $\Box_{\mathrm{P}}$ for $\mathrm{LTL}_\alpha^{\Box_{\mathrm{P}}}$, and only $\boxtimes$ for $\mathrm{LTL}_\alpha^{\boxtimes}$, in which the temporal operators are not nested.*

In the following sections we consider only formulas given in this normal form $\Psi \wedge \boxtimes \Phi$.

3 Introduction of backdoors for the global fragment of LTL

In the following, we will introduce a notion of backdoors for formulas in the global fragment of linear temporal logic. The definition of these backdoors turns out to be very similar to

the definition of the so-called strong backdoor sets for propositional formulas [18]. The main difference is that whenever a propositional variable is in the backdoor set then also all of its temporal literals are required to be in the backdoor set as well. A consequence of this is that in contrast to propositional formulas, where a backdoor set needs to consider all assignments of the backdoor set variables, we only need to consider assignments that are consistent between propositional variables and their temporal literals.

Let $\mathcal{O}$ be a set of operators. An assignment $\theta\colon \mathrm{Vars}(\phi) \cup \{\, Ox \mid x \in \mathrm{Vars}(\phi) \wedge O \in \mathcal{O} \,\} \to \{0,1\}$ is *consistent* if for every $x \in \mathrm{Vars}(\phi)$ it holds that if $\theta(\circledast x) = 1$, then also $\theta(\Box_P x) = 1$, $\theta(\Box_F x) = 1$, and $\theta(x) = 1$.

▶ **Definition 3** (Backdoors). Let $\mathcal{C}$ be a class of CNF-formulas, $\mathcal{O}$ be a set of operators, and ϕ be an $\mathrm{LTL}^{\mathcal{O}}_{\mathrm{CNF}}$ formula. A set $X \subseteq \mathrm{Vars}(\phi)$ is a *(strong)* $(\mathcal{C}, \mathcal{O})$*-backdoor* if for every consistent assignment $\theta\colon X \cup \{Ox \mid x \in X, O \in \mathcal{O}\} \to \{0,1\}$ it holds that $\phi[\theta]$ is in $\mathcal{C}$.

The *reduct* $\phi[\theta]$ is defined similarly to that for standard CNF-formulas, i.e., all clauses that contain a satisfied literal are deleted, and all falsified literals are deleted from their clauses. Here empty clauses are substituted by false, and the empty formula by true. Sometimes if the context of $\mathcal{O}$ is clear, we omit to state it and just mention the backdoor class $\mathcal{C}$.

To exploit backdoor sets to obtain efficient fpt-algorithms for LTL one needs to accomplish two tasks: first, one needs to find a small backdoor set, and then one needs to show how the backdoor set can be exploited to efficiently evaluate the formula. This leads to the following problem definitions for every class $\mathcal{C}$ of formulas and set of operators $\mathcal{O}$.

Problem: $\mathrm{Eval}^{\mathcal{O}}(\mathcal{C})$ — Backdoor evaluation to $\mathrm{LTL}^{\mathcal{O}}_{\mathcal{C}}$.
Input: $\mathrm{LTL}^{\mathcal{O}}_{\mathrm{CNF}}$ formula ϕ, strong $(\mathcal{C}, \mathcal{O})$-backdoor X.
Parameter: $|X|$.
Question: Is ϕ satisfiable?

Problem: $\mathrm{Detect}^{\mathcal{O}}(\mathcal{C})$ — Backdoor detection to $\mathrm{LTL}^{\mathcal{O}}_{\mathcal{C}}$.
Input: $\mathrm{LTL}^{\mathcal{O}}_{\mathrm{CNF}}$ formula ϕ, integer $k \in \mathbb{N}$.
Parameter: k.
Task: Find a strong $(\mathcal{C}, \mathcal{O})$-backdoor of size $\leq k$ if one exists.

Of course, this approach is only meaningful if one considers target classes that have polynomial time solvable satisfiability problems. Artale et al. have shown [2] that satisfiability for $\mathrm{LTL}^{\circledast}_{\mathrm{HORN}}$ and $\mathrm{LTL}^{\circledast}_{\mathrm{KROM}}$ are solvable in P. Adding $\Box_{\mathrm{F}}, \Box_{\mathrm{P}}$ to the set of allowed operators makes the KROM fragment NP-complete whereas for HORN formulas the problem stays in P. Therefore we will consider in the following only KROM and HORN formulas. Moreover, note that when considering arbitrary CNF formulas instead of HORN or KROM formulas, then $\mathrm{LTL}^{\mathcal{O}}_{\mathrm{CNF}}$ is known to be NP-complete for any (even empty) subset $\mathcal{O} \subseteq \{\Box_{\mathrm{F}}, \Box_{\mathrm{P}}, \circledast\}$ [2].

4 Backdoor set detection

In this section, we show that finding strong $\mathcal{C}$-backdoor sets (under the parameter size of the set) is fixed-parameter tractable if $\mathcal{C}$ is either HORN or KROM. The algorithms that we will present are very similar to the algorithms that are known for the detection of strong backdoors for propositional CNF formulas [18].

We first show how to deal with the fact that we only need to consider consistent assignments. The following observation is easily witnessed by the fact that if one of $\Box_{\mathrm{P}} x, \Box_{\mathrm{F}} x, x$ does not hold then $\neg \circledast x$ is true.

▶ **Observation 4.** *Let* $\phi := \Psi \wedge \boxtimes \Phi$ *be an* $\mathrm{LTL}^{\Box_P,\Box_F,\boxtimes}$ *formula. Then any clause* C *of* Φ *containing* $\neg\boxtimes x$ *and (at least) one of* $\Box_P x$, $\Box_F x$ *or* x *for some variable* $x \in \mathrm{Vars}(\phi)$ *is tautological and can thus be removed from* ϕ *(without changing the satisfiability of* ϕ*).*

Observe that the tautological clauses above are exactly the clauses that are satisfied by every consistent assignment. It follows that once these clauses are removed from the formula, it holds that for every clause C of ϕ there is a consistent assignment θ such that C is not satisfied by θ. This observation will be crucial for our detection algorithms described below.

▶ **Theorem 5.** *For every* $\mathcal{O} \subseteq \{\boxtimes, \Box_P, \Box_F\}$, $\mathrm{Detect}^{\mathcal{O}}(\textsc{horn})$ *is in* FPT.

Proof. Let $\mathcal{O} \subseteq \{\boxtimes, \Box_P, \Box_F\}$. We will reduce $\mathrm{Detect}^{\mathcal{O}}(\textsc{horn})$ to the problem VertexCover which is well-known to be fixed-parameter tractable (parameterized by the solution size) and which can actually be solved very efficiently in time $O(1.2738^k + kn)$ [6], where k is the size of the vertex cover and n the number of vertices in the input graph. Recall that given an undirected graph G and an integer k, VertexCover asks whether there is a subset $C \subseteq V(G)$ of size at most k (which is called a vertex cover of G) such that $C \cap e \neq \emptyset$ for every $e \in E(G)$. Given an $\mathrm{LTL}^{\mathcal{O}}$ formula $\phi := \Psi \wedge \boxtimes \Phi$, we will construct an undirected graph G such that ϕ has a strong HORN-backdoor of size at most k if and only if G has a vertex cover of size at most k. The graph G has vertex set $\mathrm{Vars}(\phi)$ and there is an edge between two vertices x and y in G if and only if there is a clause that contains at least two literals from $\{x, y\} \cup \{\, Ox, Oy \mid O \in \mathcal{O} \,\}$. Note that if $x = y$, the graph G contains a self-loop. We claim that a set $X \subseteq \mathrm{Vars}(\phi)$ is a strong HORN-backdoor if and only if X is a vertex cover of G.

Towards showing the forward direction, let $X \subseteq \mathrm{Vars}(\phi)$ be a strong HORN-backdoor set of ϕ. We claim that X is also a vertex cover of G. Suppose for a contradiction that X is not a vertex cover of G, i.e., there is an edge $\{x, y\} \in E(G)$ such that $X \cap \{x, y\} = \emptyset$. Because $\{x, y\} \in E(G)$, we obtain that there is a clause C in Φ that contains at least two literals from $\{x, y\} \cup \{\, Ox, Oy \mid O \in \mathcal{O} \,\}$. Moreover, because of Observation 4 there is a consistent assignment $\theta\colon X \cup \{\, Ox \mid x \in X \wedge O \in \mathcal{O}\} \to \{0, 1\}$ that falsifies all literals of C over variables in X. Consequently, $\phi[\theta]$ contains a sub-clause of C that still contains at least two literals from $\{x, y\} \cup \{\, Ox, Oy \mid O \in \mathcal{O} \,\}$. Hence, $\phi[\theta] \notin$ HORN, contradicting our assumption that X is a strong HORN-backdoor set of ϕ.

Towards showing the reverse direction, let $X \subseteq V(G)$ be a vertex cover of G. We claim that X is also a strong HORN-backdoor of ϕ. Suppose for a contradiction that this is not the case, then there is an (consistent) assignment $\theta\colon X \cup \{Ox \mid x \in X \wedge O \in \mathcal{O}\} \to \{0, 1\}$ and a clause C in $\phi[\theta]$ containing two positive literals say over variables x and y. We obtain that C contains at least two positive literals from $\{x, y\} \cup \{\, Ox, Oy \mid O \in \mathcal{O} \,\}$ and hence G contains the edge $\{x, y\}$, contradicting our assumption that X is a vertex cover of G. ◀

The proof of the following theorem can be found in the appendix.

▶ **Theorem 6.** *For every* $\mathcal{O} \subseteq \{\boxtimes, \Box_P, \Box_F\}$, $\mathrm{Detect}^{\mathcal{O}}(\textsc{krom})$ *is in* FPT.

Having shown that the detection problem is fixed-parameter tractable, we now proceed to the backdoor set evaluation problem. We begin by investigating this problem for the class HORN and show that the problem lies in FPT.

5 Backdoor set evaluation

5.1 Formulas using only the always operator

We showed in the previous section that strong backdoors can be found to the classes HORN and KROM in FPT time. In fact, this result holds independently of the considered temporal

operators. In this section, we will consider the question of efficiently *using* a backdoor set to decide the satisfiability of a formula in the case of formulas restricted to the $\boxtimes$ operator. We will show that this problem is in FPT for the class of HORN formulas but not for KROM formulas. Our fixed-parameter tractability result for HORN formulas largely depends on the special semantics of formulas restricted to the $\boxtimes$ operators. Hence, we will start by stating some properties of these formulas necessary to obtain our tractability result.

Let $\mathfrak{M} = (\mathbb{Z}, <, V)$ be a temporal interpretation. We denote by $\mathrm{Vars}(\mathfrak{M})$ the set of propositions (in the following referred to as variables) for which V is defined. For a set of variables $X \subseteq \mathrm{Vars}(\mathfrak{M})$, we denote by $\mathfrak{M}_{|X}$ the *projection* of $\mathfrak{M}$ onto X, i.e., the temporal interpretation $\mathfrak{M}_{|X} = (\mathbb{Z}, <, V_{|X})$, where $V_{|X}$ is only defined for the variables in X and $V_{|X}(x) = V(x)$ for every $x \in X$. For an integer z, we denote by $\mathbf{A}(\mathfrak{M}, z)$ the assignment $\theta\colon \mathrm{Vars}(\mathfrak{M}) \to \{0,1\}$ holding at world z in $\mathfrak{M}$, i.e., $\theta(v) = 1$ if and only if $z \in \mathfrak{M}(v)$ for every $v \in \mathrm{Vars}(\mathfrak{M})$. Moreover, for a set of worlds $Z \subseteq \mathbb{Z}$ we denote by $\mathbf{A}(\mathfrak{M}, Z)$ the set of all assignments occurring in some world in Z of $\mathfrak{M}$, i.e., $\mathbf{A}(\mathfrak{M}, Z) := \{\, \mathbf{A}(\mathfrak{M}, z) \mid z \in Z \,\}$. We also set $\mathbf{A}(\mathfrak{M})$ to be $\mathbf{A}(\mathfrak{M}, \mathbb{Z})$. For an assignment $\theta\colon X \to \{0,1\}$, we denote by $\mathbf{W}(\mathfrak{M}, \theta)$ the set of all worlds $z \in \mathbb{Z}$ of $\mathfrak{M}$ such that $\mathbf{A}(\mathfrak{M}, z)$ is equal to θ on all variables in X.

Let $\varphi := \Psi \wedge \boxtimes \Phi \in \mathrm{LTL}^{\boxtimes}_{\mathrm{CNF}}$. We denote by $\mathbf{CNF}(\Phi)$ the propositional CNF formula obtained from Φ after replacing each occurrence of $\boxtimes x$ in Φ with a fresh propositional variable (with the same name). For a set of variables V and a set of assignments $\mathbb{A}$ of the variables in V, we denote by $\mathbf{G}(\mathbb{A}, V)\colon \{\, \boxtimes v \mid v \in V \,\} \to \{0,1\}$ the assignment defined by setting $\mathbf{G}(\mathbb{A}, V)(\boxtimes v) = 1$ if and only if $\alpha(v) = 1$ for every $\alpha \in \mathbb{A}$. Moreover, if $\theta\colon V \to \{0,1\}$ is an assignment of the variables in V, we denote by $\mathbf{G}(\mathbb{A}, V, \theta)$ the assignment defined by setting $\mathbf{G}(\mathbb{A}, V, \theta)(v) = \theta(v)$ and $\mathbf{G}(\mathbb{A}, V, \theta)(\boxtimes v) = \mathbf{G}(\mathbb{A}, V)(\boxtimes v)$ for every $v \in V$. For a set $\mathbb{A}$ of assignments over V and an assignment $\theta\colon V' \to \{0,1\}$ with $V' \subseteq V$, we denote by $\mathbb{A}(\theta)$ the set of all assignments $\alpha \in \mathbb{A}$ such that $\alpha(v) = \theta(v)$ for every $v \in V'$.

For a set $\mathbb{A}$ of assignments over some variables V and a subset $V' \subseteq V$, we denote by $\mathbb{A}|_{V'}$ the *projection* of $\mathbb{A}$ onto V', i.e., the set of assignments $\alpha \in \mathbb{A}$ restricted to the variables in V'.

Intuitively the next lemma describes the translation of a temporal model into separate satisfiability checks for propositional formulas.

▶ **Lemma 7.** *Let $\varphi := \Psi \wedge \boxtimes \Phi \in \mathrm{LTL}^{\boxtimes}$. Then, φ is satisfiable if and only if there is a set $\mathbb{A}$ of assignments of the variables in φ and an assignment $\alpha_0 \in \mathbb{A}$ such that: α_0 satisfies Ψ and for every assignment $\alpha \in \mathbb{A}$ it holds that $\mathbf{G}(\mathbb{A}, \mathrm{Vars}(\varphi), \alpha)$ satisfies the propositional formula $\mathbf{CNF}(\Phi)$.*

Proof. Towards showing the forward direction assume that $\varphi := \Psi \wedge \boxtimes \Phi$ is satisfiable and let $\mathfrak{M}$ be a temporal interpretation witnessing this. We claim that the set of assignments $\mathbb{A} := \mathbf{A}(\mathfrak{M})$ together with the assignment $\alpha_0 := \mathbf{A}(\mathfrak{M}, 0)$ satisfy the conditions of the lemma.

Towards showing the reverse direction assume that $\mathbb{A} := \{\alpha_0, \ldots, \alpha_{|\mathbb{A}|}\}$ is as given in the statement of the lemma. We claim that the temporal interpretation $\mathfrak{M}$ defined below satisfies the formula φ. Let $\mathbb{Z}_{<0}$ be the set of all integers smaller than 0 and let $\mathbb{Z}_{>|\mathbb{A}|}$ be the set of all integers greater than $|\mathbb{A}|$. Then for every variable $v \in \mathrm{Vars}(\varphi)$, the set $\mathfrak{M}(v)$ contains the set $\{\, z \mid \alpha_z(v) = 1 \wedge 0 \leq z \leq |\mathbb{A}| \,\}$. Moreover, if $\alpha_0(v) = 1$, $\mathfrak{M}(v)$ also contains the set $\mathbb{Z}_{<0}$ and if $\alpha_{|\mathbb{A}|}(v) = 1$, $\mathfrak{M}(v)$ additionally contains the set $\mathbb{Z}_{>|\mathbb{A}|}$. It is easy to verify that $\mathfrak{M}, 0 \models \varphi$. ◀

Informally, the following lemma shows that for deciding the satisfiability of an $\mathrm{LTL}^{\boxtimes}$ formula, we only need to consider sets of assignments $\mathbb{A}$, whose size is linear (instead of exponential) in the number of variables.

▶ **Lemma 8.** *Let $\varphi := \Psi \wedge \boxdot\Phi \in \mathrm{LTL}^{\boxdot}$ and $X \subseteq \mathrm{Vars}(\varphi)$. Then φ is satisfiable if and only if there is a set Θ of assignments of the variables in X, an assignment $\theta_0 \in \Theta$, a set $\mathbb{A}$ of assignments of the variables in $\mathrm{Vars}(\varphi)$, and an assignment $\alpha_0 \in \mathbb{A}$ such that:*
(C1) *the set Θ is equal to $\mathbb{A}_{|X}$,*
(C2) *the assignment θ_0 is equal to $\alpha_0|_X$,*
(C3) *$\mathbb{A}$ and α_0 satisfy the conditions stated in Lemma 7, and*
(C4) *$|\mathbb{A}(\theta)| \leq |\mathrm{Vars}(\varphi) \setminus X| + 1$ for every $\theta \in \Theta$.*

Proof. Note that the reverse direction follows immediately from Lemma 7, because the existence of the set of assignments $\mathbb{A}$ and the assignment α_0 satisfying condition (C3) imply the satisfiability of φ.

Towards showing the forward direction assume that φ is satisfiable. Because of Lemma 7 there is a set $\mathbb{A}$ of assignments of the variables in φ and an assignment $\alpha_0 \in \mathbb{A}$ that satisfy the conditions of Lemma 7. Let Θ be equal to $\mathbb{A}_{|X}$ and θ_0 be equal to $\alpha_0|_X$. Observe that setting Θ and θ_0 in this way already satisfies (C1) to (C3). We will show that there is a subset of $\mathbb{A}$ that still satisfies (C1)–(C3) and additionally (C4). Towards showing this consider any subset $\mathbb{A}'$ of $\mathbb{A}$ that satisfies the following three conditions: (1) $\alpha_0 \in \mathbb{A}'$, (2) for every $\theta \in \Theta$ it holds that $\mathbb{A}'(\theta) \neq \emptyset$, and (3) for every variable v of φ and every $b \in \{0,1\}$ it holds that there is an assignment $\alpha \in \mathbb{A}$ with $\alpha(v) = i$ if and only if there is an assignment $\alpha' \in \mathbb{A}'$ with $\alpha'(v) = i$. Note that conditions (1) and (2) ensure that $\mathbb{A}'$ satisfies (C1) and (C2) and condition (3) ensures (C3). Hence, any subset $\mathbb{A}'$ satisfying conditions (1)–(3) still satisfies (C1)–(C3). It remains to show how to obtain such a subset $\mathbb{A}'$ that additionally satisfies (C4). We define $\mathbb{A}'$ as follows. Let $\mathbb{A}'_0$ be a subset of $\mathbb{A}$ containing α_0 as well as one arbitrary assignment $\alpha \in \mathbb{A}(\theta)$ for every $\theta \in \Theta$. Note that $\mathbb{A}'_0$ already satisfies conditions (1) and (2) as well as condition (3) for every variable $v \in X$. Observe furthermore that if there is a variable v of φ such that condition (3) is violated by $\mathbb{A}'_0$ then it is sufficient to add at most one additional assignment to $\mathbb{A}'_0$ in order to satisfy condition (3) for v. Let $\mathbb{A}'$ be obtained from $\mathbb{A}'_0$ by adding (at most $|\mathrm{Vars}(\varphi) \setminus X|$) assignments in order to ensure condition (3) for every variable $v \in \mathrm{Vars}(\varphi) \setminus X$. Then $\mathbb{A}'$ satisfies the conditions of the lemma. ◀

We are now ready to show tractability for the evaluation of strong HORN-backdoor sets.

▶ **Theorem 9.** $\mathrm{Eval}^{\boxdot}(\textsc{horn})$ *is in* FPT.

Proof. Let $\varphi := \Psi \wedge \boxdot\Phi \in \mathrm{LTL}^{\boxdot}$ and let $X \subseteq \mathrm{Vars}(\varphi)$ be a strong HORN-backdoor of φ. The main idea of the algorithm is as follows: For every set Θ of assignments of the variables in X and every $\theta_0 \in \Theta$, we will construct a propositional HORN-formula F_{Θ,θ_0}, which is satisfiable if and only if there is a set $\mathbb{A}$ of assignments of the variables in $\mathrm{Vars}(\varphi)$ and an assignment $\alpha_0 \in \mathbb{A}$ satisfying the conditions of Lemma 8. It then follows from Lemma 8 that φ is satisfiable if and only if there is such a set Θ of assignments and an assignment $\theta_0 \in \Theta$ for which F_{Θ,θ_0} is satisfiable. Because there are at most $2^{2^{|X|}}$ such sets Θ and at most $2^{|X|}$ such assignments θ_0 and for each of these sets the formula F_{Θ,θ_0} is a HORN-formula, it follows that checking whether there are Θ and θ_0 such that the formula F_{Θ,θ_0} is satisfied (and therefore decide the satisfiability of φ) can be done in time $O(2^{2^{|X|}} \cdot 2^{|X|} \cdot |F_{\Theta,\theta_0}|)$. Since we will show below that the length of the formula F_{Θ,θ_0} can be bounded by an (exponential) function of $|X|$ times a polynomial in the input size, i.e., the length of the formula φ, this implies that $\mathrm{Eval}^{\boxdot}(\textsc{horn})$ is in FPT.

The remainder of the proof is devoted to the construction of the formula F_{Θ,θ_0} for a fixed set of assignments Θ and a fixed assignment $\theta_0 \in \Theta$ (and to show that it enforces the conditions of Lemma 8).

Let $R := \mathrm{Vars}(\varphi) \setminus X$ and $r := |R| + 1$. For a propositional formula F, a subset $V \subseteq \mathrm{Vars}(F)$, an integer i and a label s, we denote by $\mathbf{copy}(F, V, i, s)$ the propositional formula obtained from F after replacing each occurrence of a variable $v \in V$ with a novel variable v^i_s. We need the following auxiliary formulas. For every $\theta \in \Theta \setminus \theta_0$, let $F^{\theta}_{\Theta,\theta_0}$ be the formula:

$$\bigwedge_{1 \leq i \leq r} \mathbf{copy}(\mathbf{CNF}(\Phi[\mathbf{G}(\Theta, X, \theta)]), R, i, \theta).$$

Moreover, let $F^{\theta_0}_{\Theta,\theta_0}$ be the formula:

$$\mathbf{copy}(\Psi[\theta_0] \wedge \mathbf{CNF}(\Phi[\mathbf{G}(\Theta, X, \theta_0)]), R, 1, \theta_0) \wedge \bigwedge_{2 \leq i \leq r} \mathbf{copy}(\mathbf{CNF}(\Phi[\mathbf{G}(\Theta, X, \theta_0)]), R, i, \theta_0).$$

Observe that because X is a strong HORN-backdoor set (and the formula Ψ only consists of unit clauses), it holds that the formula $F^{\theta}_{\Theta,\theta_0}$ is HORN for every $\theta \in \Theta$.

We also need the propositional formula F_{const} that enforces the consistency between the propositional variables $\boxed{*}x$ and the variables in $\{ x^i_\theta \mid \theta \in \Theta \wedge 1 \leq i \leq r \}$ for every $x \in \mathrm{Vars}(\varphi) \setminus X$. The formula F_{const} consists of the following clauses: for every $\theta \in \Theta$, i with $1 \leq i \leq r$, and $v \in R$, the clause $\boxed{*}v \to v^i_\theta = \neg\boxed{*}v \vee v^i_\theta$ and for every $v \in R$ the clause

$$\neg\boxed{*}v \to \bigvee_{\theta \in \Theta \wedge 1 \leq i \leq r} \neg v^i_\theta = \boxed{*}v \vee \bigvee_{\theta \in \Theta \wedge 1 \leq i \leq r} \neg v^i_\theta.$$

Observe that F_{const} is a HORN formula.

Finally the formula F_{Θ,θ_0} is defined as: $\bigwedge_{\theta \in \Theta} F^{\theta}_{\Theta,\theta_0} \wedge F_{\mathrm{const}}$.
Note that F_{Θ,θ_0} is HORN and the length of F_{Θ,θ_0} is at most

$$\begin{aligned} |F_{\Theta,\theta_0}| &\leq \sum_{\theta \in \Theta} |F^{\theta}_{\Theta,\theta_0}| + |F_{\mathrm{const}}| \\ &\leq 2^{|X|}(|\mathrm{Vars}(\varphi) \setminus X| + 1)(|\Phi| + |\Psi|) + 2 \cdot 2^{|X|} \cdot (|\mathrm{Vars}(\varphi) \setminus X| + 1)^2 \end{aligned}$$

and consequently bounded by a function of $|X|$ times a polynomial in the input size. It is now relatively straightforward to verify that $F_{\Theta,\theta}$ is satisfiable if and only if there is a set $\mathbb{A}$ of assignments of the variables in $\mathrm{Vars}(\varphi)$ and an assignment $\alpha_0 \in \mathbb{A}$ satisfying the conditions of Lemma 8. Informally, for every $\theta \in \Theta$, each of the r copies of the formula $\mathbf{CNF}(\Phi[\mathbf{G}(\Theta, X, \theta)])$ represent one of the at most r assignments in $\mathbb{A}(\theta)$, the formula $F^{\theta_0}_{\Theta,\theta_0}$ ensures (among other things) that the assignment chosen for α_0 satisfies Ψ and the formula F_{const} ensures that the "global assignments" represented by the propositional variables $\boxed{*}x$ are consistent with the set of local assignments in $\mathbb{A}$ represented by the variables in $\{ x^i_\theta \mid \theta \in \Theta \wedge 1 \leq i \leq r \}$ for every $x \in \mathrm{Vars}(\varphi) \setminus X$. ◀

Surprisingly, the next result will show that KROM formulas turn out to be quite challenging. Backdoor set evaluation of this class of formulas is proved to be paraNP-complete which witnesses an intractability degree in the parameterized sense.

▶ **Theorem 10.** Eval$^{\boxed{*}}$(KROM) *is* paraNP*-complete (the* NP*-completeness already holds for backdoor sets of size two).*

Proof. The membership in paraNP follows because the satisfiability of $\mathrm{LTL}^{\boxed{*}}_{\mathrm{CNF}}$ can be decided in NP [2, Table 1].

We show paraNP-hardness of Eval$^{\boxed{*}}$(KROM) by giving a polynomial time reduction from the NP-hard problem 3COL to Eval$^{\boxed{*}}$(KROM) for backdoors of size two. In 3COL one asks

	b_1	b_2	v_1	v_2	v_3	$e^{b_1 b_2}_{v_1 v_2}$	$e^{\bar{b}_1 b_2}_{v_1 v_2}$	$e^{b_1 \bar{b}_2}_{v_1 v_2}$	$e^{b_1 b_2}_{v_1 v_3}$	$e^{\bar{b}_1 b_2}_{v_1 v_3}$	$e^{b_1 \bar{b}_2}_{v_1 v_3}$	$e^{b_1 b_2}_{v_2 v_3}$	$e^{\bar{b}_1 b_2}_{v_2 v_3}$	$e^{b_1 \bar{b}_2}_{v_2 v_3}$
1	0	0	0	1	1	1	0	–	1	–	0	–	1	0
2	1	0	1	0	1	1	0	–	1	–	0	–	1	0
3	0	1	1	1	0	1	0	–	1	–	0	–	1	0

Figure 1 Given a graph $G = (\{v_1, v_2, v_3\}, \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}\})$ together with a 3-Coloring $f(v_i) = i$ for $1 \leq i \leq 3$, leads to the depicted temporal interpretation $\mathfrak{M}$ satisfying $\mathfrak{M} \models \phi$ given as a table. Each row of the table corresponds to a world as indicated by the first column of the table. Each column represents the assignments of a variable as indicated in the first row. A "–" indicates that the assignment is not fixed, i.e., the assignment does not influence whether $\mathfrak{M} \models \phi$.

whether a given input graph $G = (V, E)$ has a coloring $f \colon V(G) \to \{1, 2, 3\}$ of its vertices with at most three colors such that $f(v) \neq f(u)$ for every edge $\{u, v\}$ of G. Given such a graph $G = (V, E)$, we will construct an $\mathrm{LTL}^{\boxed{*}}_{\mathrm{CNF}}$ formula $\phi := \Psi \wedge \boxed{*}\Phi$, which has a strong KROM-backdoor B of size two, such that the graph G has a 3-coloring if and only if ϕ is satisfiable.

For the remainder we will assume that there exists an arbitrary but fixed ordering of the vertices $V(G) = \{v_1, \ldots, v_n\}$. Further for the construction we assume w.l.o.g. that any undirected edge $e = \{v_i, v_j\} \in E$ follows this ordering, i.e., $i < j$. The formula ϕ contains the following variables:

(V1) The variables b_1 and b_2. These variables make up the backdoor set B, i.e., $B := \{b_1, b_2\}$.
(V2) For every i with $1 \leq i \leq n$, the variable v_i.
(V3) For every $e = \{v_i, v_j\} \in E(G)$ with $1 \leq i, j \leq n$ the variables $e^{b_1 b_2}_{v_i v_j}$, $e^{\bar{b}_1 b_2}_{v_i v_j}$, and $e^{b_1 \bar{b}_2}_{v_i v_j}$.

We set Ψ to be the empty formula and the formula Φ contains the following clauses:

(C1) For every i with $1 \leq i \leq n$, the clause $\neg \boxed{*} v_i$. Informally, this clause ensures that v_i has to be false at least at one world, which will later be used to assign a color to the vertex v_i of G. Observe that the clause is KROM.
(C2) For every $e = \{v_i, v_j\} \in E(G)$ with $1 \leq i, j \leq n$, the clauses $v_i \vee \boxed{*} e^{b_1 b_2}_{v_i v_j} \vee b_1 \vee b_2$, $v_i \vee \boxed{*} e^{\bar{b}_1 b_2}_{v_i v_j} \vee \neg b_1 \vee b_2$, and $v_i \vee \boxed{*} e^{b_1 \bar{b}_2}_{v_i v_j} \vee b_1 \vee \neg b_2$ as well as the clauses $v_j \vee \neg \boxed{*} e^{b_1 b_2}_{v_i v_j} \vee b_1 \vee b_2$, $v_j \vee \neg \boxed{*} e^{\bar{b}_1 b_2}_{v_i v_j} \vee \neg b_1 \vee b_2$, and $v_j \vee \neg \boxed{*} e^{b_1 \bar{b}_2}_{v_i v_j} \vee b_1 \vee \neg b_2$. Observe that all of these clauses are KROM after deleting the variables in B.
(C3) The clause $\neg b_1 \vee \neg b_2$. Informally, this clause excludes the color represented by setting b_1 and b_2 to true. Observe that the clause is KROM.

It follows from the definition of ϕ that $\phi[\theta] \in \mathrm{LTL}^{\boxed{*}}_{\mathrm{KROM}}$ for every assignment θ of the variables in B. Hence, B is a strong KROM-backdoor of size two of ϕ as required. Moreover, since ϕ can be constructed in polynomial time, it only remains to show that G has a 3-Coloring if and only if ϕ is satisfiable.

Towards showing the forward direction assume that G has a 3-Coloring and let $f \colon V(G) \to \{1, 2, 3\}$ be such a 3-Coloring for G. We will show that ϕ is satisfiable by constructing a temporal interpretation $\mathfrak{M}$ such that $\mathfrak{M} \models \phi$. $\mathfrak{M}$ is defined as follows:

- For every i with $1 \leq i \leq n$, we set $\mathfrak{M}(v_i) = \mathbb{Z} \setminus \{f(v_i)\}$.
- We set $\mathfrak{M}(b_1) = \{2\}$ and $\mathfrak{M}(b_2) = \{3\}$.
- For every $e = \{v_i, v_j\} \in E(G)$:
  - if $f(v_i) = 1$ set $\mathfrak{M}(e^{b_1 b_2}_{v_i v_j}) = \mathbb{Z}$, else set $\mathfrak{M}(e^{b_1 b_2}_{v_i v_j}) = \emptyset$.
  - if $f(v_i) = 2$ set $\mathfrak{M}(e^{\bar{b}_1 b_2}_{v_i v_j}) = \mathbb{Z}$, else set $\mathfrak{M}(e^{\bar{b}_1 b_2}_{v_i v_j}) = \emptyset$.
  - if $f(v_i) = 3$ set $\mathfrak{M}(e^{b_1 \bar{b}_2}_{v_i v_j}) = \mathbb{Z}$, else set $\mathfrak{M}(e^{b_1 \bar{b}_2}_{v_i v_j}) = \emptyset$.

An example for such a temporal interpretation resulting for a simple graph is illustrated in Figure 1. Towards showing that $\mathfrak{M} \models \phi$, we consider the different types of clauses given in (C1)–(C3).

- The clauses in (C1) hold because $\mathfrak{M}, f(v_i) \not\models v_i$ for every i with $1 \leq i \leq n$.
- For every $e = \{v_i, v_j\} \in E(G)$, we have to show that the clauses given in (C2) are satisfied for every world. Because f is a 3-Coloring of G, we obtain that $f(v_i) \neq f(v_j)$. W.l.o.g. we assume in the following that $f(v_i) = 1$ and $f(v_j) = 2$. We first consider the clauses given in (C2) containing v_i. Because $\mathfrak{M}(v_i) = \mathbb{Z} \setminus \{1\}$, it only remains to consider the world 1. In this world b_1 and b_2 are false. It follows that all clauses containing either $\neg b_1$ or $\neg b_2$ are satisfied in this world. Hence, it only remains to consider clauses of the form $v_i \vee \boxdot \circledast e^{b_1 b_2}_{v_i v_j} \vee b_1 \vee b_2$. But these are satisfied because $f(v_i) = 1$ implies that $\mathfrak{M}(e^{b_1 b_2}_{v_i v_j}) = \mathbb{Z}$. Consider now the clauses given in (C2) that contain v_j. Using the same argumentation as used above for v_i, we obtain that we only need to consider world 2 and moreover we only need to consider clauses of the form $v_j \vee \neg \boxdot \circledast e^{\bar{b}_1 b_2}_{v_i v_j} \vee \neg b_1 \vee b_2$. Because $f(v_i) = 1$, we obtain that $\mathfrak{M}(e^{\bar{b}_1 b_2}_{v_i v_j}) = \emptyset$, which implies that these clauses are also satisfied.
- The clause $\neg b_1 \vee \neg b_2$ is trivially satisfied, because there is no world in which b_1 and b_2 hold simultaneously.

Towards showing the reverse direction assume that ϕ is satisfiable and let $\mathfrak{M}$ be a temporal interpretation witnessing this. First note that because of the clauses added by C1, it holds that $\mathfrak{M}(v_i) \neq \mathbb{Z}$ for every i with $1 \leq i \leq n$. Let $w \colon V(G) \to \mathbb{Z}$ be defined such that for every i with $1 \leq i \leq n$, $w(v_i)$ is an arbitrary world in $\mathbb{Z} \setminus \mathfrak{M}(v_i)$. We define $f \colon V(G) \to \{1, 2, 3\}$ by setting:

- $f(v_i) = 1$ if $\mathfrak{M}, w(v_i) \not\models b_1 \vee b_2$,
- $f(v_i) = 2$ if $\mathfrak{M}, w(v_i) \not\models \neg b_1 \vee b_2$, and
- $f(v_i) = 3$ if $\mathfrak{M}, w(v_i) \not\models b_1 \vee \neg b_2$.

Note that because of the clause added by (C3), f assigns exactly one color to every vertex v_i of G. We claim that f is a 3-Coloring of G. To show this it suffices to show that for every $e = \{v_i, v_j\} \in E(G)$, it holds that $f(v_i) \neq f(v_j)$. Assume for a contradiction that this is not the case, i.e., there is an edge $e = \{v_i, v_j\} \in E(G)$ such that $f(v_i) = f(v_j)$. W.l.o.g. assume furthermore that $f(v_i) = f(v_j) = 1$. Consider the clause $v_i \vee \boxdot \circledast e^{b_1 b_2}_{v_i v_j} \vee b_1 \vee b_2$ (which was added by C2). Then, because of the definition of w and f, we obtain that $\mathfrak{M}, w(v_i) \not\models v_i \vee b_1 \vee b_2$. It follows that $\mathfrak{M}, w(v_i) \models \boxdot \circledast e^{b_1 b_2}_{v_i v_j}$. Consider now the clause $v_j \vee \neg \boxdot \circledast e^{b_1 b_2}_{v_i v_j} \vee b_1 \vee b_2$ (which was added by C2). Then, again because of the choice of w and f, we obtain that $\mathfrak{M}, w(v_j) \not\models v_j \vee b_1 \vee b_2$. Hence, $\mathfrak{M}, w(v_j) \models \neg \boxdot \circledast e^{b_1 b_2}_{v_i v_j}$ contradicting $\mathfrak{M}, w(v_i) \models \boxdot \circledast e^{b_1 b_2}_{v_i v_j}$. This completes the proof of the theorem. ◀

5.2 Globally in the past and globally in the future

Now we turn to a more flexible fragment where we can talk about the past as well as about the future and show it is possible to encode NP-complete problems into the HORN-fragment yielding a paraNP lower bound.

▶ **Theorem 11.** $\mathrm{Eval}^{\Box_F, \Box_P}(\mathrm{HORN})$ *is* paraNP*-complete (the* NP*-completeness already holds for backdoor sets of size four).*

Proof. The membership in paraNP follows because the satisfiability of $\mathrm{LTL}^{\Box_F, \Box_P}_{\mathrm{CNF}}$ can be decided in NP [2, Table 1].

We show paraNP-hardness of $\mathrm{Eval}^{\Box_F, \Box_P}(\mathrm{HORN})$ by describing a polynomial time reduction again from 3COL to $\mathrm{Eval}^{\Box_F, \Box_P}(\mathrm{HORN})$ for backdoors of size four. Recall that in 3COL one

asks whether a given input graph $G = (V, E)$ has a coloring $f\colon V(G) \to \{1, 2, 3\}$ of its vertices with at most three colors such that $f(v) \neq f(u)$ for every edge $\{u, v\}$ of G. Given such a graph $G = (V, E)$, we will construct an $\mathrm{LTL}_{\mathrm{CNF}}^{\Box_F, \Box_P}$ formula $\phi := \Psi \wedge \boxtimes \Phi$, which has a strong HORN-backdoor B of size four, such that the graph G has a 3-coloring if and only if ϕ is satisfiable.

For the remainder we will assume that $V(G) = \{v_1, \ldots, v_n\}$ and $E(G) = \{e_1, \ldots, e_m\}$. The formula ϕ contains the following variables:

(V1) The variables c_1, c_2, c_3, p'_n . These variables make up the backdoor set B, i.e., $B := \{c_1, c_2, c_3, p'_n\}$.

(V2) The variable s, which indicates the starting world.

(V3) For every i with $1 \leq i \leq n$, three variables v_i^1, v_i^2, v_i^3.

(V4) For every i with $1 \leq i \leq n$ the variable p_i.

We set Ψ to be the formula s and the formula Φ contains the following clauses:

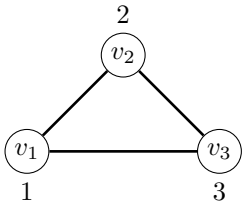
(C1) The clauses $c_1 \vee c_2 \vee c_3$, $\neg c_1 \vee \neg c_2 \vee \neg c_3$, $c_1 \vee \neg c_2 \vee \neg c_3$, $\neg c_1 \vee \neg c_2 \vee c_3$, and $\neg c_1 \vee c_2 \vee \neg c_3$. Informally, these clauses ensure that in every world it holds that exactly one of the variables c_1, c_2, c_3 is true. Note that $c_1 \vee c_2 \vee c_3$ is not HORN, however, all of its variables are contained in the backdoor set B.

(C2) For every i and c with $1 \leq i \leq n$ and $1 \leq c \leq 3$, the clauses $v_i^c \to \Box_F v_i^c = \neg v_i^c \vee \Box_F v_i^c$ and $v_i^c \to \Box_P v_i^c = \neg v_i^c \vee \Box_P v_i^c$. Informally, these clauses ensure that the variable v_i^c either holds in every world or in no world for every i and c as above. Observe that both of these clauses are HORN.

(C3) Informally, the following set of clauses ensures together that for every i with $1 \leq i \leq n$, it holds that p_i is true in every world apart from the i-th world (where p_i is false). Here, the first world is assumed to be the starting world.

- **(C3-1)** The clauses $s \to \neg p_1 = \neg s \vee \neg p_1$, $s \to \Box_F p_1 = \neg s \vee \Box_F p_1$, and $s \to \Box_P p_1 = \neg s \vee \Box_P p_1$. Informally, these ensure that p_1 is only false in the starting world (and otherwise true).
- **(C3-2)** The clause $p_i \wedge \Box_F p_i \to \Box_F p_{i+1} = \neg p_i \vee \neg \Box_F p_i \vee \Box_F p_{i+1}$ for every i with $1 \leq i < n$. Informally, these clauses (together with the clauses from C3-1) ensure that for every i with $2 \leq i \leq n$, it holds that p_i is true in every world after the i-th world.
- **(C3-3)** The clause $\neg p_i \to \neg \Box_F p_{i+1} = p_i \vee \neg \Box_F p_{i+1}$ for every i with $1 \leq i < n$. Informally, these clauses (together with the clauses from C3-1 and C3-2) ensure that for every i with $2 \leq i \leq n$, it holds that p_i is false at the i-th world. Observe that the clauses from C3-1 to C3-3 already ensure that $\neg p_i \wedge \Box_F p_i$ holds if and only if we are at the i-th world of the model for every i with $1 \leq i \leq n$.
- **(C3-4)** The clauses $\neg p_n \wedge \Box_F p_n \to p'_n = p_n \vee \neg \Box_F p_n \vee p'_n$ and $\neg p_n \wedge \Box_F p_n \leftarrow p'_n = \neg p_n \wedge \Box_F p_n \vee \neg p'_n = (\neg p_n \vee \neg p'_n) \wedge (\Box_F p_n \vee \neg p'_n)$. Informally, these clauses (together with the clauses from C3-1 to C3-3) ensure that p'_n only holds in the n-th world of the model. Observe that all these clauses are HORN after removing the backdoor set variable p'_n.
- **(C3-5)** The clause $p'_n \to \Box_P p_n = \neg p'_n \vee \Box_P p_n$. Informally, this clause (together with the clauses from C3-1 to C3-4) ensures that p_n is only false in the n-th world of the model.
- **(C3-6)** The clause $p_i \wedge \Box_P p_i \to \Box_P p_{i-1} = \neg p_i \vee \neg \Box_P p_i \vee \Box_P p_{i-1}$ for every i with $2 \leq i \leq n$. Informally, these clauses (together with the clauses from C3-1 to C3-5) ensure that p_i is true before the i-th world for every i with $2 \leq i < n$.

Observe that all of the above clauses are HORN or become HORN after removing all variables from B. Note furthermore that all the above clauses ensure that $\Box_P p_i \wedge \Box_F p_i$ holds if and only if we are at the i-th world of the model for every i with $1 \leq i \leq n$.



	s	c_1	c_2	c_3	p'_n	v_1^1	v_1^2	v_1^3	v_2^1	v_2^2	v_2^3	v_3^1	v_3^2	v_3^3	p_1	p_2	p_3
< 1	0	–	–	–	0	1	0	0	0	1	0	0	0	1	1	1	1
1	1	1	0	0	0	1	0	0	0	1	0	0	0	1	0	1	1
2	0	0	1	0	0	1	0	0	0	1	0	0	0	1	1	0	1
3	0	0	0	1	1	1	0	0	0	1	0	0	0	1	1	1	0
> 3	0	–	–	–	0	1	0	0	0	1	0	0	0	1	1	1	1

Figure 2 Left: A graph G with vertices v_1, v_2, and v_3 together with a 3-Coloring given by the numbers above and below respectively of every vertex. Right: A temporal interpretation $\mathfrak{M}$ that corresponds to the given 3-Coloring of G and satisfies $\mathfrak{M} \models \phi$ given as a table. Each row of the table corresponds to a world (or a set of worlds) as indicated by the first column of the table. Each column represents the assignments of a variable as indicated in the first row. A "–" indicates that the assignment is not fixed, i.e., the assignment does not influence whether $\mathfrak{M} \models \phi$.

(C4) For every i and j with $1 \leq i \leq n$ and $1 \leq j \leq 3$ the clauses $\Box_F p_i \wedge \Box_P p_i \wedge v_i^j \to c_j = \neg\Box_F p_i \vee \neg\Box_P p_i \vee \neg v_i^j \vee c_j$ and $\Box_F p_i \wedge \Box_P p_i \wedge c_j \to v_i^j = \neg\Box_F p_i \vee \neg\Box_P p_i \vee \neg c_j \vee v_i^j$. Informally, these clauses ensure that in the i-th world for every $1 \leq i \leq n$, the variables c_1, c_2, c_3 are a copy of the variables v_i^1, v_i^2, v_i^3. Observe that all of these clauses are HORN.

(C5) For every edge $e = \{v_i, v_j\} \in E(G)$ and every c with $1 \leq c \leq 3$, the clause $\neg v_i^c \vee \neg v_j^c$. Informally, these clauses ensure that the 3-partition (of the vertices of G) given by the (global) values of the variables $v_1^1, v_1^2, v_1^3, \ldots, v_n^1, v_n^2, v_n^3$ is a valid 3-Coloring for G. Observe that all of these clauses are HORN.

It follows from the definition of ϕ that $\phi[\theta] \in \mathrm{LTL}_{\mathrm{HORN}}^{\Box_F,\Box_P}$ for every assignment θ of the variables in B. Hence, B is a strong HORN-backdoor of size four of ϕ as required. Moreover, since ϕ can be constructed in polynomial time, it only remains to show that G has a 3-Coloring if and only if ϕ is satisfiable.

Towards showing the forward direction assume that G has a 3-Coloring and let $f\colon V(G) \to \{1,2,3\}$ be such a 3-Coloring for G. We will show that ϕ is satisfiable by constructing a temporal interpretation $\mathfrak{M}$ such that $\mathfrak{M} \models \phi$. $\mathfrak{M}$ is defined as follows:

- For every j with $1 \leq j \leq 3$, we set $\mathfrak{M}(c_j) = \{\, i \mid f(v_i) = j \,\}$.
- We set $\mathfrak{M}(p'_n) = \{n\}$.
- For every i and c with $1 \leq i \leq n$ and $1 \leq c \leq 3$, we set $\mathfrak{M}(v_i^c) = \mathbb{Z}$ if $c = f(v_i)$ and otherwise we set $\mathfrak{M}(v_i^c) = \emptyset$.
- For every i with $1 \leq i \leq n$, we set $\mathfrak{M}(p_i) = \mathbb{Z} \setminus \{i\}$.

An example for such a temporal interpretation resulting for a simple graph is illustrated in Figure 2. It is straightforward (but a little tedious) to verify that $\mathfrak{M} \models \phi$ by considering all the clauses of ϕ.

Towards showing the reverse direction assume that ϕ is satisfiable and let $\mathfrak{M}$ be a temporal interpretation witnessing this. We will start by showing the following series of claims for $\mathfrak{M}$.

(M1) For every $a \in \mathbb{Z}$ exactly one of $\mathfrak{M}, a \models c_1$, $\mathfrak{M}, a \models c_2$, and $\mathfrak{M}, a \models c_3$ holds.

(M2) For every i, c, a, and a' with $1 \leq i \leq n$, $1 \leq c \leq 3$, and $a, a' \in \mathbb{Z}$, it holds that $\mathfrak{M}, a \models v_i^c$ if and only if $\mathfrak{M}, a' \models v_i^c$.

(M3) For every i with $1 \leq i \leq n$ and every $a \in \mathbb{Z}$, it holds that $\mathfrak{M}, a \models p_i$ if and only if $a \neq i$.

(M4) For every i and j with $1 \leq i \leq n$ and $1 \leq j \leq 3$, it holds that $\mathfrak{M}, i \models c_j$ if and only if $\mathfrak{M}, i \models v_i^j$.

(M1) holds because of the clauses added by (C1). Towards showing (M2) consider the clauses added by (C2) and assume for a contradiction that there are i, c, a, and a' as in the statement of (M2) such that w.l.o.g. $\mathfrak{M}, a \models v_i^c$ but $\mathfrak{M}, a' \not\models v_i^c$. Then, $a \neq a'$. If $a < a'$, then we obtain a contradiction because of the clause $v_i^c \rightarrow \Box_F v_i^c$ and if on the other hand $a' < a$, we obtain a contradiction to the clause $v_i^c \rightarrow \Box_P v_i^c$. This completes the proof of (M2). We will show (M3) with the help of the following series of claims.

(M3-1) For every $a \in \mathbb{Z}$ it holds that $\mathfrak{M}, a \models p_1$ if and only if $a \neq 1$ (here we assume that 1 is the starting world).

(M3-2) For every i and a with $1 \leq i \leq n$, $a \in \mathbb{Z}$, and $a > i$, it holds that $\mathfrak{M}, a \models p_i$.

(M3-3) For every i with $1 \leq i \leq n$, it holds that $\mathfrak{M}, i \not\models p_i$.

(M3-4) For every $a \in \mathbb{Z}$, it holds that $\mathfrak{M}, a \models p'_n$ if and only if $a = n$.

(M3-5) For every $a \in \mathbb{Z}$, it holds that $\mathfrak{M}, a \not\models p_n$ if and only if $a = n$.

Because of the clause $s \rightarrow \neg p_1$ (added by C3-1) and the fact that $s \in \Psi$, we obtain that $\mathfrak{M}, 1 \not\models p_1$. Moreover, because of the clauses $s \rightarrow \Box_F p_1$ and $s \rightarrow \Box_P p_1$, we obtain that $\mathfrak{M}, a \models p_1$ for every $a \neq 1$. This completes the proof for (M3-1).

We show (M3-2) via induction on i. The claim clearly holds for $i = 1$ because of (M3-1). Now assume that the claim holds for p_{i-1} and we want to show it for p_i. Because of the induction hypothesis, we obtain that $\mathfrak{M}, i \models p_{i-1} \wedge \Box_F p_{i-1}$. Moreover, because ϕ contains the clause $p_{i-1} \wedge \Box_F p_{i-1} \rightarrow \Box_F p_i$ (which was added by (C3-2)), we obtain that $\mathfrak{M}, i \models \Box_F p_i$. This completes the proof of (M3-2).

We show (M3-3) via induction on i. The claim clearly holds for $i = 1$ because of (M3-1). Now assume that the claim holds for p_{i-1} and we want to show it for p_i. Because of the induction hypothesis, we obtain that $\mathfrak{M}, (i-1) \not\models p_{i-1}$. Furthermore, because of (M3-2), we know that $\mathfrak{M}, i \models \Box_F p_i$. Since ϕ contains the clause $\neg p_{i-1} \rightarrow \neg \Box_F p_i$ (which was added by (C3-3)), we obtain $\mathfrak{M}, (i-1) \models \neg \Box_F p_i$, which because $\mathfrak{M}, i \models \Box_F p_i$ can only hold if $\mathfrak{M}, i \not\models p_i$. This completes the proof of (M3-3).

Towards showing (M3-4), first note that because of (M3-2) and (M3-3), we have that $\mathfrak{M}, a \models \neg p_n \wedge \Box_F p_n$ if and only if $a = n$. Then, because of the clauses (added by C3-4) ensuring that $\neg p_n \wedge \Box_F p_n \leftrightarrow p'_n$, the same applies to p'_n (instead of $\neg p_n \wedge \Box_F p_n$). This completes the proof of (M3-4).

It follows from (M3-2) and (M3-3) that (M3-5) holds for every $a \in \mathbb{Z}$ with $a \geq n$. Moreover, because of (M3-4), we have that $\mathfrak{M}, n \models p'_i$. Because of the clause $p'_n \rightarrow \Box_P p_n$ (which was added by (C3-5)), we obtain $\mathfrak{M}, a \models p_n$ for every $a < n$. This completes the proof of (M3-5).

We are now ready to prove (M3). It follows from (M3-2) and (M3-3) that (M3) holds for every i and a with $a \geq i$. Furthermore, we obtain from (M3-5) that (M3) already holds if $i = n$. We complete the proof of (M3) via an induction on i starting from $i = n$. Because of the induction hypothesis, we obtain that $\mathfrak{M}, i+1 \models p_{i+1} \wedge \Box_P p_{i+1}$. Hence, because of the clause $p_{i+1} \wedge \Box_P p_{i+1} \rightarrow \Box_P p_i$ (added by (C3-6)), we obtain that $\mathfrak{M}, i+1 \models \Box_P p_i$, which completes the proof of (M3).

Towards showing (M4) first note that it follows from (M3) that $\mathfrak{M}, i \models \Box_F p_i \wedge \Box_P p_i$. Now suppose that there are i and j such that either $\mathfrak{M}, i \models c_j$ but $\mathfrak{M}, i \not\models v_i^j$ or $\mathfrak{M}, i \not\models c_j$ but $\mathfrak{M}, i \models v_i^j$. In the former case, consider the clause $\Box_F p_i \wedge \Box_P p_i \wedge c_j \rightarrow v_i^j$ (which was added by (C4)). Since $\mathfrak{M}, i \models \Box_F p_i \wedge \Box_P p_i$, we obtain that $\mathfrak{M}, i \models v_i^j$; a contradiction. In the later case, consider the clause $\Box_F p_i \wedge \Box_P p_i \wedge v_i^j \rightarrow c_j$ (which was added by (C4)). Since $\mathfrak{M}, i \models \Box_F p_i \wedge \Box_P p_i$, we obtain that $\mathfrak{M}, i \models c_j$; again a contradiction. This completes the proof of the claims (M1)–(M4).

It follows from (M1) and (M4) that for every i and a with $1 \leq i \leq n$ and $a \in \mathbb{Z}$ there is exactly one c with $1 \leq c \leq 3$, such that $\mathfrak{M}, a \models v_i^c$. Moreover, because of (M2) the choice of c is independent of a. Hence, the coloring f that assigns the unique color c to every vertex v_i such that $\mathfrak{M}, a \models v_i^c$ forms a partition of the vertex set of G. We claim that f is also a valid 3-Coloring of G. Assume not, then there is an edge $\{v_i, v_j\} \in E(G)$ such that $c = f(v_i) = f(v_j)$. Consider the clause $\neg v_i^c \vee \neg v_j^c$ (which was added by C5). Because of the definition of f, we obtain that $\mathfrak{M}, a \not\models \neg v_i^c \vee \neg v_j^c$ for every $a \in \mathbb{Z}$, a contradiction to our assumption that $\mathfrak{M} \models \phi$. ◀

▶ **Corollary 12.** *Let* $O \in \{\Box_{\mathrm{F}}, \Box_{\mathrm{P}}\}$ *then* $\mathrm{Eval}^O(\textsc{krom})$ *is* paraNP*-complete (the* NP*-completeness already holds for backdoor sets of size zero).*

Proof. Satisfiability of $\mathrm{LTL}^O_{\textsc{krom}}$ is NP-hard [2, Theorem 5]. ◀

6 Conclusion and discussion

We lift the well-known concept of backdoor sets from propositional logic up to the clausal fragment of linear temporal logic LTL. From the investigated cases we exhibit a parameterized complexity dichotomy for the problem of backdoor set evaluation. The evaluation parameterized by the size of the backdoor into KROM formulas becomes in all cases paraNP-complete and thus is unlikely to be solvable in FPT whereas the case of backdoor evaluation into the fragment HORN behaves differently. While allowing only $\boxtimes$ makes the problem fixed-parameter tractable, allowing both, $\Box_{\mathrm{F}}$ and $\Box_{\mathrm{P}}$, makes it paraNP-complete. The last open case, i.e., the restriction to either $\Box_{\mathrm{F}}$ or $\Box_{\mathrm{P}}$ is open for further research and might yield an FPT result. We want to note here that all of our results still hold if LTL is evaluated over the natural numbers instead of the integers.

Satisfiability of $\mathrm{LTL}^{\boxtimes}_{\mathrm{CNF}}$ is NP-complete, for HORN/KROM it is in P/NL [2]. With the help of our backdoor notion, we achieved for a HORN-backdoor an FPT membership. However, for KROM this surprisingly was not possible (paraNP-c., Theorem 10). For the "full global" fragment only for HORN satisfiability is in P and for KROM it is NP-complete [2]. Here in both cases, our notion of backdoors was not fruitful. This is, however, natural since applying the backdoor approach to a novel problem is never a simple nor straightforward task. We see our work as a first attempt to come up with such a notion for LTL, and, given the notorious difficulty of the LTL-satisfiability problem, we believe our tractability result for LTL formulas restricted to the always operator that are almost HORN is an encouraging result that justifies further investigation of this approach. As mentioned earlier, LTL restricted to the always operator, is already pretty interesting, since it allows one to express "Safety" properties of a system. Moreover, our intractability results for the remaining fragments of LTL indicate that a different notion of "closeness" is required to obtain tractability results for these fragments.

References

1 F. N. Abu-Khzam. A kernelization algorithm for d-hitting set. *Journal of Computer and System Sciences*, 76(7):524–531, 2010.

2 A. Artale, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev. The complexity of clausal fragments of LTL. In *Proc. 19th LPAR*, volume 8312 of *LNCS*, 2013.

3 M. Bauland, T. Schneider, H. Schnoor, I. Schnoor, and H. Vollmer. The complexity of generalized satisfiability for linear temporal logic. *LMCS*, 5(1), 2009.

4 C.-C. Chen and I.-P. Lin. The computational complexity of satisfiability of temporal Horn formulas in propositional linear-time temporal logic. *IPL*, 45(3):131–136, 1993.

5 J. Chen, B. Chor, M. Fellows, X. Huang, D. Juedes, I. Kanji, and G. Xia. Tight Lower Bounds for Certain Parameterized NP-Hard Problems. *Information and Computation*, 201(2):216–231, 2005.

6 J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40–42):3736–3756, 2010.

7 E. M. Clarke and E. A. Emerson. Design and synthesis of synchronisation skeletons using branching time temporal logic. In *Logic of Programs*, volume 131 of *LNCS*, pages 52–71. Springer Verlag, 1981.

8 S. Demri and P. Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Information and Computation*, 174(1):84–103, 2002.

9 B. N. Dilkina, C. P. Gomes, and A. Sabharwal. Tradeoffs in the complexity of backdoor detection. In *Proc. 13th CP*, volume 4741 of *Lecture Notes in Computer Science*, pages 256–270. Springer Verlag, 2007.

10 B. N. Dilkina, C. P. Gomes, and A. Sabharwal. Backdoors in the context of learning. In *Proc. 12th SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 73–79. Springer Verlag, 2009.

11 C. Dixon, M. Fisher, and B. Konev. Tractable temporal reasoning. In *Proc. of IJCAI*, pages 318–323, 2007.

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

13 W. Dvorák, S. Ordyniak, and S. Szeider. Augmenting tractable fragments of abstract argumentation. *Artificial Intelligence*, 186:157–173, 2012.

14 E. Allen Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30(1):1–24, 1985.

15 M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Transactions on Computational Logic*, 2(1):12–56, 2001.

16 Michael Fisher. A normal form for temporal logic and its application in theorem-proving and execution. *Journal of Logic and Computation*, 7:429–456, 1997.

17 D. M. Gabbay, I. Hodkinsion, and M. Reynolds. *Temporal logic: mathematical foundations and computational aspects*, volume 1. Oxford University Press, Inc. New York, USA, 1994.

18 S. Gaspers and S. Szeider. Backdoors to satisfaction. In *The Multivariate Algorithmic Revolution and Beyond – Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *LNCS*, pages 287–317. Springer, 2012.

19 S. Gaspers and S. Szeider. Strong backdoors to bounded treewidth SAT. In *Proc. 54th FOCS*, pages 489–498. IEEE Computer Society, 2013.

20 S. Kottler, M. Kaufmann, and C. Sinz. A new bound for an NP-hard subclass of 3-SAT using backdoors. In *Proc. 11th SAT*, Lecture Notes in Computer Science, pages 161–167. Springer Verlag, 2008.

21 S. Kripke. Semantical considerations on modal logic. In *Acta philosophica Fennica*, volume 16, pages 84–94, 1963.

22 M. Kronegger, S. Ordyniak, and A. Pfandler. Backdoors to planning. In *Proc. 28th AAAI*, pages 2300–2307. AAAI Press, 2014.

23 M. Kronegger, S. Ordyniak, and A. Pfandler. Variable-deletion backdoors to planning. In *Proc. 29th AAAI*, pages 2300–2307. AAAI Press, 2014.

24 R. LeBras, R. Bernstein, C. P. Gomes, B. Selman, and R. Bruce van Dover. Crowdsourcing backdoor identification for combinatorial optimization. In *Proc. 23rd IJCAI*. AAAI, 2013.

25 M. Lück and A. Meier. LTL Fragments are Hard for Standard Parameterisations. In *Proc. of TIME*, pages 59–68, 2015. `doi:10.1109/TIME.2015.9`.

26 N. Markey. Past is for free: On the complexity of verifying linear temporal properties with past. *Acta Informatica*, 40(6-7):431–458, 2004.

27 H. Ono and A. Nakamura. On the size of refutation Kripke models for some linear modal and tense logics. *Studia Logica*, 39(325–333), 1980.

28 S. Ordyniak, D. Paulusma, and S. Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. *Theoretical Computer Science*, 481:85–99, 2013.

29 A. Pfandler, S. Rümmele, and S. Szeider. Backdoors to abduction. In *Proc. 23rd IJCAI*, 2013.

30 A. Pnueli. The temporal logic of programs. In *Proc. of FOCS*, pages 46–57. IEEE Comp. Soc. Press, 1977.

31 Y. Ruan, H. A. Kautz, and E. Horvitz. The backdoor key: A path to understanding problem hardness. In *Proc. 19th IAAI*, pages 124–130. AAAI Press, 2004.

32 M. Samer and S. Szeider. Backdoor sets of quantified Boolean formulas. *Journal of Automated Reasoning*, 42(1):77–97, 2009.

33 M. Samer and S. 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.

34 A. Sistla and E. Clarke. The complexity of propositional linear temporal logics. In *Proc. of STOC*, pages 159–168. ACM, 1982.

35 S. Szeider. On fixed-parameter tractable parameterizations of SAT. In *Proc. of SAT*, pages 188–202, 2003. doi:10.1007/978-3-540-24605-3_15.

36 R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proc. 18th IJCAI*, pages 1173–1178. Morgan Kaufmann, 2003.

A Proof of Theorem 6

Let $\mathcal{O} \subseteq \{\boxtimes, \Box_P, \Box_F\}$. We will reduce Detect$^{\mathcal{O}}$(KROM) to the 3-HittingSet problem, which is well-known to be fixed-parameter tractable (parameterized by the solution size) [1]. Recall that given a universe U, a family $\mathcal{F}$ of subsets of U of size at most three, and an integer k, 3-HittingSet asks whether there is a subset $S \subseteq U$ of size at most k (which is called a hitting set of $\mathcal{F}$) such that $S \cap F \neq \emptyset$ for every $F \in \mathcal{F}$. Given an LTL$^{\mathcal{O}}$ formula $\phi := \Psi \wedge \boxtimes \Phi$, we will construct a family $\mathcal{F}$ of subsets (of size at most three) of a universe U such that ϕ has a strong KROM-backdoor of size at most k if and only if $\mathcal{F}$ has a hitting set of size at most k. The universe U is equal to Vars(ϕ) and $\mathcal{F}$ contains the set Vars(C) for every set C of exactly three literals contained in some clause of Φ. We claim that a set $X \subseteq$ Vars(ϕ) is a strong KROM-backdoor if and only if X is a hitting set of $\mathcal{F}$.

Towards showing the forward direction, let $X \subseteq$ Vars(ϕ) be a strong KROM-backdoor set of ϕ and suppose for a contradiction that there is a set $F \in \mathcal{F}$ such that $X \cap F = \emptyset$. It follows from the construction of $\mathcal{F}$ that Φ contains a clause C containing at least three literals over the variables in F. Moreover, because of Observation 4 there is a consistent assignment $\theta\colon X \cup \{\, Ox \mid x \in X \wedge O \in \mathcal{O}\} \to \{0,1\}$ that falsifies all literals of C over variables in X. Consequently, $\phi[\theta]$ contains a sub-clause of C that still contains at least three literals over the variables in F. Hence, $\phi[\theta] \notin$ KROM, contradicting our assumption that X is a strong KROM-backdoor set of ϕ.

Towards showing the reverse direction, let $X \subseteq U$ be a hitting set of $\mathcal{F}$ and suppose for contradiction that there is an (consistent) assignment $\theta\colon X \cup \{Ox \mid x \in X \wedge O \in \mathcal{O}\} \to \{0,1\}$ and a clause C in $\phi[\theta]$ containing at least three literals. Let C' be a set of at exactly three literals from C. It follows from the construction of $\mathcal{F}$, that $\mathcal{F}$ contains the set Vars(C'), however, Vars(C') $\cap X = \emptyset$ contradicting our assumption that X is a hitting set of G.

Improved Bounds for Minimal Feedback Vertex Sets in Tournaments

Matthias Mnich[*1] **and Eva-Lotta Teutrine**[2]

1 **Universität Bonn, Bonn, Germany**
mmnich@uni-bonn.de
2 **Universität Bonn, Bonn, Germany**
eva.teutrine@uni-bonn.de

Abstract

We study feedback vertex sets (FVS) in tournaments, which are orientations of complete graphs. As our main result, we show that any tournament on n nodes has at most 1.5949^n minimal FVS. This significantly improves the previously best upper bound of 1.6667^n by Fomin et al. (STOC 2016). Our new upper bound almost matches the best known lower bound of $21^{n/7} \approx 1.5448^n$, due to Gaspers and Mnich (ESA 2010). Our proof is algorithmic, and shows that all minimal FVS of tournaments can be enumerated in time $O(1.5949^n)$.

1998 ACM Subject Classification F2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Exponential-time algorithms, feedback vertex sets, tournaments

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.24

1 Introduction

The Minimum Feedback Vertex Set (FVS) problem in directed graphs is a fundamental problem in combinatorial optimization: given a directed graph G, find a smallest set of vertices in G whose removal yields an acyclic digraph. This problem belongs to Karp's original list of 21 NP-complete problems [8].

The Minimum FVS problem remains NP-complete even in tournaments [13], which are orientations of complete undirected graphs. In other words, a tournament T is a digraph with exactly one arc between any two of its vertices. Various approaches have been suggested to solve the Minimum FVS problem on tournaments, including approximation algorithms [3, 10], fixed-parameter algorithms [4, 9] as well as exact exponential-time algorithms [4, 5, 6]. In particular, one approach that was used to find a minimum FVS is to list all inclusion-minimal FVS of a given tournament using a polynomial-delay enumeration algorithm [6, 12]. The run time of this approach is within a polynomial factor of the number $M(T)$ of minimal FVS in T. Therefore, the complexity of the Minimum FVS problem is within a polynomial factor of the maximum of $M(T)$ over all n-vertex tournaments, which we denote by $M(n)$.

The first one to provide non-trivial bounds on $M(n)$ was Moon [11], who in 1971 established that $1.4757^n \leq M(n) \leq 1.7170^n$. This was improved by Gaspers and Mnich [6] in 2010 to $1.5448^n \leq M(n) \leq 1.6740^n$. Very recently, an improvement on the upper bound was made by Fomin et al. [5], who show that $M(n) \leq 1.6667^n$. The problem of exactly determining $M(n)$ was explicitly posed by Woeginger [15].

* Supported by ERC Starting Grant 306465 (BeyondWorstCase).

Table 1 State of the art for lower and upper bounds on the number of minimal FVS in tournaments.

$M(n)$	lower bound	upper bound
Moon (1971)	1.4757^n	1.7170^n
Gaspers and Mnich (ESA 2010)	$21^{n/7} \approx 1.5448^n$	1.6740^n
Fomin et al. (STOC 2016)		1.6667^n
This paper		1.5949^n
This paper, regular tournaments:		$21^{n/7} \approx 1.5448^n$

Our Contributions

In this paper we make significant progress on establishing better bounds for $M(n)$. Our main combinatorial result is as follows:

▶ **Theorem 1.** *Any tournament of order n has at most $M(n) \leq 1.5949^n$ minimal FVS.*

We also consider regular tournaments (in which all vertices have the same out-degree), because the best known lower bound on $M(n)$ is attained by regular tournaments. For regular tournaments, we show an upper bound on $M(n)$ that matches the lower bound:

▶ **Theorem 2.** *Any regular tournament of order n has at most $21^{n/7}$ minimal FVS, and this is sharp: some regular tournament of order n has exactly $21^{n/7}$ minimal FVS.*

Table 1 provides an overview on lower and upper bounds on $M(n)$.

Our proof of Theorem 1 is inspired by the one of Gaspers and Mnich [6] for their weaker upper bound. Their proof works by induction on the number n of nodes in the input tournament T. Starting with T, they consider a vertex v with maximum out-degree Δ, and depending on the value of Δ and neighbors of v, they construct subtournaments by deleting distinct vertices, such that each maximal transitive vertex set of T is contained in at least one subtournament. Applying the induction hypothesis to the subtournaments then implies their upper bound.

Here, we use a refined technique, that yields upper bounds on the number of inclusion-maximal vertex sets with certain properties. Namely, in addition to deleting vertices to generate subtournaments, we also keep fixed vertex sets. Within these subtournaments we only consider maximal transitive vertex sets that contain all the fixed vertices. We introduce a new function $M(n, k)$ for the maximum number of maximal transitive vertex sets in a tournament of order n containing a fixed set of k vertices, and we will show that $M(n, k) \leq 1.5949^{n-k}$ for all $0 \leq k \leq n$. A similar approach has been used by Gupta et al. [7] to bound the number of maximal r-regular induced subgraphs in undirected graphs.

Our combinatorial result has algorithmic consequences. First, our proof of Theorem 1 is algorithmic, and shows that all minimal FVS of any tournament of order n can be listed in time $O(1.5949^n)$. Second, using an algorithm by Gaspers and Mnich [6] to list all minimal FVS of a tournament with polynomial delay and in polynomial space, we directly obtain the following:

▶ **Corollary 3.** *Given any tournament T of order n, all its minimal FVS can be listed in time $M(T) \cdot n^{O(1)} = O(1.5949^n)$ with polynomial delay and in polynomial space.*

Enumerating the minimal FVS in tournaments has several interesting applications. For example, Banks [1] introduced the notion "Banks winner" in a social choice context, which is a vertex v with in-degree 0 in a subtournament induced by a maximal transitive vertex

set. Brandt et al. [2] consider the problem of determining the "Banks set", which is the set of all Banks winners. As Woeginger [14] showed that deciding whether a vertex is a Banks winner is NP-complete, a feasible approach to determine the Banks set is to enumerate all minimal FVS. For this purpose, Brandt et al. [2] implemented the algorithm of Gaspers and Mnich. Thus, our new algorithm in this paper can be used to compute the Banks set of a tournament asymptotically faster.

2 Preliminaries

A *tournament* $T = (V, A)$ is a directed graph with exactly one edge between each pair of vertices. We denote the set of all tournaments with n vertices by $\mathcal{T}_n$. A *feedback vertex set* (FVS) of T is a set $F \subseteq V(T)$ such that $T - F$ is free of (directed) cycles, where $T - F$ is the induced subgraph of T after removing all vertices in F. An FVS is *minimal* if none of its proper subsets is an FVS.

Denote by $M(T)$ the number of minimal FVS in a tournament T, and define

$$M(n) = \max_{T \in \mathcal{T}_n} M(T)$$

to be the maximum number of minimal FVS in tournaments of order n.

Let $T = (V, A)$ be a tournament. For a set $V' \subseteq V$, let $T[V']$ be the subtournament of T induced by V'. For each $v \in V$, let $N^-(v) = \{u \in V \mid (u, v) \in A\}$ and let $N^+(v) = \{u \in V \mid (v, u) \in A\}$. We write $v \to u$ if $u \in N^+(v)$ and call v a *predecessor* of u and u a *successor* of v. For each $v \in V$, its *in-degree* is $d^-(v) = |N^-(v)|$ and its *out-degree* is $d^+(v) = |N^+(v)|$; call T *regular* if all its vertices have the same out-degree. Let $\Delta^+(T)$ denote the maximum out-degree over all vertices of T. Further, T is *strong* if there is a directed path from v to u for each pair of vertices $v, u \in V$; let $\mathcal{T}_n^\star$ denote the set of strong tournaments of order n. Note that any tournament can uniquely be decomposed into strong subtournaments $S_1, \ldots, S_r$ such that $v \to u$ for all $v \in V(S_i)$, $u \in V(S_j)$ for all $i < j$.

▶ **Observation 4.** *For any tournament T, we obtain $M(T) = M(S_1) \cdot \ldots \cdot M(S_r)$.*

Therefore, we can bound $M(n)$ from above by β^n for some β by considering strong tournaments of every order n.

Our proofs will use the following well-known observation about cycles in tournaments:

▶ **Lemma 5.** *In a tournament, any vertex contained in a cycle is contained in a triangle.*

Proof. Let $v_1, \ldots, v_\ell$ be a shortest cycle containing v_1 with $\ell > 3$, $v_i \to v_{i+1}$ for all $i \in \{1, \ldots, \ell - 1\}$ and $v_\ell \to v_1$. Depending on the orientation of the arc between v_1 and v_3, either v_1, v_2, v_3 form a triangle or $v_1, v_3, v_4, \ldots, v_\ell$ is a shorter cycle containing v_1. ◀

We call a vertex set *transitive* if its induced subtournament is acyclic. Thus, a vertex set is a maximal transitive vertex set if and only if its complement is a minimal FVS. Instead of counting minimal FVS, we count maximal transitive vertex sets. The next property of maximal transitive vertex sets was already used by Moon [11] and Gaspers and Mnich [6]:

▶ **Lemma 6.** *For any tournament T, $M(T) \leq \sum_{v \in V(T)} M(d^+(v))$.*

Proof. Any maximal transitive vertex set W of T has a vertex v with in-degree 0 in $T[W]$. Hence, W is also a maximal transitive vertex set in $T[N^+(v) \cup \{v\}]$; this yields the bound. ◀

Lemma 6 allows us to effectively bound $M(T)$ in terms of a recurrence relation, in particular in combination with the next lemma that extends Lemma 3 by Gaspers and Mnich [6]:

▶ **Lemma 7.** *Let $n \in \mathbb{N}$ and let $T \in \mathcal{T}_n^\star$. Then either T is regular, or for any $d \in \mathbb{N}$ at most $2d$ vertices in T have out-degree at least $n - d - 1$.*

Proof. Let $\tilde{V}$ be the set of vertices in T with out-degree at least $n - d - 1$. Then any vertex in $\tilde{V}$ has in-degree at most d. Hence,

$$\sum_{v \in \tilde{V}} |N^-(v)| \leq |\tilde{V}| \cdot d \ . \tag{1}$$

We may suppose that $\tilde{V} \neq \emptyset$, for otherwise the statement of the lemma holds. We distinguish two cases.

Consider first the case that $\tilde{V} \neq V(T)$. Then, since T is strong and $\tilde{V} \neq \emptyset$, there is some arc from $V(T) \setminus \tilde{V}$ to $\tilde{V}$. There are $\binom{|\tilde{V}|}{2}$ arcs between vertices in $\tilde{V}$. Therefore, $\sum_{v \in \tilde{V}} |N^-(v)| \geq \binom{|\tilde{V}|}{2} + 1$. Combining this inequality with (1) and solving for $d \in \mathbb{N}$ yields $|\tilde{V}| \leq 2d$.

Second, consider the case that $\tilde{V} = V(T)$. We may suppose that T is not regular, for otherwise the statement of the lemma holds. Note that not every vertex of $\tilde{V} = V(T)$ can have in-degree exactly d, since T is not regular. Hence, some vertex in $\tilde{V}$ has in-degree at most $d - 1$. Consequently,

$$\sum_{v \in \tilde{V}} |N^-(v)| \leq (|\tilde{V}| - 1) \cdot d + (d - 1) \ .$$

There are $\binom{|\tilde{V}|}{2}$ arcs between vertices in $\tilde{V}$. Thus, $\sum_{v \in \tilde{V}} |N^-(v)| \geq \binom{|\tilde{V}|}{2}$. Combining these two inequalities and solving for $d \in \mathbb{N}$ yields $|\tilde{V}| \leq 2d$. ◀

We remark that a regular tournament may have more than $2d$ vertices of out-degree at least $n - d - 1$, as witnessed for instance by the triangle and $d = 1$.

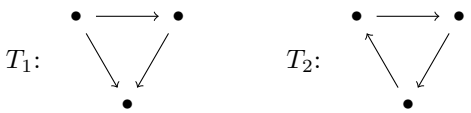
3 Improved Upper Bound on the Maximum Number of Minimal FVS

In this section we show that the maximum number $M(n)$ of minimal FVS in any tournament of order n is bounded from above by 1.5949^n. For this purpose, for a tournament T and $V' \subseteq V(T)$ let $M(T, V')$ be the number of maximal transitive vertex sets in T that contain all vertices in V'. Also, let

$$M(n, k) = \max_{T \in \mathcal{T}_n, V' \subseteq V(T), |V'| = k} M(T, V') \ .$$

Note that $M(n) = M(n, 0)$.

Example. To clarify the definition, we compute $M(3, 1)$. Precisely, we show that $M(3, 1) = 2$. There are two non-isomorphic tournaments for $n = 3$:



The tournament T_1 is acyclic and thus has only a single maximal transitive vertex set, $V(T_1)$. Thus, $M(T_1, \{v\}) = 1$ for all $v \in V(T_1)$. The tournament T_2 has three maximal transitive vertex sets, each consisting of exactly two vertices. Thus, each vertex of T_2 is contained in

exactly two maximal transitive vertex sets. This yields $M(T_2, \{v\}) = 2$ for all $v \in V(T_2)$. Summarizing, we get $M(3,1) = 2$. ◀

Henceforth, fix $\beta = 1.5949$. We will show that $M(n,k) \leq \beta^{n-k}$ for all $n \in \mathbb{N}$ and $k \in \{0, \ldots, n\}$. To this end, ideally we would like to prove the following statements:

(I) *It holds* $M(n,k) \leq \beta^{n-k}$ *for all* $n \geq k > 0$.

(II) *It holds* $M(n,0) \leq \beta^n$.

Unfortunately, we are unable to do prove these directly. The reason is that our proof of Statement (I) for a fixed pair (n,k) with $n \geq k > 0$ *depends* on the validity of Statement (II) for values $\tilde{n} < n$. Vice-versa, our proof of the validity of Statement (II) for fixed $n \in \mathbb{N}$ depends on the validity of Statement (I).

We will therefore establish the following two lemmas:

▶ **Lemma 8.** *Let* $n \in \mathbb{N}$. *If* $M(\tilde{n}) \leq \beta^{\tilde{n}}$ *and* $M(\tilde{n}, \tilde{k}) \leq \beta^{\tilde{n}-\tilde{k}}$ *holds for all* $0 < \tilde{k} \leq \tilde{n} < n$, *then* $M(n,k) \leq \beta^{n-k}$ *for* $0 < k \leq n$.

The proof of Lemma 8 is given in Sect. 4.

▶ **Lemma 9.** *Let* $n \in \mathbb{N}$. *If* $M(\tilde{n}) \leq \beta^{\tilde{n}}$, $M(\tilde{n}, \tilde{k}) \leq \beta^{\tilde{n}-\tilde{k}}$ *and* $M(n, \tilde{k}) \leq \beta^{n-\tilde{k}}$ *for all* $0 < \tilde{k} \leq \tilde{n} < n$, *then* $M(n) \leq \beta^n$.

The proof of Lemma 9 consists of a lengthy case analysis; we thus defer it to the full version of this paper.

We are ready to prove Theorem 1.

Proof of Theorem 1. We show that for all $n \in \mathbb{N}$, it holds $M(n) \leq 1.5949^n$. Clearly, $M(1) \leq 1 \leq 1.5949$ and $M(1,k) \leq 1 \leq 1.5949^{1-k}$ for all $k \in \{0,1\}$. This yields our induction hypothesis. Lemma 8 and Lemma 9 yield our inductive step and prove the desired bound on $M(n)$ for all $n \in \mathbb{N}$. ◀

4 Proof of Lemma 8

In this section we prove Lemma 8. For sake of contradiction, suppose that the statement of the lemma does not hold. Let (T, V') be a minimum counterexample, that is, T is a tournament and $V' \subseteq V(T)$ such that $|V(T)| - |V'|$ is minimum and $M(T, V') > \beta^{|V(T)|-|V'|}$. Throughout this section, write $n = |V(T)|$ and $k = |V'| > 0$.

We will distinguish several cases and show that $M(T, V') \leq \beta^{n-k}$ for each of them; this yields the desired contradiction (and hence the truth of the statement of the lemma). In each case, we will use the minimality of (T, V') to bound $M(T, V')$ from above.

Case 1: Three vertices in V' form a triangle. Then, as no transitive vertex set contains all of these three vertices, $M(T, V') = 0 \leq \beta^{n-k}$.

Case 2: Two vertices in V' form a triangle with some vertex $v \in V(T) \setminus V'$. Any transitive vertex set that contains all vertices in V' does not contain v. Hence,

$$M(T, V') = M(T - \{v\}, V') \leq M(n-1, k) \leq \beta^{n-k-1} \leq \beta^{n-k} \quad .$$

Case 3: There is a vertex $v \in V'$ that is not contained in any cycle of T. Then, a set $W \supseteq V'$ is a maximal transitive vertex set of T if and only if $W \setminus \{v\} \supseteq V' \setminus \{v\}$ is a maximal transitive vertex set of $T - v$. This yields

$$M(T, V') = M(T - \{v\}, V' \setminus \{v\}) \leq M(n-1, k-1) \leq \beta^{n-k} \ .$$

▶ Remark. We remark that it is this case where we rely on the validity of Lemma 9, namely that $M(\tilde{n}) < \beta^{\tilde{n}}$ for $\tilde{n} < n$. The reason is that possibly $V' \setminus \{v\} = \emptyset$, in which case $k - 1 = 0$ and we need that $M(n-1, 0) \leq \beta^{n-1}$.

Henceforth, consider pairs (T, V') to which Cases 1–3 do not apply.

▶ **Observation 10.** *If Cases 1–3 do not apply to* (T, V')*, then* (i) *any vertex of* V' *is contained in at least one triangle (by Lemma 5), and* (ii) *any triangle contains at most one vertex of* V'*.*

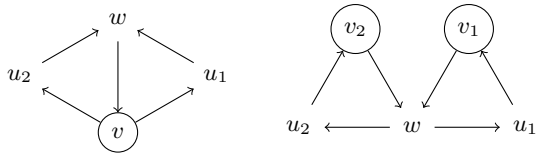
▶ Remark. We remark that with Case 1–3 we can already show a bound of $M(T, V') \leq \beta_0^{n-k}$ for $\beta_0 = 1.6181$ (under the conditions imposed by the lemma). By Observation 10, there is a vertex $v \in V'$ that forms a triangle with two vertices $w_1, w_2 \notin V'$. Any maximal transitive vertex set $W \supseteq V'$ (and thus containing v) cannot contain both w_1 and w_2. Therefore, $w_1 \in W$ implies $w_2 \notin W$ and we get

$$\begin{aligned} M(T, V') &\leq M(T - \{w_1\}, V') + M(T - \{w_2\}, V' \cup \{w_1\}) \\ &\leq M(n-1, k) + M(n-1, k+1) \leq \beta_0^{n-k-1} + \beta_0^{n-k-2} \ , \end{aligned}$$

which is bounded by β_0^n for $\beta_0 = 1.6181$.

The subsequent cases allow us to improve $\beta_0 = 1.6181$ to $\beta = 1.5949$.

Case 4: There is a vertex $w \notin V'$ that is contained in two distinct triangles, both of which contain a vertex from V' (possibly shared by both triangles). Then we are in one of two cases, where vertices in V' are circled:

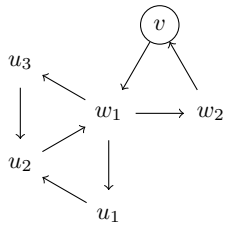


Let $(w, u_1, v_1), (w, u_2, v_2)$ be distinct triangles containing w, such that $v_1, v_2 \in V'$ where possibly $v_1 = v_2$. Let W be a maximal transitive vertex set of T containing V'. Then either $w \notin W$ or $w \in W$. Clearly, if $w \in W$ then $u_1, u_2 \notin W$. We therefore have

$$\begin{aligned} M(T, V') &\leq M(T - \{w\}, V') + M(T - \{u_1, u_2\}, V' \cup \{w\}) \\ &\leq M(n-1, k) + M(n-2, k+1) \leq \beta^{n-k-1} + \beta^{n-k-3}. \end{aligned}$$

The last expression on the right-hand side is at most β^{n-k}, since $\beta \geq 1.4656$.

Case 5: There are vertices $v \in V'$ and $w_1, w_2 \in V(T) \setminus V'$ that form a triangle, such that w_1 also belongs to triangles $(w_1, u_1, u_2), (w_1, u_2, u_3)$ for some $u_1, u_2, u_3 \in V(T) \setminus \{v, w_2\}$.



Then we can assume that $u_1, u_2, u_3 \in V(T) \setminus V'$, as otherwise Case 2 or Case 4 would apply. Any transitive vertex set $W \supseteq V'$ either contains w_1 or not. If $w_1 \in W$ then $w_2 \notin W$. Moreover, $w_1 \in W$ implies that either $u_2 \notin W$, or $u_2 \in W$ but $u_1, u_3 \notin W$. Thus,

$$\begin{aligned} M(T, V') &\leq M(T - \{w_1\}, V') + M(T - \{w_2\}, V' \cup \{w_1\}) \\ &\leq M(T - \{w_1\}, V') + M(T - \{w_2, u_2\}, V' \cup \{w_1\}) \\ &\quad + M(T - \{w_2, u_1, u_3\}, V' \cup \{w_1, u_2\}) \\ &\leq M(n-1, k) + M(n-2, k+1) + M(n-3, k+2) \\ &\leq \beta^{n-k-1} + \beta^{n-k-3} + \beta^{n-k-5} \ . \end{aligned}$$

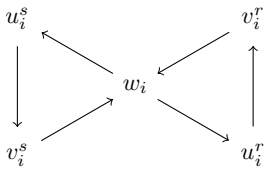
The last expression on the right-hand side is at most β^{n-k}, since $\beta \geq 1.5702$.

Henceforth, we assume that Cases 1-5 do not apply to (T, V'). Then some vertex $v_0 \in V'$ forms a triangle with some $w_1, w_2 \in V(T) \setminus V'$, as Cases 1-3 do not apply. For $i = 1, 2$, let Δ_i be the set of triangles $t_i = (u_i, v_i, w_i)$ that are disjoint from w_{3-i} and for which $T[\{u_i, v_i, v'\}]$ is acyclic for all $v' \in V'$. Consequently, all triangles in $\Delta_1 \cup \Delta_2$ are disjoint from V', as Case 4 does not apply. Further, all triangles in Δ_i are pairwise edge-disjoint (as Case 5 does not apply), and therefore intersect only in w_i.

To prove an upper bound on $M(T, V')$, we again distinguish the maximal transitive vertex sets that contain w_1 or w_2, from those that do not contain either of them. Let W be a maximal transitive vertex set of T containing V'.

First consider that $w_1, w_2 \notin W$. Then, $T[W \cup \{w_i\}]$ contains a cycle for $i = 1, 2$, by maximality of W. Thus, by Lemma 5, there is a triangle $t = (w_i, z_1, z_2)$ for some $z_1, z_2 \in W$. We have that $t \in \Delta_i$, since z_1, z_2 do not form a triangle with any $v' \in V'$ as $z_1, z_2 \in W$. Thus, those W with $w_1, w_2 \notin W$ can be partitioned into $|\Delta_i|$ classes, where the r-th class contains the sets W that contain the two vertices of the r-th triangle in Δ_i.

To use this argument effectively, we need some further observations about the relation among triangles in $\Delta_1 \cup \Delta_2$. Consider two triangles $t_i^r = (u_i^r, v_i^r, w_i), t_i^s = (u_i^s, v_i^s, w_i) \in \Delta_i$:



Since all triangles that contain w_i are pairwise edge-disjoint (as Case 5 does not apply), the edge between u_i^r and v_i^s has to be directed from v_i^s to u_i^r; else, w_i, u_i^r, v_i^s would form a triangle that is not edge-disjoint from the triangle w_i, u_i^r, v_i^r. Likewise, the edge between u_i^s and v_i^r has to be directed from v_i^r to u_i^s. Ignoring symmetries obtained by swapping the roles of t_i^r and t_i^s, there are only two possibilities how the two remaining edges (between u_i^r, u_i^s and v_i^r, v_i^s) can be oriented:

We refer to the situation in the left figure as **Case A**, and to the situation in the right figure as **Case B**. Note that in Case A, (u_i^r, u_i^s, v_i^s) and (v_i^r, u_i^s, v_i^s) form triangles; while in Case B, triangles are formed by (u_i^r, u_i^s, v_i^s) and (u_i^r, v_i^r, v_i^s).

▶ **Observation 11.** *In Case A, $u_i^s, v_i^s \in W$ implies that $u_i^r, v_i^r \notin W$. In Case B, $u_i^r, v_i^r \in W$ implies that $v_i^s \notin W$; and $u_i^s, v_i^s \in W$ implies that $u_i^r \notin W$.*

Thus, for each $t_i^r = (u_i^r, v_i^r, w_i) \in \Delta_i$ let $V_{t_i^r}$ be the set of vertices that are excluded from those W with $u_i^r, v_i^r \in W$ due to Observation 11.

In Lemma 12, we will show that any two triangles in Δ_1 and Δ_2 are vertex-disjoint. Therefore, for each $t_i^r \in \Delta_i$, every vertex in $V_{t_i^r}$ is not contained in any triangle of Δ_{3-i}. This implies that for any pair of triangles $t_1 \in \Delta_1, t_2 \in \Delta_2$ the sets V_{t_1}, V_{t_2} are disjoint. Altogether, this means that we can bound the number of maximal transitive vertex sets $W \supseteq V'$ not containing w_1, w_2 from above by

$$\sum_{t=(w_1,u_1,u_2)\in\Delta_1} \sum_{t'=(w_2,u_1',u_2')\in\Delta_2} M(T - \{w_1, w_2\} - V_t - V_{t'}, V' \cup \{u_1, u_2, u_1', u_2'\})$$
$$\leq \sum_{t\in\Delta_1} \sum_{t'\in\Delta_2} \beta^{n-2-|V_t|-|V_{t'}|-(k+4)} \leq \beta^{n-k-6} \underbrace{\sum_{t\in\Delta_1} \beta^{-|V_t|} \sum_{t'\in\Delta_2} \beta^{-|V_{t'}|}}_{(\star)} . \tag{2}$$

Thus, our goal is now to bound $(\star)$. Fix $i \in \{1, 2\}$. Let $t_i^1, \ldots, t_i^{|\Delta_i|}$ be an ordering of the triangles in Δ_i such that $|V_{t_i^r}| \leq |V_{t_i^s}|$ for $1 \leq r < s \leq |\Delta_i|$. Then for any pair $r, s \in \{1, \ldots, \Delta_i\}$ with $r \neq s$, Observation 11 implies

$$|V_{t_i^s} \cap \{u_i^r, v_i^r\}| + |V_{t_i^r} \cap \{u_i^s, v_i^s\}| = 2 \ .$$

Thus, for any $r < s$, since $\beta \geq 1$, we get

$$\beta^{-|V_{t_i^s}|} + \beta^{-|V_{t_i^r}|} \leq \beta^{-(|V_{t_i^s} \cup \{u_i^r, v_i^r\}|)} + \beta^{-(|V_{t_i^r} \setminus \{u_i^s, v_i^s\}|)} \ .$$

Thus, we can bound $(\star)$ by the case where for any $r < s$,

$$|V_{t_i^s} \cap \{u_i^r, v_i^r\}| = 2 \wedge |V_{t_i^r} \cap \{u_i^s, v_i^s\}| = 0 \ .$$

Hence, we can assume that $|V_{t_i^r}| = 2(r-1)$ for all $r = 1, \ldots, |\Delta_i|$. We obtain

$$\sum_{t\in\Delta_i} \beta^{-|V_t|} \leq \sum_{r=0}^{|\Delta_i|-1} \beta^{-2r} \leq \sum_{r=0}^{\infty} \beta^{-2r} = \frac{\beta^2}{\beta^2 - 1} \ .$$

Consequently, $(\star)$ is bounded by $(\frac{\beta^2}{\beta^2-1}) \cdot (\frac{\beta^2}{\beta^2-1}) = \frac{\beta^4}{(\beta^2-1)^2}$.

Let us now prove that indeed any triangle in Δ_1 is disjoint from every triangle in Δ_2.

▶ **Lemma 12.** *Let v_0, w_1, w_2, Δ_1 and Δ_2 be defined as before. Then any triangle in Δ_1 is vertex-disjoint from every triangle in Δ_2.*

Proof. First note that V' is a transitive set, as Case 1 does not apply. Thus, the vertices in V' admit a topological order such that $v'_x \to v'_y$ for all $v'_x, v'_y \in V'$ with $x > y$. Second, for each vertex $z \in V(T) \setminus V'$ the set $V' \cup \{z\}$ is a transitive set, as Case 2 does not apply. Therefore, the vertices of $V(T) \setminus V'$ can be partitioned into layers $Z_1, \ldots, Z_\ell$ such that for each $z \in Z_r$, $z \to v'_s$ if and only if $s < r$.

We claim that for $i = 1, 2$, the vertices of any triangle $(u_i^r, v_i^r, w_i) \in \Delta_i$ all belong to the same layer. This implies in particular that for $i = 1, 2$, *all* vertices in triangles of Δ_i belong to the same layer. Since w_1 and w_2 are in different layers (as $v_0 \to w_1$, $w_2 \to v_0$), this shows that any triangle in Δ_1 is vertex-disjoint from any triangle in Δ_2.

To show the claim, let $i \in \{1, 2\}$ and let $(u_i^r, v_i^r, w_i) \in \Delta_i$ be a triangle with $w_i \to u_i^r, u_i^r \to v_i^r, v_i^r \to w_i$. Suppose that $u_i^r \in Z_u, v_i^r \in Z_v, w_i \in Z_w$ for some $u, v, w \in \{1, \ldots, l\}$. So we must show that $u = v = w$ to prove the claim.

If $u < w$ then w_i, u_i^r, v'_u form a triangle, contradicting that Case 4 does not apply. If $v > w$ then w_i, v_i^r, v'_v form a triangle, again contradicting that Case 4 does not apply. Hence, $v \leq w \leq u$ holds. If $v < u$ then u_i^r, v_i^r, v'_v form a triangle, contradicting the definition of Δ_i. So indeed $u = v = w$, and the claim holds. ◀

To complete the proof of Lemma 8, we must also consider those $W \supseteq V'$ that contain exactly one of w_1, w_2 (recall that at most one of w_1, w_2 belongs to W as $v_0 \in W$, so $w_i \in W$ implies $w_{3-i} \notin W$ for $i = 1, 2$). Overall, if Cases 1–5 do not apply, with the obtained bound on $(\star)$, by (2) we have

$$\begin{aligned} M(T, V') &\leq M(T - \{w_1\}, V' \cup \{w_2\}) + M(T - \{w_2\}, V' \cup \{w_1\}) + \beta^{n-6-k} \cdot \frac{\beta^4}{(\beta^2-1)^2} \\ &\leq 2 \cdot M(n-1, k+1) + \beta^{n-6-k} \cdot \frac{\beta^4}{(\beta^2-1)^2} \leq 2 \cdot \beta^{n-k-2} + \frac{\beta^{n-k-2}}{(\beta^2-1)^2} \ . \end{aligned}$$

The last expression on the right-hand side is at most β^{n-k}, since $\beta \geq 1.5703$.

This completes the proof of Lemma 8. ◀

5 Discussion

In this paper we narrowed the gap between the lower and upper bounds for the maximum number $M(n)$ of minimal FVS in n-vertex tournaments, to $1.5448^n \leq M(n) \leq 1.5949^n$. It remains to determine the growth of $M(n)$ exactly—Gaspers and Mnich [6] conjectured that $M(n) \leq 21^{n/7} \approx 1.5448^n$ for all $n \in \mathbb{N}$, and we re-pose this conjecture here.

In a different direction, it would be interesting to prove non-trivial upper bounds of the form c^n for some constant $c < 2$, on the number of minimal FVS in general directed graphs. As far as we know, currently only a bound of $2^n/\sqrt{n}$ is known, implied by Sperner's Lemma.

Acknowledgements. We thank the anonymous reviewers of an earlier version for helpful remarks how we could improve the presentation of these results.

References

1 Jeffrey S. Banks. Sophisticated voting outcomes and agenda control. *Soc. Choice Welf.*, 1(4):295–306, 1985.

2 Felix Brandt, Andre Dau, and Hans Georg Seedig. Bounds on the disparity and separation of tournament solutions. *Discrete Appl. Math.*, 187:41–49, 2015.

3 Mao-Cheng Cai, Xiaotie Deng, and Wenan Zang. An approximation algorithm for feedback vertex sets in tournaments. *SIAM J. Comput.*, 30(6):1993–2007, 2001.

4 Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truss. Fixed-parameter tractability results for feedback set problems in tournaments. *J. Discrete Algorithms*, 8(1):76–86, 2010.

5 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. In *Proc. STOC 2016*, pages 764–775, 2016.

6 Serge Gaspers and Matthias Mnich. Feedback vertex sets in tournaments. *J. Graph Theory*, 72(1):72–89, 2013.

7 Sushmita Gupta, Venkatesh Raman, and Saket Saurabh. Maximum r-regular induced subgraph problem: fast exponential algorithms and combinatorial bounds. *SIAM J. Discrete Math.*, 26(4):1758–1780, 2012.

8 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.

9 Mithilesh Kumar and Daniel Lokshtanov. Faster exact and parameterized algorithm for feedback vertex set in tournaments. In *Proc. STACS 2016*, volume 47 of *Leibniz Int. Proc. Informatics*, pages 49:1–49:13, 2016.

10 Matthias Mnich, Virginia Vassilevska Williams, and László A. Végh. A 7/3-approximation for feedback vertex sets in tournaments. In *Proc. ESA 2016*, volume 57 of *Leibniz Int. Proc. Informatics*, pages 67:1–67:14, 2016.

11 J. W. Moon. On maximal transitive subtournaments. *Proc. Edinburgh Math. Soc. (2)*, 17:345–349, 1970/71.

12 Benno Schwikowski and Ewald Speckenmeyer. On enumerating all minimal solutions of feedback problems. *Discrete Appl. Math.*, 117(1-3):253–265, 2002.

13 Ewald Speckenmeyer. On feedback problems in digraphs. In *Proc. WG 1989*, volume 411 of *Lecture Notes Comput. Sci.*, pages 218–231. Springer, 1990.

14 Gerhard J. Woeginger. Banks winners in tournaments are difficult to recognize. *Soc. Choice Welf.*, 20(3):523–528, 2003.

15 Gerhard J. Woeginger. Open problems around exact algorithms. *Discrete Appl. Math.*, 156(3):397–405, 2008.

Ground Reachability and Joinability in Linear Term Rewriting Systems are Fixed Parameter Tractable with Respect to Depth

Mateus de Oliveira Oliveira*

Department of Informatics, University of Bergen, Norway
mateus.oliveira@uib.no

Abstract

The ground term reachability problem consists in determining whether a given variable-free term t can be transformed into a given variable-free term t' by the application of rules from a term rewriting system $\mathfrak{R}$. The joinability problem, on the other hand, consists in determining whether there exists a variable-free term t'' which is reachable both from t and from t'. Both problems have proven to be of fundamental importance for several subfields of computer science. Nevertheless, these problems are undecidable even when restricted to linear term rewriting systems. In this work, we approach reachability and joinability in linear term rewriting systems from the perspective of parameterized complexity theory, and show that these problems are fixed parameter tractable with respect to the depth of derivations. More precisely, we consider a notion of parallel rewriting, in which an unbounded number of rules can be applied simultaneously to a term as long as these rules do not interfere with each other. A term t_1 can reach a term t_2 in depth d if t_2 can be obtained from t_1 by the application of d parallel rewriting steps. Our main result states that for some function $f(\mathfrak{R}, d)$, and for any linear term rewriting system $\mathfrak{R}$, one can determine in time $f(\mathfrak{R}, d) \cdot |t_1| \cdot |t_2|$ whether a ground term t_2 can be reached from a ground term t_1 in depth at most d by the application of rules from $\mathfrak{R}$. Additionally, one can determine in time $f(\mathfrak{R}, d)^2 \cdot |t_1| \cdot |t_2|$ whether there exists a ground term u, such that u can be reached from both t_1 and t_2 in depth at most d. Our algorithms improve exponentially on exhaustive search, which terminates in time $2^{|t_1| \cdot 2^{O(d)}} \cdot |t_2|$, and can be applied with regard to any linear term rewriting system, irrespective of whether the rewriting system in question is terminating or confluent.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems, F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Linear Term Rewriting Systems, Ground Reachability, Ground Joinability, Fixed Parameter Tractability

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.25

1 Introduction

Term rewriting systems have played a major role in several fields of computer science, such as, functional programming languages, specification of abstract data types, symbolic computation and automated theorem proving [16, 1, 2]. Many practical and theoretical aspects of the theory of term rewriting system revolve around two fundamental problems: ground reachability, and ground joinability. In the former, given a rewriting system $\mathfrak{R}$ and

* The author is currently supported by the Bergen Research Foundation. This work was concluded while the author was at the Czech Academy of Sciences, supported by the European Research Council (grant number 339691).

two ground terms t and t' one is asked to determine whether t' can be reached from t by the application of a sequence of rewriting rules from $\mathfrak{R}$. In the latter, joinability, one is asked whether there is a ground term u which can be reached from both t and t'. Both problems are known to be decidable for several restricted classes of rewriting systems, such as ground rewriting systems, right-ground systems, shallow right-linear systems, and left-linear growing systems [13, 14, 15, 18]. On the other hand, ground reachability and joinability on linear term rewriting systems are known to be undecidable if no other restriction is imposed [20]. Indeed, it can be shown that for each Turing machine M, there exists a linear term rewriting system $\mathfrak{R}_M$ such that the halting problem for M can be reduced to ground reachability (joinability) in $\mathfrak{R}_M$ [20].

A rewriting system $\mathfrak{R}$ is said to be linear if it contains only rules of the form $l \to r$ where $var(r) \subseteq var(l)$ and each variable occurs at most once in l, and at most once in r. For instance, the associativity rule $x \cdot (y \cdot z) \to (x \cdot y) \cdot z$ is linear. In this work, we show that despite the undecidability of ground state reachability and joinability for linear term rewriting systems, both problems are fixed parameter tractable with respect to the depth of derivations. In this context, we consider a notion of parallel rewriting in which an unbounded number of rules can be applied simultaneously to a term, as long as these rules do not interfere with each other [19, 4]. Such a simultaneous application of independent rewriting rules is known in term-rewriting theory literature as *multi-step*. We say that a term t can reach a term t' in depth at most d if t' can be obtained from t by the application of d multi-steps.

▶ **Theorem 1** (Main Theorem). *There is a function $f(\mathfrak{R}, d)$ such that for any set of linear term rewriting rules $\mathfrak{R}$, and any ground terms t and t' over Σ,*

- *one can determine in time $f(\mathfrak{R}, d) \cdot |t| \cdot |t'|$ whether t' can be reached from t in depth at most d.*
- *one can determine in time $f(\mathfrak{R}, d)^2 \cdot |t| \cdot |t'|$ whether there exists a ground term u such that u is reachable in depth at most d from both t and t'.*

Our algorithms improve substantially on the running time of exhaustive search. We note that given a term t of size $|t|$, there may be up to $2^{O(|t|)}$ possible ways of applying simultaneous rewriting rules to t. Additionally, if a term t' is obtained from t in depth d, then the size of t' may be as large as $|t| \cdot 2^{O(d)}$. Therefore, as many as $2^{|t| \cdot 2^{O(d)}}$ distinct terms may be derived from a term t in depth at most d. Indeed this upper bound is asymptotically tight and can be matched even in ground rewriting systems. Consider for instance, a ranked alphabet $\Sigma = \{g, a\}$ consisting of one binary function symbol g, one constant symbol a, and a term rewriting system $\mathfrak{R}$ consisting of a single rewriting rule $a \to g(a, a)$. Then for any term t over Σ, one can derive at least $2^{|t| \cdot 2^{\Omega(d)}}$ distinct terms from t in depth at most d. In other words, even when d is a constant, determining whether a term t' can be reached from a term t in depth at most d by exhaustive search takes time exponential in t in the worst case, while using our approach, this problem can be solved in time $f(\mathfrak{R}, d) \cdot |t| \cdot |t'|$. We also should note that it is straightforward to define infinite families of pairs of terms (t_n, t'_n) such that t'_n can be reached from t_n in a single multi-step, but which require the application of an unbounded number of sequential individual rewriting steps. For instance, consider the term $t_n = a_1 + b_1 + a_2 + b_2 + ... + a_n + b_n$ where $a_i = 1$, $b_i = 2$ and $+$ is an associative commutative binary function symbol. Then in one multi-step one can reach the term $t'_n = b_1 + a_1 + b_2 + a_2 + ... + b_n + a_n$, whereas one would need to use n individual rewriting steps to derive t' from t. Note that the larger the n, the larger is the number of individual rewriting rules necessary to reach t' from t, while a single multi-step is sufficient for any n.

2 Term Rewriting and Tree Automata

In this section we define standard notions from term rewriting systems and tree automata. Extensive treatments of term rewriting theory can be found in [1, 7] and on tree-automata theory can be found in [5, 12].

2.1 Terms

The set of natural numbers, excluding 0, is denoted by $\mathbb{N}$. We let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. A *ranked alphabet* is a finite set Σ of function symbols together with an arity function $\mathfrak{a} : \Sigma \to \mathbb{N}_0$. Intuitively the arity $\mathfrak{a}(f)$ of a symbol $f \in \Sigma$ specifies the number of inputs of f. A function symbol of arity 0 is called a constant symbol. We let $\mathfrak{a}(\Sigma) = \max\{\mathfrak{a}(f) \mid f \in \Sigma\}$ be the maximum arity of a symbol in Σ. Let X be a finite set of variables and Σ be a ranked alphabet. The set $Ter(\Sigma \cup X)$ of all *terms* over $\Sigma \cup X$ is inductively defined as follows:

- If x is a variable in X then x is a term in $Ter(\Sigma \cup X)$
- if $f \in \Sigma$ and $t_1, ..., t_{\mathfrak{a}(f)}$ are terms in $Ter(\Sigma \cup X)$ then $f(t_1, t_2, ..., t_{\mathfrak{a}(f)})$ is a term in $Ter(\Sigma \cup X)$.

If f is a function symbol of arity 0 then we write simply f to denote the term $f()$. A position for a term t is a string over $\mathbb{N}$. The empty string is denoted by ε. The set of positions for a term t is inductively defined as follows.

- $Pos(t) = \{\varepsilon\}$ if $t \in X$.
- $Pos(f(t_1, ..., t_{\mathfrak{a}(f)})) = \{\varepsilon\} \cup \{i.p \mid 1 \leq i \leq \mathfrak{a}(f),\ p \in Pos(t_i)\}$

We note that if t is either a variable or a function symbol of arity 0, then $Pos(t) = \{\varepsilon\}$. We let $|t| = |Pos(t)|$ denote the *size* of the term t. The *subterm* $t|_p$ of t at position p is inductively defined as follows. At the base case, $t|_\varepsilon = t$. Now, if $t = f(t_1, t_2, ..., t_{\mathfrak{a}(f)})$, then for each $j \in \{1, ..., \mathfrak{a}(f)\}$ and each position $jp \in Pos(t)$, $t|_{jp} = t_j|_p$.

Let $t = f(t_1, ..., t_{\mathfrak{a}(f)})$ be a term in $Ter(\Sigma \cup X)$. We let $\boldsymbol{rs}(t) = f$ be the *root symbol* of t. If $t = x$ for a variable $x \in X$ then we set $\boldsymbol{rs}(t) = x$. For each $p \in Pos(t)$, we let $t(p) = \boldsymbol{rs}(t|_p)$ denote the root symbol of the subterm of t at position p.

We denote by $var(t)$ the set of variables occurring in t. A *ground term* is a term t such that $var(t) = \emptyset$. In other words, a term t is ground if it contains no variables. In some places we may write $t \in Ter(\Sigma)$ to indicate that t is a ground term.

A *substitution* is a function $\sigma : X \to Ter(\Sigma \cup X)$ mapping variables in X to terms in $Ter(\Sigma \cup X)$. If t is a term, and σ is a substitution, then we denote by t^σ the term that is obtained from t by replacing each variable $x \in X$ with the term $\sigma(x)$. If t and s are terms and p is a position in $Pos(t)$ then we denote by $t[s]_p$ the term that is obtained from t by replacing the subterm $t|_p$ with the term s.

2.2 Term Rewriting

A *rewriting rule* is a pair $l \to r$ where l and r are terms in $Ter(\Sigma \cup X)$ with $var(r) \subseteq var(l)$. We say that a rule $l \to r$ is *linear* if each variable occurs at most once in l and at most once in r. Note that this definition of linearity allows $var(l) \cap var(r) \neq \emptyset$. A term rewriting system is any finite set $\mathfrak{R}$ of rewriting rules. We say that $\mathfrak{R}$ is linear if each rewriting rule $l \to r$ in $\mathfrak{R}$ is linear.

Let t be a term in $Ter(\Sigma \cup X)$, p be a position in $Pos(t)$, and $l \to r$ be a rewriting rule in $\mathfrak{R}$. We say that $l \to r$ can be applied to t at position p if there is a substitution $\sigma : X \to Ter(\Sigma \cup X)$ such that $t|_p = l^\sigma$. In this case, we let $t' = t[r^\sigma]_p$ be the term that is obtained from t by the application of the rewriting rule $l \to r$ at position p. We write $t \to_{\mathfrak{R}} t'$

to denote that t' can be obtained from t by the application of some rewriting rule $l \to r \in \mathfrak{R}$ at some position p of t. We say that $\to_{\mathfrak{R}}$ is the relation induced by $\mathfrak{R}$ on $Ter(\Sigma \cup X)$. We let $\to^*_{\mathfrak{R}}$ be the transitive closure of $\to_{\mathfrak{R}}$. In other words, $t \to^*_{\mathfrak{R}} t'$ if and only if t' is obtained from t by the application of a finite number of rewriting rules from $\mathfrak{R}$.

2.3 Tree Automata

Let Q be a finite set of symbols of arity 0 called states. The elements of the set $Ter(\Sigma \cup Q)$ are called *configurations*. A *transition* is a rewriting rule of the form $f(q_1, ..., q_{\mathfrak{a}(f)}) \to q$ for some function symbol $f \in \Sigma$ and states $q_1, ..., q_{\mathfrak{a}(f)}, q \in Q$. A *(bottom-up non-deterministic) finite tree-automaton* over Σ is a tuple $\mathcal{A} = (Q, \Sigma, F, \Delta)$ where $F \subseteq Q$ is a set of final states and δ is a set of transitions. Note that Δ should be regarded as a term rewriting system acting on terms in $Ter(\Sigma \cup Q)$. We may write $\to_{\mathcal{A}}$ to denote the rewriting relation induced by the transitions Δ. Analogously, we may write $\to^*_{\mathcal{A}}$ to denote $\to^*_{\Delta}$. The tree language recognized by a state q in $\mathcal{A}$ is defined as

$$\mathcal{L}(\mathcal{A}, q) = \{t \in Ter(\Sigma) \mid t \to^*_{\mathcal{A}} q\}.$$

Intuitively, $\mathcal{L}(\mathcal{A}, q)$ is the set of all ground terms in $Ter(\Sigma)$ that can be reduced to the state q by the application of transitions (rewriting rules) in Δ. The tree language accepted by $\mathcal{A}$ is defined as $\mathcal{L}(\mathcal{A}) = \bigcup_{q \in F} \mathcal{L}(\mathcal{A}, q)$. As an abuse of notation we will often write $q \in \mathcal{A}$ and $t \to q \in \mathcal{A}$ to denote respectively that $q \in Q$ and $t \to q \in \Delta$.

The size of $\mathcal{A}$, which is defined as $|\mathcal{A}| = |Q| + |\Delta|$, measures the number of states in Q plus the number of transitions in Δ. If $f(q_1, ..., q_{\mathfrak{a}(f)}) \to q$ is a transition in Δ, then we say that q is the consequent of $f(q_1, ..., q_{\mathfrak{a}(f)}) \to q$, while each state in $\{q_1, ..., q_{\mathfrak{a}(f)}\}$ is an antecedent of $f(q_1, ..., q_{\mathfrak{a}(f)}) \to q$. We say that q is incident with a transition if it is either an antecedent or a consequent of the transition. The *in-degree* of a state q in Q, denoted by $\delta(q)$ is the number of transitions in Δ that have q as a consequent. The *maximum state in-degree* of $\mathcal{A}$, defined as $\delta(\mathcal{A}) = \max_{q \in Q} \delta(q)$, is the maximum in-degree of a state in $\mathcal{A}$. We say that $\mathcal{A}$ is reachable if for each state $q \in Q$ the language $\mathcal{L}(\mathcal{A}, q)$ is non-empty.

▶ **Lemma 2** (Membership [10]). *Let $\mathcal{A}$ be a tree automaton over Σ, and let t be a ground term in $Ter(\Sigma)$. One can determine in time $O(|t| \cdot |\mathcal{A}|)$ whether $t \in \mathcal{L}(\mathcal{A})$.*

▶ **Lemma 3** (Emptiness of Intersection [10]). *Let $\mathcal{A}$ and $\mathcal{A}'$ be two tree automata over Σ. One can determine in time $O(|\mathcal{A}| \cdot |\mathcal{A}'|)$ whether $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}') = \emptyset$.*

2.4 Simultaneous Rewriting via Multi-Steps

In this work we will be interested in a notion of rewriting that allows for the simultaneous application of several rules to a term as long as these rules do not interfere with each other. Such a notion of simultaneous rewriting can be formalized via the notion of *multi-step* [19]. A more detailed treatment rewriting by multi-steps can be found in [4] (Chapter 4). When restricted to the setting of rewriting on ground terms the notion of multi-step can be formalized as in Definition 4.

▶ **Definition 4** (Multi-Step). Let $\mathfrak{R}$ be a term rewriting system. The multi-step relation $\multimap \subseteq Ter(\Sigma) \times Ter(\Sigma)$ induced by $\mathfrak{R}$ is inductively defined as follows.

1. $f(t_1, ..., t_{\mathfrak{a}(f)}) \multimap f(t'_1, ..., t'_{\mathfrak{a}(f)})$ if $f \in \Sigma$ and $t_i \multimap t'_i$ for each $i \in \{1, ..., \mathfrak{a}(f)\}$.
2. $l^\sigma \multimap r^\theta$ if $l \to r \in \mathfrak{R}$, and $\sigma, \theta : X \to Ter(\Sigma)$ are substitutions such that $\sigma(x) \multimap \theta(x)$ for each variable $x \in var(l)$.

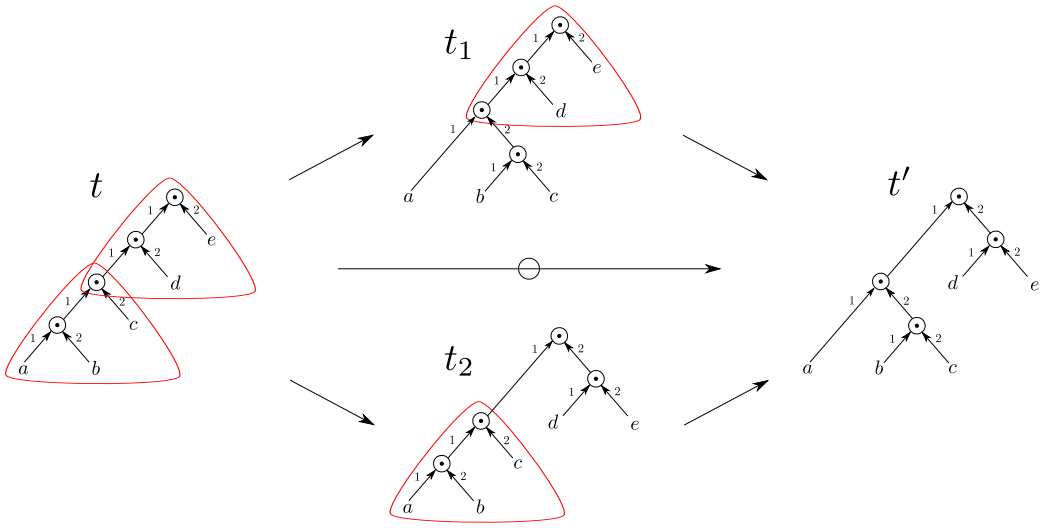


Figure 1 The application of a multi-step $t \multimap t'$ where $t = (((a \cdot b) \cdot c) \cdot d) \cdot e$ and $t' = (a \cdot (b \cdot c)) \cdot (d \cdot e)$. This multi-step $t \multimap t'$ intuitively corresponds to the simultaneous application of two instances of the associativity rule $(x \cdot y) \cdot z \to x \cdot (y \cdot z)$ to t. The regions surrounded by red curves indicate the portions of the term identifying a match for the left-hand side of the rule. The multistep can be decomposed in two ways as a sequence of individual rewriting rules: either as $t \to_{\mathfrak{R}} t_1 \to_{\mathfrak{R}} t'$ or as $t \to_{\mathfrak{R}} t_2 \to_{\mathfrak{R}} t'$.

Note that the base case of the inductive definition is embedded in Condition 1. More precisely, for each function symbol $f \in \Sigma$ of arity 0, we have that $f \multimap f$. We note that a multi-step $t \multimap t'$ may be decomposed in several different ways as sequences of applications of individual rewriting rules from $\mathfrak{R}$. For instance, in Figure 1 we depict the application of a multi-step to a term t. Intuitively, this multi-step $t \multimap t'$ corresponds to the simultaneous application of two instances of the associativity rule $(x \cdot y) \cdot z \to x \cdot (y \cdot z)$. There are two different ways of decomposing $t \multimap t'$ into sequences of individual rewriting rules: $t \to_{\mathfrak{R}} t_1 \to_{\mathfrak{R}} t'$ and $t \to_{\mathfrak{R}} t_2 \to_{\mathfrak{R}} t'$.

We say that a term t' is derived from a term t in depth at most d if there is a sequence of multi-steps $t_0 \multimap t_1 \multimap \ldots \multimap t_d$ such that $t_0 = t$ and $t_d = t'$. We write $t \overset{d}{\multimap} t'$ to denote that t' can be obtained from t in depth at most d.

3 Tree Automata Completion for Multi-Steps

Tree Automata completion is a powerful set of techniques which has found many applications in the field of termination analysis of rewriting systems [9, 11, 17, 8]. In this section we show that completion techniques, which have been so far used only in the context of sequential term rewriting, can be used to characterize derivability in one multi-step on linear term rewriting systems. More precisely, given a tree-automaton $\mathcal{A}$ and a linear term rewriting system $\mathfrak{R}$, we use a special instance of the tree-automata completion algorithm introduced in [9] in the context of sequential rewriting to construct a particular tree-automaton $\mathcal{N}(\mathcal{A}, \mathfrak{R})$. Subsequently, in Lemma 7 we show that the language accepted by $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ consists precisely of the set of terms that can be obtained from terms in $\mathcal{L}(\mathcal{A})$ by the application of one multi-step. A crucial aspect of our construction is that the size of the tree automaton $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ is upper bounded by $g(\mathfrak{R}, \delta) \cdot |\mathcal{A}|$ where $g(\mathfrak{R}, \delta)$ is a function that depends only on the term rewriting system $\mathfrak{R}$ and on the maximum state in-degree δ of $\mathcal{A}$. In other words,

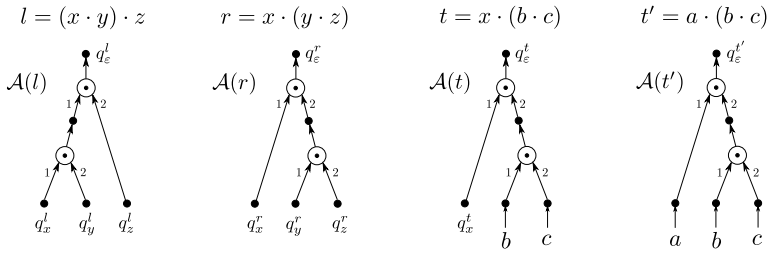


Figure 2 Graphical representation of the tree automata associated with terms l, r, t and t' respectively. The symbols x, y and z are variables. The symbol $\odot$ is a function symbol of arity 2, and a, b, c are constants (function symbols of arity 0). States are denoted by black dots.

the size of $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ grows linearly with the size of $\mathcal{A}$. This linear growth will be crucial for our fixed parameter tractability results.

▶ **Definition 5** (Tree Automaton Associated with a Term)**.** Let t be a term in $Ter(\Sigma \cup X)$. The tree automaton associated with t, denoted by $\mathcal{A}(t) = (Q, \Sigma, F, \Delta)$ is defined as follows.

$$Q = \{q^t_p \mid p \in Pos(t)\} \qquad F = \{q^t_\varepsilon\}$$
$$\Delta = \{f(q^t_{p.1}, ..., q^t_{p.\mathtt{a}(f)}) \to q^t_p \mid t(p) = f\} \tag{1}$$

Intuitively, if t is a ground term, then $\mathcal{A}(t)$ is the 'simplest' (but not necessarily minimal) tree automaton that accepts t and no other term. On the other hand, if t has some variable then the language of $\mathcal{A}(t)$ is empty. Nevertheless, this is not relevant, since in this case these tree-automata will be glued to other tree-automata in order to define a meaningful tree-language. In Figure 2 we depict tree-automata associated with several terms with and without variables.

Let $\mathcal{A} = (Q, \Sigma, F, \Delta)$ be a tree automaton and let l be a term in $Ter(\Sigma \cup X)$. A *state-substitution* for l is a function $\gamma : var(l) \to Q$ that associates with each variable $x \in var(l)$ a state $\gamma(x) \in Q$. Note that the term l^γ obtained from l by replacing each variable x with the state $\gamma(x)$ is a configuration in $Ter(\Sigma \cup Q)$. Also note that if l is a ground term in $Ter(\Sigma)$, then the only state-substitution for l is the empty function $\gamma :\to Q$. In this case, $l^\gamma = l$. We say that a state-substitution $\gamma : var(l) \to Q$ is good for a pair (q, l) if $l^\gamma \to_\Delta q$. In other words, γ is good for (q, l) if the configuration l^γ can be reduced to state q by the application of transitions in Δ. We let $\mathcal{M}(\mathcal{A}, q, l)$ be the set good state-substitutions for (q, l).

Let $l \to r$ be a linear term rewriting rule over Σ. We let q^l_ε denote the unique accepting state of $\mathcal{A}(l)$, and q^r_ε denote the unique accepting state of $\mathcal{A}(r)$. Additionally, for each variable $x \in var(l) \cap var(r)$, we let q^l_x denote the unique state of $\mathcal{A}(l)$ corresponding to the variable x, and we let q^r_x denote the unique state of $\mathcal{A}(r)$ corresponding to the variable x. Now let $\mathcal{A}$ be a tree automaton over Σ, and $\gamma : var(l) \to Q$ be a state-substitution in $\mathcal{M}(\mathcal{A}, q, l)$. We denote by $\mathcal{C}(\mathcal{A}, q, l \to r, \gamma)$ the tree automaton which is obtained by creating a fresh copy of $\mathcal{A}(r)$, and by renaming the minimal and maximal states of $\mathcal{A}(r)$ as follows.

1. Rename the state q^r_ε in $\mathcal{A}(r)$ to the state q.
2. For each variable $x \in var(l) \cap var(r)$, rename the state q^r_x of $\mathcal{A}(r)$ to the state $\gamma(x)$.

Now consider the tree automaton $\mathcal{A} \cup \mathcal{C}(\mathcal{A}, q, l \to r, \gamma)$. Intuitively, this tree automaton is obtained by creating a copy of $\mathcal{A}(r)$ and subsequently, by identifying its accepting state q^r_ε with the state q of $\mathcal{A}$, and by identifying, for each variable $x \in var(l) \cap var(r)$, the state q^r_x with the state $\gamma(x)$. This process is illustrated in Figure 3.

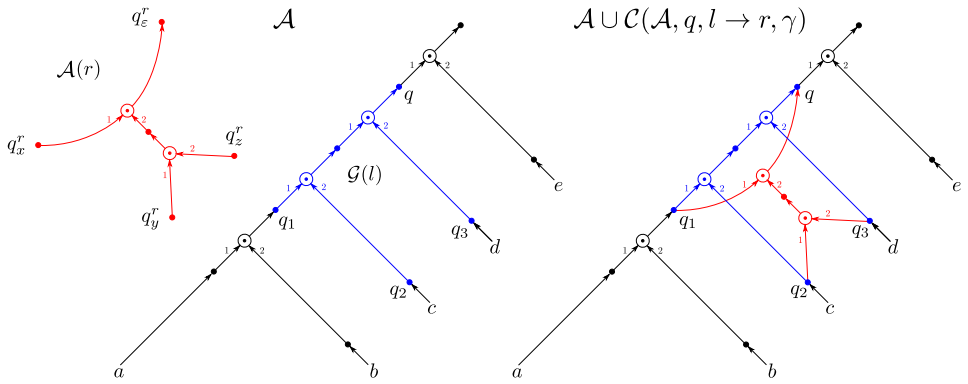


Figure 3 Let $l = (x \cdot y) \cdot z$ and $r = x \cdot (y \cdot z)$. Left: The disjoint union of a tree automaton $\mathcal{A}$ with the tree automaton $\mathcal{A}(r)$. The state-substitution γ maps the state q_x^l to q_1 the state q_y^l to q_2 and the state q_z^l to q_3. The portion of $\mathcal{A}$ in blue shows that the configuration l^γ can be reduced to the state q using transitions of $\mathcal{A}$. Right: The tree automaton $\mathcal{A} \cup \mathcal{C}(\mathcal{A}, \gamma, l \to r)$ obtained by identifying q_ε^r with q, q_x^r with q_1, q_y^r with q_2, and q_z^r with q_3. The tree automaton $\mathcal{C}(\mathcal{A}, q, l \to r, \gamma)$ is illustrated in red.

▶ **Definition 6** (Next Layer Operator). Let $\mathfrak{R}$ be a linear term rewriting system and $\mathcal{A}$ be a reachable tree automaton. The tree automaton $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ is defined as follows.

$$\mathcal{N}(\mathcal{A}, \mathfrak{R}) \quad = \quad \mathcal{A} \quad \cup \quad \bigcup_{\substack{q \in Q, l \to r \in \mathfrak{R} \\ \gamma \in \mathcal{M}(\mathcal{A}, q, l)}} \mathcal{C}(\mathcal{A}, q, l \to r, \gamma). \tag{2}$$

We say that each sub-automaton $\mathcal{C}(\mathcal{A}, q, l \to r, \gamma)$ of $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ is a *right-component* of $\mathcal{N}(\mathcal{A}, \mathfrak{R})$. Intuitively, $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ may be regarded as being constructed by adding one right component $\mathcal{C}(\mathcal{A}, q, l \to r, \gamma)$ to $\mathcal{A}$ at a time, in any desired order (See Figure 4).

The next lemma states that $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ accepts precisely those terms that can be reached from terms in $\mathcal{L}(\mathcal{A})$ by the application of one multi-step.

▶ **Lemma 7.** *Let $\mathfrak{R}$ be a linear term rewriting system, and let $\mathcal{A}$ be a tree automaton over Σ. The tree automaton $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ accepts the following language.*

$$\mathcal{L}(\mathcal{N}(\mathcal{A}, \mathfrak{R})) = \{t' \mid \exists t \in \mathcal{L}(\mathcal{A}),\ t \multimap t'\}. \tag{3}$$

Proof. Let $\mathcal{A} = (Q, \Sigma, F, \Delta)$ and $\mathcal{N}(\mathcal{A}, \mathfrak{R}) = (Q \cup Q', \Sigma, F, \Delta \cup \Delta')$ where $Q \cap Q' = \emptyset$ and $\Delta \cap \Delta' = \emptyset$. Note that the final states of $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ are the same as those of $\mathcal{A}$.

Completeness Proof

First we show that $\mathcal{L}(\mathcal{N}(\mathcal{A}, \mathfrak{R})) \supseteq \{t' \mid \exists t \in \mathcal{L}(\mathcal{A}),\ t \multimap t'\}$. It is enough to show that for each state $q \in Q$ if $t \in \mathcal{L}(\mathcal{A}, q)$ and $t \multimap t'$ then $t' \in \mathcal{L}(\mathcal{N}(\mathcal{A}, \mathfrak{R}), q)$. The proof of this claim is by induction on the structure of t, using Definition 4.

In the base case of Condition 1 of Definition 4, $t = a$ where a is a function symbol of arity 0 and $t' = a$. In this case the claim follows trivially. In the base case of Condition 2 of Definition 4, both t and t' are ground terms and $t \to t'$ belongs to $\mathfrak{R}$. Let $\gamma : \emptyset \to Q$ be the empty substitution. Since the term t' reaches the state q in $\mathcal{C}(\mathcal{A}, q, a \to t', \gamma)$ we have that $t' \in \mathcal{L}(\mathcal{N}(\mathcal{A}, \mathfrak{R}), q)$.

For the inductive step of Condition 1, let $t = f(t_1, ..., t_{\mathfrak{a}(f)})$ and $t' = f(t'_1, ..., t'_{\mathfrak{a}(f)})$ where $t_i \multimap t'_i$ for each $i \in \{1, ..., \mathfrak{a}(f)\}$. Since $t \in \mathcal{L}(\mathcal{A}, q)$, there exists a transition

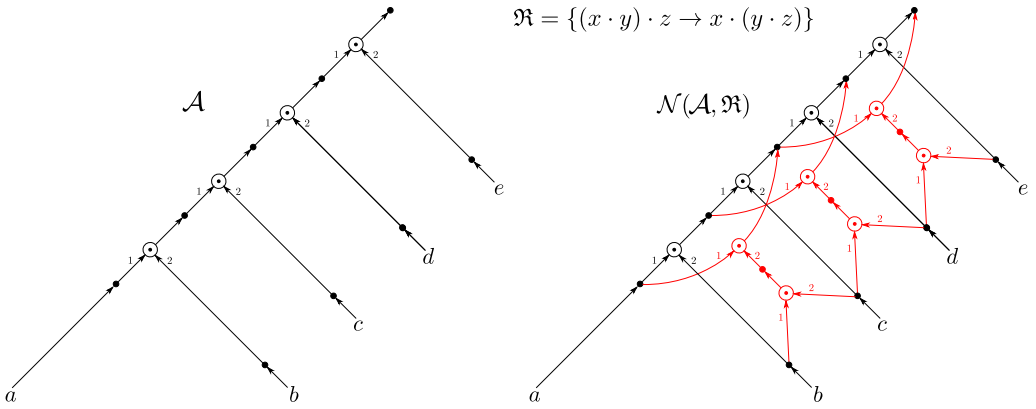


Figure 4 Left: A tree automaton $\mathcal{A}$ which recognizes the language $\mathcal{L}(\mathcal{A}) = \{(((a \cdot b) \cdot c) \cdot d) \cdot e\}$. Right: The tree automaton $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ which recognizes the language $\mathcal{L}(\mathcal{N}(\mathcal{A}, \mathfrak{R})) = \{(((a \cdot b) \cdot c) \cdot d) \cdot e, ((a \cdot (b \cdot c)) \cdot d) \cdot e, ((a \cdot b) \cdot (c \cdot d)) \cdot e, ((a \cdot b) \cdot c) \cdot (d \cdot e), (a \cdot (b \cdot c)) \cdot (d \cdot e)\}$. The three right components of $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ are depicted in red.

$(q_1, ..., q_{\mathfrak{a}(f)}, f) \to q$ in $\mathcal{A}$ such that t_i reaches q_i for each $i \in \{1, ..., \mathfrak{a}(f)\}$. By the induction hypothesis, since $t_i \multimap\!\!\to t'_i$, we have that $t'_i \in \mathcal{L}(\mathcal{N}(\mathcal{A}, \mathfrak{R}), q_i)$. Therefore, $f(t'_1, ..., t'_{\mathfrak{a}(f)}) \in \mathcal{L}(\mathcal{N}(\mathcal{A}, \mathfrak{R}), q)$.

For the inductive step of Condition 2, let $t = l^\sigma$ for some substitution $\sigma : X \to Ter(\Sigma)$, and let $t' = r^\theta$ where for each variable $x \in var(l)$, $\sigma(x) \multimap\!\!\to \theta(x)$. Since $t \in \mathcal{L}(\mathcal{A}, q)$, there exists a state substitution $\gamma : var(l) \to Q$ such that $l^\gamma \to^*_\Delta q$. By the induction hypothesis, $\theta(x) \in \mathcal{L}(\mathcal{N}(\mathcal{A}, \mathfrak{R}), \gamma(x))$ for each $x \in var(l)$. Additionally, $r^\gamma \to^*_{\Delta'} q$ in the right component $\mathcal{C}(\mathcal{A}, q, l \to r, \gamma)$. This implies that $t' \in \mathcal{L}(\mathcal{N}(\mathcal{A}, \mathfrak{R}), q)$.

Soundness Proof

Now we show that $\mathcal{L}(\mathcal{N}(\mathcal{A}, \mathfrak{R})) \subseteq \{t' \mid \exists t \in \mathcal{L}(\mathcal{A}),\ t \multimap\!\!\to t'\}$. It is enough to show that for each state q of $\mathcal{A}$, if $t' \in \mathcal{L}(\mathcal{N}(\mathcal{A}, \mathfrak{R}), q)$ then there is a ground term $t \in \mathcal{L}(\mathcal{A}, q)$ such that $t \multimap\!\!\to t'$. The proof is by well founded induction with terms ordered by the strict subterm relation.

Let $q \in Q$ and let $t \in \mathcal{L}(\mathcal{N}(\mathcal{A}, \mathfrak{R}), q)$. In the base case, let $t' = a$ for some function symbol a of arity 0. Let $a \to q$ be a transition in $\mathcal{N}(\mathcal{A}, \mathfrak{R})$. If $a \to q$ belongs to Δ then $a \in \mathcal{L}(\mathcal{A}, q)$ and additionally, by the base case of Condition 1 of Definition 4, we have that $a \multimap\!\!\to a$. If $a \to q$ belongs to Δ', then there is some rewriting rule $l \to r$ in $\mathfrak{R}$ and some state-substitution $\gamma : var(l) \to Q$ such that $a \to q$ is a transition in $\mathcal{C}(\mathcal{A}, q, l \to r, \gamma)$. Let $\sigma : var(l) \to Ter(\Sigma)$ be a substitution that associates with each variable $x \in var(l)$ an arbitrary term in $\mathcal{L}(\mathcal{A}, \gamma(x))$. Note that such term is guaranteed to exist, since by assumption $\mathcal{A}$ is reachable. Then the term l^σ belongs to $\mathcal{L}(\mathcal{A}, q)$. Additionally, by Condition 2 of Definition 4, we have that $l^\sigma \multimap\!\!\to a^\tau$ where $\tau : \emptyset \to Ter(\Sigma)$ is the empty substitution. This verifies the claim in the base case.

Now let $t' = f(t'_1, ..., t'_{\mathfrak{a}(f)})$ where f has arity at least 1. Then there exists a transition $f(q_1, ..., q_{\mathfrak{a}(f)}) \to q$ in $\Delta \cup \Delta'$ such that $t'_i \in \mathcal{L}(\mathcal{N}(\mathcal{A}), q_i)$ for each $i \in \{1, ..., \mathfrak{a}(f)\}$. There are two cases to be analysed.

1. If $f(q_1, ..., q_{\mathfrak{a}(f)}) \to q$ belongs to Δ then all states $q_1, ..., q_{\mathfrak{a}(f)}$ belong to Q. In this case, by the induction hypothesis, there exists terms $t_1, ..., t_{\mathfrak{a}(f)}$ such that $t_i \multimap\!\!\to t'_i$ and $t_i \in \mathcal{L}(\mathcal{A}, q_i)$ for each $i \in \{1, ..., \mathfrak{a}(f)\}$. This implies that the term $t = f(t_1, ..., t_{\mathfrak{a}(f)})$ is in $\mathcal{L}(\mathcal{A}, q)$ and additionally, by Condition 1 of Definition 4, we have that $t \multimap\!\!\to t'$.

2. If $f(q_1, ..., q_{\mathfrak{a}(f)}) \to q$ belongs to Δ' then the states $q_1, ..., q_{\mathfrak{a}(f)}$ belong to some right component of $\mathcal{N}(\mathcal{A}, \mathfrak{R})$. In other words, there is some rewriting rule $l \to r$ in $\mathfrak{R}$, some substitution $\theta : var(r) \to Ter(\Sigma)$ and some state-substitution $\gamma : var(l) \to Q$ such that $t' = r^\theta$, $l^\gamma \to^*_\Delta q$ and $r^\gamma \to^*_{\Delta'} q$ and such that $\theta(x) \in \mathcal{L}(\mathcal{N}(\mathcal{A}, \mathfrak{R}), \gamma(x))$ for each $x \in var(r)$. By the induction hypothesis, for each $x \in var(r) \subseteq var(l)$ there is a term s_x which belongs to $\mathcal{L}(\mathcal{A}, \gamma(x))$ and $s_x \multimap\!\!\to \theta(x)$. Additionally, since $\mathcal{A}$ is reachable, for each variable $y \in var(l) \backslash var(r)$ there is at least one term s_y in $\mathcal{L}(\mathcal{A}, \gamma(y))$. Let $\sigma : var(l) \to Ter(\Sigma)$ be a substitution that sets $\sigma(x) = s_x$ for each variable $x \in var(r)$, and which sets $\sigma(y) = s_y$ for each variable $y \in var(l) \backslash var(r)$. Then the term l^σ belongs to $\mathcal{L}(\mathcal{A}, q)$ and additionally, by Condition 2 of Definition 4, $l^\sigma \multimap\!\!\to r^\theta$. ◀

4 Bounding the Size of $\mathcal{N}(\mathcal{A}, \mathfrak{R})$

In this section we will establish an upper bound for the size of the tree automaton $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ in terms of the size of $\mathcal{A}$, the maximum state in-degree of $\mathcal{A}$, and several parameters extracted from the term rewriting system $\mathfrak{R}$. Subsequently, we will use Lemma 7, together with the size upper bound mentioned above to establish the fixed parameter tractability of reachability and joinability in depth d.

A morphism from a tree automaton $\mathcal{A} = (Q, \Sigma, F, \Delta)$ to a tree automaton $\mathcal{A}' = (Q', \Sigma, F', \Delta')$ is a function $\mu : Q \to Q'$ such that for each transition $f(q_1, ..., q_{\mathfrak{a}(f)}) \to q$ in Δ, the transition $f(\mu(q_1), ..., \mu(q_{\mathfrak{a}(f)})) \to \mu(q)$ is in Δ'. As an abuse of notation we write $\mu : \mathcal{A} \to \mathcal{A}'$ to denote such a morphism.

Let $l \in Ter(\Sigma \cup X)$ and $\mathcal{A}(l)$ be the tree automaton associated with l. We say that a morphism $\mu : \mathcal{A}(l) \to \mathcal{A}$ from $\mathcal{A}(l)$ to $\mathcal{A}$ is rooted at a state $q \in \mathcal{A}$ if $\mu(q^l_\varepsilon) = q$ where q^l_ε is the unique maximal state of $\mathcal{A}(l)$. Each such morphism μ defines a state-substitution $\gamma : var(l) \to Q$ which is defined by setting $\gamma(x) = \mu(q^l_x)$ for each variable $x \in var(l)$. Intuitively, the morphism μ specifies a run of the automaton $\mathcal{A}$ (i.e. a sequence of transitions from $\mathcal{A}$) which are applied to reduce the configuration l^γ to the state q.

For each tree automaton $\mathcal{A}$, each state q of $\mathcal{A}$, and each positive integer s, we denote by $\eta(\mathcal{A}, q, s)$ the set of all pairs (l, μ) where l is a term in $Ter(\Sigma \cup X)$ of size at most s, and μ is a morphism from $\mathcal{A}(l)$ to $\mathcal{A}$ rooted at q.

In the following lemma we show an upper bound on the size of $\eta(\mathcal{A}, q, s)$ in terms of the maximum state in-degree of $\mathcal{A}$.

▶ **Lemma 8.** *Let $\mathcal{A}$ be a tree-automaton over Σ of maximum state in-degree δ, q be a state of $\mathcal{A}$, and s be a positive integer. Let $\mathfrak{a}$ be the maximum arity of a function symbol in Σ. Then $|\eta(\mathcal{A}, q, s)| \leq (e \cdot \max\{\delta, \mathfrak{a}\})^{2s+1}$.*

Proof. Let $T(\delta)$ be the rooted infinite δ-regular tree. Let $b_{k,\delta}$ be the number of rooted subtrees of $T(\delta)$ of size k. It is well known that $b_{k,\delta} \leq \binom{\delta \cdot k}{k}$ (See for instance [6, 3]). Using the inequality $\binom{n}{k} \leq \left(\frac{e \cdot n}{k}\right)^k$ (where $e \approx 2.71$ is the Euler number) we have that $b_{k,\delta} \leq (e \cdot \delta)^k$. This implies that the number $c_{k,\delta}$ of rooted subtrees of $T(\delta)$ of size *at most* k is upper bounded by $c_{k,\delta} \leq (e \cdot \delta)^{k+1}$.

Now let $\mathrm{U}(\mathcal{A}, q)$ be the unfolding of $\mathcal{A}$ rooted at q. Since the maximum in-degree of $\mathcal{A}$ is δ, we have that U is a rooted infinite tree of degree at most $\max\{\mathfrak{a}, \delta\}$. Additionally, if l is a term in $Ter(\Sigma \cup X)$ of size s, then for each pair $(l, \mu) \in \eta(\mathcal{A}, q, s)$ the image of $\mathcal{A}(l)$ under μ on $\mathcal{A}$ corresponds unequivocally to a rooted subtree of $\mathrm{U}(\mathcal{A}, q)$ of size at most $2s$. Therefore, the number of possible morphisms rooted at q from $\mathcal{A}(t)$ to $\mathcal{A}$ where t is a term of size s is upper bounded by the number of rooted subtrees of $\mathrm{U}(\mathcal{A}, q)$ of size at most $2s$. This implies that $|\eta(\mathcal{A}, q, s)| \leq (e \cdot \delta)^{2s+1}$ if $\delta \geq \mathfrak{a}$, and $|\eta(\mathcal{A}, q, s)| \leq (e \cdot \mathfrak{a})^{2s+1}$ if $\mathfrak{a} \geq \delta$. ◀

The next lemma establishes an upper-bound for the size and for the state in-degree of $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ in terms of the size of $\mathcal{A}$, state in-degree of $\mathcal{A}$, and several parameters extracted from the term rewriting system $\mathfrak{R}$.

▶ **Lemma 9.** *Let $\mathcal{A}$ be a tree automaton of maximum in-degree δ. Let $\mathfrak{R}$ be a linear term rewriting system. Let s_1 be the maximum size of the left-hand side of a rule in $\mathfrak{R}$, and s_2 be the maximum size of a right-hand side of a rule in $\mathfrak{R}$. Let ρ be the maximum number of rules in $\mathfrak{R}$ with the same left-hand side.*

1. *The maximum in-degree of $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ is at most $\delta + \rho \cdot (e \cdot \max\{\mathfrak{a}, \delta\})^{2s_1+1}$.*
2. *The size of $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ is at most $|\mathcal{A}| + 2 \cdot s_2 \cdot \rho \cdot (e \cdot \max\{\mathfrak{a}, \delta\})^{2s_1+1} \cdot |\mathcal{A}|$.*
3. *$\mathcal{N}(\mathcal{A}, \mathfrak{R})$ can be constructed in time $O\left(s_2 \cdot \rho \cdot (e \cdot \max\{\mathfrak{a}, \delta\})^{2s_1+1} \cdot (\log |\mathfrak{R}|) \cdot |\mathcal{A}|\right)$.*

Proof.

1. Let q be a state of $\mathcal{A}$. The in-degree of q in $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ is equal to the in-degree of q in $\mathcal{A}$ plus the number of right-components $\mathcal{C}(\mathcal{A}, q, l \to r, \gamma)$ of $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ rooted at q, where $l \to r$ is a rule in $\mathfrak{R}$ and γ is a good state-substitution for (q, l). By Lemma 8 there is at most $(e \cdot \max\{\mathfrak{a}, \delta\})^{2s_1+1}$ pairs of the form (l, μ) where where l is a term of size at most s_1 and $\mu : \mathcal{A}(l) \to \mathcal{A}$ is a morphism from $\mathcal{A}(l)$ to $\mathcal{A}$ rooted at q. Therefore the number of components $\mathcal{C}(\mathcal{A}, q, l \to r, \gamma)$ is upper bounded by $(e \cdot \max\{\mathfrak{a}, \delta\})^{2s_1+1}$. Since the number of rules with same left-hand side is upper bounded by ρ, we have that each pair $(l, \mu) \in \eta(\mathcal{A}, q, s_1)$ gives rise to at most ρ right components rooted at q. Therefore, the in-degree of q in $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ is at most $\delta + \rho \cdot (e \cdot \max\{\mathfrak{a}, \delta\})^{2s_1+1}$.
2. As argued in the previous item, the number of right components of $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ rooted at q is upper bounded by $\rho \cdot (e \cdot \max\{\mathfrak{a}, \delta\})^{2s_1+1}$. Since each right-component $\mathcal{C}(\mathcal{A}, \gamma, l \to r)$ is isomorphic to $\mathcal{A}(r)$, we have that such component has size at most $2 \cdot |r| \leq 2 \cdot s_2$. Therefore, the size of $\mathcal{N}(\mathcal{A}, \mathfrak{R})$ is upper bounded by $|\mathcal{A}| + 2 \cdot s_2 \cdot \rho \cdot (e \cdot \max\{\mathfrak{a}, \delta\})^{2s_1+1} \cdot |\mathcal{A}|$.
3. Assume that $\mathfrak{R}$ is specified as a lexicographically ordered list of rules. For each state q of $\mathcal{A}$, we can enumerate in time $O((e \cdot \max\{\mathfrak{a}, \delta\})^{2s_1+1})$ the set of all pairs (l, γ) where $l \in \mathit{Ter}(\Sigma \cup X)$ and γ is a morphism from $\mathcal{A}(l)$ to $\mathcal{A}$. For each of these pairs, we use binary search to look up for the existence of a rule in $\mathfrak{R}$ having l as left-hand side. Each such look up takes time $O(\log |\mathfrak{R}|)$. Finally, for each rule $l \to r$ in $\mathfrak{R}$, we create the right component $\mathcal{C}(\mathcal{A}, q, l \to r, \gamma)$. The addition of each such component takes time $O(|r|) \leq O(s_2)$. Since there are at most ρ rules with right-hand side l, the total amount of time to construct the automaton is $O\left(s_2 \cdot \rho \cdot (e \cdot \max\{\mathfrak{a}, \delta\})^{2s_1+1} \cdot (\log |\mathfrak{R}|) \cdot |\mathcal{A}|\right)$. ◀

Now let $t \in \mathit{Ter}(\Sigma)$ be a ground term over Σ, and let $\mathfrak{R}$ be a linear term rewriting system. Let $\mathcal{A}(t)$ be the tree automaton associated with t. For each $d \in \mathbb{N}$ we inductively define the tree automaton $\mathcal{A}(t, \mathfrak{R}, d)$ as follows.

$$\mathcal{A}(t, \mathfrak{R}, d) = \begin{cases} \mathcal{A}(t) \text{ if } d = 0 \\ \\ \mathcal{N}(\mathcal{A}(t, \mathfrak{R}, d-1), \mathfrak{R}) \qquad \text{if } d \geq 1. \end{cases} \tag{4}$$

Note that $\mathcal{L}(\mathcal{A}(t, \mathfrak{R}, 0)) = \mathcal{L}(\mathcal{A}(t)) = \{t\}$. Additionally, from Lemma 7 it follows straightforwardly by induction on d that $\mathcal{A}(t, \mathfrak{R}, d)$ is a tree automaton recognizing precisely the ground terms in $\mathit{Ter}(\Sigma)$ that can be reached from t in depth at most d. Let $\mathfrak{a}$ be the maximum arity of a function symbol in Σ. Then the maximum in-degree of a vertex of $\mathcal{A}(t, \mathfrak{R}, 0)$ is $\mathfrak{a}$, while the size of $\mathcal{A}(t, \mathfrak{R}, 0)$ is $2 \cdot |t|$. The next proposition establishes upper bounds for the size and maximum in-degree of the tree automaton $\mathcal{A}(t, \mathfrak{R}, d)$.

▶ **Proposition 10.** *Let $\mathfrak{a}$ be the maximum arity of a function symbol in Σ. Let s_1 be the maximum size of the left-side of a rule in $\mathfrak{R}$, s_2 be the maximum size of the right-side of a term in $\mathfrak{R}$, and ρ be the maximum number of rules in $\mathfrak{R}$ with the same left-hand side.*

- *The maximum in-degree of $\mathcal{A}(t, \mathfrak{R}, d)$ is at most $(e \cdot \rho \cdot \mathfrak{a})^{s_1^{2 \cdot d}}$.*
- *The size of $\mathcal{A}(t, \mathfrak{R}, d)$ is at most $s_2^d \cdot (e \cdot \rho \cdot \mathfrak{a})^{s_1^{2 \cdot d}} \cdot |t|$.*
- *$\mathcal{A}(t, \mathfrak{R}, d)$ can be constructed in time $s_2^d \cdot (e \cdot \rho \cdot \mathfrak{a})^{s_1^{2 \cdot d}} \cdot (\log |\mathfrak{R}|) \cdot |t|$.*

We omit the proof of Proposition 10 since it follows straightforwardly from Lemma 9 by induction on d. Finally, we are in a position to prove Theorem 1.

Proof of Theorem 1

1. **Reachability.** Let $f(\mathfrak{R}, d) = s_2^d \cdot (e \cdot \rho \cdot \mathfrak{a})^{s_1^{2d}} \cdot (\log |\mathfrak{R}|)$. Given two ground terms t and t' we want to determine whether t' can be derived from t in depth at most d. First, we construct in time $f(\mathfrak{R}, d) \cdot |t|$ the tree automaton $\mathcal{A}(t, \mathfrak{R}, d)$, whose size is at most $f(\mathfrak{R}, d) \cdot |t|$. Then we determine whether $\mathcal{A}(t, \mathfrak{R}, d)$ accepts the term t'. By Lemma 2 this membership test can be performed in time $f(\mathfrak{R}, d) \cdot |t| \cdot |t'|$.
2. **Joinability.** Given two ground terms t and t' we want to determine whether there exists a term u such that u can be derived both from t and from t' in depth at most d. First we construct the tree automata $\mathcal{A}(t, \mathfrak{R}, d)$ and $\mathcal{A}(t', \mathfrak{R}, d)$ whose sizes are respectively upper bounded by $f(\mathfrak{R}, d) \cdot |t|$ and $f(\mathfrak{R}, d) \cdot |t'|$. We have that there exists a term u that can be reached by both t and t' in depth at most d if and only if $\mathcal{A}(t, \mathfrak{R}, d) \cap \mathcal{A}(t', \mathfrak{R}, d)$ is non-empty. By Lemma 3, this emptiness of intersection test can be realized in time $f(\mathfrak{R}, d)^2 \cdot |t| \cdot |t'|$.

5 Conclusion

In this work we have shown that reachability and joinability in linear term rewriting systems are fixed parameter tractable with respect to the depth of derivations. More precisely, we showed that given a linear term rewriting system $\mathfrak{R}$, and ground terms t and t' one can determine in time $f(\mathfrak{R}, d) \cdot |t| \cdot |t'|$ whether t' is reachable from t in depth at most d, and in time $f(\mathfrak{R}, d)^2 \cdot |t| \cdot |t'|$ whether t and t' are joinable in depth at most d. We note that the function $f(\mathfrak{R}, d)$ depends double exponentially on d. We leave open the problem of determining whether the dependence on d in the function $f(\mathfrak{R}, d)$ can be substantially improved.

References

1 Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge university press, 1999.

2 Leo Bachmair. Rewrite techniques in theorem proving. In *Proc. of the 5th International Conference on Rewriting Techniques and Applications*, LNCS, pages 1–1, 1993.

3 Andrew Beveridge, Alan Frieze, and Colin McDiarmid. Random minimum length spanning trees in regular graphs. *Combinatorica*, 18(3):311–333, 1998.

4 Marc Bezem, Jan Willem Klop, and Roel de Vrijer. Terese. term rewriting systems. *Cambridge Tracts in Theoretical Computer Science*, 55, 2003.

5 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

6 Colin Cooper and Alan Frieze. Component structure of the vacant set induced by a random walk on a random graph. *Random Structures & Algorithms*, 42(2):135–158, 2013.

7 N. Dershowitz and J. P. Jouannaud. Rewrite systems. *Handbook of Theoretical Computer Science (Chapter 6)*, pages 243–320, 1990.

8 Bertram Felgenhauer and René Thiemann. Reachability analysis with state-compatible automata. In *Proc. of the 8th International Conference on Language and Automata Theory and Applications*, LNCS, pages 347–359. Springer, 2014.

9 Guillaume Feuillade, Thomas Genet, and Valérie Viet Triem Tong. Reachability analysis over term rewriting systems. *Journal of Automated Reasoning*, 33(3-4):341–383, 2004.

10 Ferenc Gécseg and Magnus Steinby. Tree languages. In *Handbook of formal languages*, pages 1–68. Springer, 1997.

11 Thomas Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proc. of the 9th International Conference on Rewriting Techniques and Applications*, LNCS, pages 151–165, 1998.

12 Rémy Gilleron and Sophie Tison. Regular tree languages and rewrite systems. *Fundamenta informaticae*, 24(1, 2):157–175, 1995.

13 Guillem Godoy, Robert Nieuwenhuis, and Ashish Tiwari. Classes of term rewrite systems with polynomial confluence problems. *ACM Transactions on Computational Logic (TOCL)*, 5(2):321–331, 2004.

14 Guillem Godoy, Ashish Tiwari, and Rakesh Verma. Characterizing confluence by rewrite closure and right ground term rewrite systems. *Applicable Algebra in Engineering, Communication and Computing*, 15(1):13–36, 2004.

15 Lukasz Kaiser. Confluence of right ground term rewriting systems is decidable. In *Foundations of Software Science and Computational Structures*, pages 470–489. Springer, 2005.

16 Jan Willem Klop, Marc Bezem, and RC De Vrijer. *Term rewriting systems*. Cambridge University Press, 2001.

17 Martin Korp and Aart Middeldorp. Match-bounds revisited. *Information and Computation*, 207(11):1259–1283, 2009.

18 Takashi Nagaya and Yoshihito Toyama. Decidability for left-linear growing term rewriting systems. In *Proc. of the 10th International Conference on Rewriting Techniques and Applications*, LNCS, pages 256–270, 1999.

19 Vincent van Oostrom. Normalisation in weakly orthogonal rewriting. In *Proc. of the 10th International Conference on Rewriting Techniques and Applications*, LNCS, pages 60–74, 1999.

20 Rakesh M Verma, Michael Rusinowitch, and Denis Lugiez. Algorithms and reductions for rewriting problems. *Fundamenta Informaticae*, 46(3):257–276, 2001.

Edge Bipartization Faster Than 2^{k*}

Marcin Pilipczuk[1], Michał Pilipczuk[2], and Marcin Wrochna[3]

1 **Institute of Informatics, University of Warsaw, Poland**
malcin@mimuw.edu.pl
2 **Institute of Informatics, University of Warsaw, Poland**
michal.pilipczuk@mimuw.edu.pl
3 **Institute of Informatics, University of Warsaw, Poland**
m.wrochna@mimuw.edu.pl

Abstract

In the Edge Bipartization problem one is given an undirected graph G and an integer k, and the question is whether k edges can be deleted from G so that it becomes bipartite. In 2006, Guo et al. [6] proposed an algorithm solving this problem in time $\mathcal{O}(2^k \cdot m^2)$; today, this algorithm is a textbook example of an application of the iterative compression technique. Despite extensive progress in the understanding of the parameterized complexity of graph separation problems in the recent years, no significant improvement upon this result has been yet reported.

We present an algorithm for Edge Bipartization that works in time $\mathcal{O}(1.977^k \cdot nm)$, which is the first algorithm with the running time dependence on the parameter better than 2^k. To this end, we combine the general iterative compression strategy of Guo et al. [6], the technique proposed by Wahlström [18] of using a polynomial-time solvable relaxation in the form of a Valued Constraint Satisfaction Problem to guide a bounded-depth branching algorithm, together with an involved Measure&Conquer analysis of the recursion tree.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases edge bipartization, FPT algorithm

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.26

1 Introduction

The Edge Bipartization problem asks, for a given graph G and integer k, whether one can turn G into a bipartite graph using at most k edge deletions. Together with its close relative Odd Cycle Transversal (OCT), where one deletes vertices instead of edges, Edge Bipartization was one of the first problems shown to admit a fixed-parameter (FPT) algorithm using the technique of *iterative compression*. In a breakthrough paper [17] that introduces this methodology, Reed et al. showed how to solve OCT in time $\mathcal{O}(3^k \cdot kmn)$[1]. In fact, this was the first FPT algorithm for OCT. Following this, Guo et al. [6] applied iterative compression to show fixed-parameter tractability of several closely related problems,

* Mi. Pilipczuk and M. Wrochna have been supported by the Polish National Science Centre grant DEC-2013/11/D/ST6/03073. Mi. Pilipczuk has been supported by Foundation for Polish Science via the START stipend program. During the work on these results, Mi. Pilipczuk has been holding a post-doc position of Warsaw Centre of Mathematics and Computer Science. Ma. Pilipczuk has been supported by the Centre for Discrete Mathematics and its Applications (DIMAP) at the University of Warwick and by Warwick-QMUL Alliance in Advances in Discrete Mathematics and its Applications.

[1] Even though Reed et al. [17] state their running time as $\mathcal{O}(4^k \cdot kmn)$, it is not hard to adjust the analysis to show that the algorithm in fact works in time $\mathcal{O}(3^k \cdot kmn)$; see e.g. [7, 15].

including an algorithm for Edge Bipartization with running time $\mathcal{O}(2^k \cdot m^2)$. Today, both results are textbook examples of the iterative compression technique.

Iterative compression is in fact a simple idea that boils down to an algorithmic usage of induction. In case of Edge Bipartization, we introduce edges of G one by one, and during this process we would like to maintain a solution F to the problem, i.e., $F \subseteq E(G)$ is such that $|F| \leq k$ and $G - F$ is bipartite. When the next edge e is introduced to the graph, we observe that $F \cup \{e\}$ is a solution of size at most $k + 1$, that is, at most one too large. Then the task reduces to solving Edge Bipartization Compression: given a solution that exceeds the budget by at most one, we are asked to find a solution that fits into the budget.

Surprisingly, this simple idea leads to great algorithmic gains, as it reduces the matter to a cut problem. Guo et al. [6] showed that a simple manipulation of the instance reduces Edge Bipartization Compression to the following problem that we call Terminal Separation: We are given an undirected graph G with a set $\mathcal{T}$ of $k + 1$ disjoint pairs of terminals, where each terminal is of degree 1 in G. The question is whether one can color one terminal of every pair white and the second black in such a way that the minimum edge cut between white and black terminals is at most k. Thus, the algorithm of Guo et al. [6] boils down to trying all the 2^{k+1} colorings of terminals and solving a minimum edge cut problem. For OCT, we similarly have a too large solution $X \subseteq V(G)$ of size $k + 1$, and we are looking for a partition of X into (L, R, Z), where the size of the minimum vertex cut between L and R in $G - Z$ is at most $k - |Z|$. Thus it suffices to solve 3^{k+1} instances of a flow problem.

The search for FPT algorithms for cut problems has been one of the leading directions in parameterized complexity in the recent years. Among these, Odd Cycle Transversal and Edge Bipartization play a central role; see for instance [6, 12, 14, 17] and references therein. Of particular importance is the work of Kratsch and Wahlström [12], who gave the first (randomized) polynomial kernelization algorithms for both problems. The main idea is to encode the cut problems that arise when applying iterative compression into a matroid with a representation that takes small space. The result sparked a line of further work on applying matroids in parameterized complexity.

Another thriving area in parameterized complexity is the *optimality program*, probably best defined by Marx in [16]. The goal of it is to systematically investigate the optimum complexity of algorithms for parameterized problem by proving possibly tight lower and upper bounds. For the lower bounds methodology, the standard complexity assumptions used are the *Exponential Time Hypothesis* (*ETH*) and the *Strong Exponential Time Hypothesis* (*SETH*). In the recent years, the optimality program has achieved a number of successes. For instance, under the assumption of SETH, we now know the precise bases of exponents for many classical problems parameterized by treewidth [13]. To explain the complexity of fundamental parameterized problems for which natural algorithms are based on dynamic programming on subsets, Cygan et al. [1] introduced a new hypothesis resembling SETH, called the *Set Cover Conjecture* (*SeCoCo*). See [13, 16] for more examples.

For our techniques, the most important is the line of work of Guillemot [5], Cygan et al. [3], Lokshtanov et al. [14], and Wahlström [18] that developed a technique for designing parameterized algorithm for cut problems called *LP-guided branching*. The idea is to use the optimum solution to the linear programming (LP) relaxation of the considered problem in order to measure progress. Namely, during the construction of a candidate solution by means of a backtracking process, the algorithm achieves progress not only when the budget for the size of the solution decreases (as is usual in branching algorithms), but also when the LP lower bound on the optimum solution increases. Using this concept, Cygan et al. [3] showed a $2^k n^{\mathcal{O}(1)}$-time algorithm for Node Multiway Cut. Lokshtanov et al. [14] further

refined this technique and applied it to improve the running times of algorithms for several important cut problems. In particular, they obtained a $2.315^k n^{\mathcal{O}(1)}$-time algorithm for Odd Cycle Transversal, which was the first improvement upon the classic $\mathcal{O}(3^k \cdot kmn)$-time algorithm of Reed et al. [17]. From the point of view of the optimality program, this showed that the base 3 of the exponent was not the final answer for Odd Cycle Transversal.

In [3, 14] it was essential that the considered LP relaxation is half-integral, which restricts the applicability of the technique. Recently, Wahlström [18] proposed to use stronger relaxations in the form of certain polynomial-time solvable Valued Constraint Satisfaction Problems (VCSPs). Using this idea, he showed efficient FPT algorithms for node and edge deletion variants of Unique Label Cover, for which natural LP relaxations are not half-integral.

Despite substantial progress on the node deletion variant, for Edge Bipartization there has been no improvement since the classic algorithm of Guo et al. [6] that runs in time $\mathcal{O}(2^k \cdot m^2)$. The main technical contribution of Lokshtanov et al. [14] is a $2.315^k n^{\mathcal{O}(1)}$-time algorithm for Vertex Cover parameterized by the excess above the value of the LP relaxation (VC-above-LP); the algorithm for OCT then follows from folklore reductions from OCT to VC-above-LP via the Almost 2-SAT problem. Thus the algorithm for OCT in fact relies on the LP relaxation for Vertex Cover, which has very strong combinatorial properties; in particular, it is half-integral. No such strong and simple relaxation is available for Edge Bipartization. The natural question stemming from the optimality program, whether the 2^k term for Edge Bipartization can be improved, was asked repeatedly in the parameterized complexity community, e.g. by Daniel Lokshtanov at WorKer'13 [2].

Our results and techniques. In this paper we answer this question in affirmative:

▶ **Theorem 1.1.** Edge Bipartization *can be solved in time* $\mathcal{O}(1.977^k \cdot nm)$.

To prove this, we begin with the approach of Guo et al. [6], using iterative compression to reduce solving Edge Bipartization to solving Terminal Separation (see Section 2 for a formal definition of the latter). This problem has two natural parameters: $|\mathcal{T}|$, the number of terminal pairs, and p, the bound on the size of the cut between white and black terminals. The approach of Guo et al. is to use a simple $\mathcal{O}(2^{|\mathcal{T}|} \cdot pm)$ algorithm that tries all colorings of terminal pairs and computes the size of a minimum cut between the colors.

The observation that is crucial to our approach is that one can express Terminal Separation as a very restricted instance of the Edge Unique Label Cover problem. More precisely, in this setting the task is to assign each vertex of G a label from $\{\mathbf{A}, \mathbf{B}\}$. Pairs of $\mathcal{T}$ present hard (of infinite cost) inequality constraints between the labels of terminals involved, while edges of G present soft (of unit cost) equality constraints between the endpoints. The goal is to minimize the cost of the labeling, i.e., the number of soft constraints broken. An application of the results of Wahlström [18] (with further improvements of Iwata, Wahlström, and Yoshida [8] regarding linear dependency on the input size) immediately gives an $\mathcal{O}(4^p \cdot m)$ algorithm for Terminal Separation.

Thus, we have in hand two substantially different algorithms for Terminal Separation. If we plug in $|\mathcal{T}| = k + 1$ and $p = k$, as is the case in the instance that we obtain from Edge Bipartization Compression, then we obtain running times $\mathcal{O}(2^k \cdot km)$ and $\mathcal{O}(4^k \cdot m)$, respectively. The idea now is that these two algorithms present two complementary approaches to the problem, and we would like to combine them to solve the problem more efficiently. To this end, we need to explain more about the approach of Wahlström [18].

The algorithm of Wahlström [18] is based on measuring the progress by means of the optimum solution to the relaxation of the problem (in the form of a Valued CSP instance).

In our case, this relaxation of Terminal Separation has the following form: We assign each vertex a label from $\{\bot, \mathbf{A}, \mathbf{B}\}$, where $\bot$ is an additional marker that should be thought of as *not yet decided*. The hard constraints have zero cost only for labelings $(\mathbf{A}, \mathbf{B})$, $(\mathbf{B}, \mathbf{A})$ and $(\bot, \bot)$, and infinite cost otherwise. The soft constraints have cost 0 for equal labels on the endpoints, 1 for unequal from $\{\mathbf{A}, \mathbf{B}\}$, and $\frac{1}{2}$ when exactly one endpoint is assigned $\bot$. Based on previous results of Kolmogorov, Thapper, and Živný [11], Wahlström observed that this relaxation is polynomial-time solvable, and moreover it is *persistent*: whenever the relaxation assigns $\mathbf{A}$ or $\mathbf{B}$ to some vertex, then it is safe to perform the same assignment in the integral problem (i.e., not relaxed, only with the "integral" labels $\mathbf{A}, \mathbf{B}$). The algorithm constructs an integral labeling with a backtracking process that fixes labels of consecutive vertices of the graph. During this process, it maintains an optimum solution to the relaxation that is moreover *maximal*, in the sense that one cannot extend the current labeling by fixing integral labels on some undecided vertices without increasing the cost. This can be done by dint of persistence and polynomial-time solvability: we can check in polynomial time whether a non-trivial extension exists, and then it is safe to fix the labels of vertices that get decided. Thus, when the algorithm considers the next vertex u and branches into two cases, fixing label $\mathbf{A}$ or $\mathbf{B}$ on it, the optimum cost of the relaxation increases by at least $\frac{1}{2}$ in each branch. Hence the recursion tree can be pruned at depth $2p$, and we obtain a $4^p n^{\mathcal{O}(1)}$-time algorithm.

Our algorithm for Terminal Separation applies a similar branching strategy, where at each point we maintain some labeling of the vertices with $\mathbf{A}$, $\mathbf{B}$, and $\bot$ (undecided). Every terminal pair is either already *resolved* (assigned $(\mathbf{A}, \mathbf{B})$ or $(\mathbf{B}, \mathbf{A})$), or *unresolved* (assigned $(\bot, \bot)$). Using the insight of Wahlström we can assume that this labeling is maximal. Intuitively, we look at unresolved pairs from $\mathcal{T}$ and try to identify a pair (s, t) for which branching into labelings $(\mathbf{A}, \mathbf{B})$ and $(\mathbf{B}, \mathbf{A})$ leads to substantial progress. Here, we measure progress in terms of a potential μ that is a linear combination of three components:

- t, the number of unresolved terminal pairs;
- k, the current budget for the cost of the sought integral solution;
- ν, the difference between k and the cost of the current solution to the relaxation.

These ingredients are taken with weights $\alpha_t = 0.59950$, $\alpha_\nu = 0.29774$, and $\alpha_k = 1 - \alpha_t - \alpha_\nu = 0.10276$. Thus, the largest weight is put on the progress measured in terms of the number of resolved terminal pairs: We want to argue that if we can identify a possibility of recursing into two instances, where in each of them at least one new terminal pair gets resolved, but in one of them we resolve two terminal pairs, then we can pursue this branching step.

Therefore, we are left with the following situation: when branching on any terminal pair, only this terminal pair gets resolved in both branches. Then the idea is to find a branching step where the decrease of the auxiliary components of the potential, namely ν and k, is significant enough to ensure the promised running time of the algorithm. Here we apply an extensive combinatorial analysis of the instance to show that finding such a branching step is always possible. In particular, our analysis can end up with a branching not on a terminal pair, but on the label of some other vertex; however, we make sure that in both branches some terminal pair gets eventually resolved. Also, in some cases we localize a part of the input that can be simplified (a *reduction step*), and then the analysis is restarted.

To sum up, we would like to highlight two aspects of our contribution. First, we answer a natural question stemming from the optimality program, showing that 2^k is not the final dependency on the parameter for Edge Bipartization. Second, our algorithm can be seen as a "proof of concept" that the LP-guided branching technique, even in the more abstract variant of Wahlström [18], can be combined with involved Measure&Conquer analysis of the branching tree. Note that in the past Measure&Conquer and related techniques led to rapid progress in the area of moderately-exponential algorithms [4].

We remark that the goal of the current paper is clearly improving the 2^k term, and not optimizing the dependence of the running time on the input size. However, we do estimate it. Using the tools prepared by Iwata, Wahlström, and Yoishida [8], we are able to implement the algorithm so that it runs in time $\mathcal{O}(1.977^k \cdot nm)$. Naively, this seems like an improvement over the algorithm of Guo et al. [6] that had quadratic dependence on m, however this is not the case. We namely use the recent approximation algorithm for Edge Bipartization of Kolay et al. [9] that in time $\mathcal{O}(k^{\mathcal{O}(1)} \cdot m)$ either returns a solution F^{apx} of size at most $\mathcal{O}(k^2)$, or correctly concludes that there is no solution of size k. Then we start iterative compression from $G - F^{\text{apx}}$ and introduce edges of F^{apx} one by one, so we need to solve the Terminal Separation problem only $\mathcal{O}(k^2)$ times. In our case each iteration takes time $\mathcal{O}(1.977^k \cdot nm)$, but for the approach of Guo et al. it would take time $\mathcal{O}(2^k \cdot km)$. Thus, by using the same idea based on [9], the algorithm of Guo et al. can be adjusted to run in time $\mathcal{O}(2^k \cdot k^3 m)$.

2 Overview of the algorithm

As announced in the introduction, the application of iterative compression and the reduction to a Terminal Separation instance closely follows the approach of [6]; in this extended abstract, we give only an overview of the branching algorithm for Terminal Separation.

Let us start with some notation. Consider a graph G with a family $\mathcal{T}$ of disjoint pairs of vertices in G; we call those vertices *terminals*. A *terminal separation* is a pair (A, B) with $A, B \subseteq V(G)$ such that $A \cap B = \emptyset$ and, for every terminal pair P, either one of the terminals in P belongs to A and the second to B, or $P \subseteq V(G) \setminus (A \cup B)$. A terminal separation (A, B) is *integral* if $A \cup B = V(G)$.[2] A terminal separation (A', B') *extends* (A, B) if $A \subseteq A'$ and $B \subseteq B'$. The *cost* of a terminal separation (A, B) is defined as $c(A, B) = (d(A) + d(B))/2$, where $d(X)$ is the number of edges between X and $V(G) \setminus X$, for $X \subseteq V(G)$. Note that if (A, B) is integral, then we have $c(A, B) = d(A) = d(B)$. We say that a terminal separation (A, B) is *maximal* if every other separation extending it has strictly larger cost.

Terminal Separation
Input: A graph G with a set of disjoint terminal pairs $\mathcal{T}$ such that every terminal is of degree at most one in G; a terminal separation (A°, B°); and an integer k.
Goal: Find an integral terminal separation (A, B) extending (A°, B°) of cost at most k, or report that no such separation exists.

We borrow the basic toolbox from [18, 8], in the form of the following two statements.

▶ **Theorem 2.1** (persistence [18]). *Let $(G, \mathcal{T}, (A^\circ, B^\circ), k)$ be a* Terminal Separation *instance, and let (A, B) be a terminal separation in G of minimum cost among separations that extend (A°, B°). Then there exists an integral separation (A^*, B^*) that has minimum cost among all integral separations extending (A°, B°), with the additional property that (A^*, B^*) extends (A, B).*

▶ **Theorem 2.2** (polynomial-time solvability, [18, 8]). *Given a* Terminal Separation *instance $(G, \mathcal{T}, (A^\circ, B^\circ), k)$ with $c(A^\circ, B^\circ) \leq k$, one can in $\mathcal{O}(k^{\mathcal{O}(1)}m)$ time find a maximal terminal separation (A, B) in G that has minimum cost among all separations extending (A°, B°).*

[2] The word *integral* stems from the fact that an integral separation corresponds to a solution to the relaxed Terminal Separation problem that actually does not use the relaxed value $\perp$. In fact, it also corresponds to an integral solution of an LP formulation underlying the algorithmic results of [11].

From Theorems 2.1 and 2.2 it follows that, while working on a Terminal Separation instance $(G, \mathcal{T}, (A^\circ, B^\circ), k)$, we can always assume that (A°, B°) is a maximal separation: If that is not the case, we can obtain an extending separation (A, B) via Theorem 2.2, and set $(A^\circ, B^\circ) := (A, B)$; the safeness of the last step is guaranteed by Theorem 2.1.

2.1 The potential to measure progress of the algorithm

Let $\mathcal{I} = (G, \mathcal{T}, (A^\circ, B^\circ), k)$ be a Terminal Separation instance, where (A_0, B_0) is a maximal terminal separation; we henceforth call such an instance *maximal*. We are interested in keeping track of the following partial measures:

- $t_\mathcal{I}$ is the number of unresolved terminal pairs;
- $\nu_\mathcal{I} = k - c(A^\circ, B^\circ)$;
- $k_\mathcal{I} = k$.

The $\mathcal{O}(2^k km)$-time algorithm used in [6] can be interpreted in our framework as an $\mathcal{O}(2^{t_\mathcal{I}} k_\mathcal{I} m)$-time algorithm for Terminal Separation, while the generic LP-branching algorithm for Edge Unique Label Cover of [18, 8] can be interpreted as an $\mathcal{O}(4^{\nu_\mathcal{I}} m)$-time algorithm. Our main goal is to blend the two, by analyzing the cases where both perform badly.

An important insight is that all these inefficient cases happen when A° and B° increase their common boundary. If this is the case, a simple reduction rule is applicable that also reduces the allowed budget k.

▶ **Reduction 2.3** (Boundary Reduction). *If there exists an edge ab with $a \in A^\circ$, $b \in B^\circ$, then delete the edge ab and decrease k by one. If there exist two edges va, vb with $a \in A^\circ$, $b \in B^\circ$, and $v \notin A^\circ \cup B^\circ$, then delete both edges va and vb, and decrease k by one.*

In some sense, with this reduction rule the budget k represents the yet undetermined part of the boundary between A^* and B^* in the final integral solution (A^*, B^*). For this reason, we also include the budget k in the potential.

Formally, we fix three constants $\alpha_t = 0.59950$, $\alpha_\nu = 0.29774$, and $\alpha_k = 1 - \alpha_t - \alpha_\nu = 0.10276$ and define a potential of an instance $\mathcal{I}$ as

$$\mu_\mathcal{I} = \alpha_t \cdot t_\mathcal{I} + \alpha_\nu \cdot \nu_\mathcal{I} + \alpha_k \cdot k_\mathcal{I}.$$

Our main technical result is the following.

▶ **Theorem 2.4.** *A Terminal Separation instance $\mathcal{I}$ can be solved in time $\mathcal{O}(c^{\mu_\mathcal{I}} nm)$ for some $c < 1.977$.*

We remark that instances of Terminal Separation we encounter while solving an Edge Bipartization instance (G, k) satisfy $t_\mathcal{I} = k + 1$, $\nu_\mathcal{I} = k$, and $\alpha_k = k$, hence $\mu_\mathcal{I} < k + 1$. Consequently, Theorem 1.1 follows from Theorem 2.4 by using it in the general iterative compression approach proposed by Guo et al. [6].

The algorithm of Theorem 2.4 follows a typical outline of a recursive branching algorithm. At every step, the current instance is analyzed, and either it is reduced, or some two-way branching step is performed. The potential $\mu_\mathcal{I}$ is used to measure the progress of the algorithm and to limit the size of the branching tree.

Observe that the Boundary Reduction reduces already determined parts of the boundary between A^* and B^* for the minimum-cost solution (A^*, B^*), and hence the integer $k_\mathcal{I}$, present in the potential $\mu_\mathcal{I}$, represents the yet unknown part of this boundary. It is easy to see that every application of the Boundary Reduction decreases the potential by exactly α_k; in multiple branches we show that a sufficient number of Boundary Reductions follow the branching step to ensure the promised running time bound.

2.2 Structure of a branching step

In every branching step, we identify two terminal separations (A_1, B_1) and (A_2, B_2) extending (A°, B°), and branch into two subcases; in subcase i we replace (A°, B°) with (A_i, B_i). We always argue the *correctness* of a branch by showing that there exists an integral solution (A^*, B^*) extending (A°, B°) of minimum cost, with the additional property that (A^*, B^*) extends (A_i, B_i) for some $i = 1, 2$. In subcase i, we apply the algorithm of Theorem 2.2 to $(G, \mathcal{T}, (A_i, B_i), k)$ to obtain a maximal separation (A_i°, B_i°), and pass the instance $\mathcal{I}_i = (G, \mathcal{T}, (A_i^\circ, B_i^\circ), k)$ to a recursive call.

To show the *running time bound* for a branching step, we analyze how the measure $\mu_\mathcal{I}$ decreases in the subcases, taking into account the reductions performed in the subsequent recursive calls. More formally, we say that a branching case *fulfills a branching vector* $[t_1, \nu_1, k_1;\ t_2, \nu_2, k_2]$ if, in subcase $i = 1, 2$, at least t_i terminal pairs become resolved or reduced with one of the reductions, the cost of the separation (A_i°, B_i°) grows by at least $\nu_i/2$, and the Boundary Reduction gets applied at least k_i times in the instance $(G, \mathcal{T}, (A_i^\circ, B_i^\circ), k)$.

A branching vector $[t_1, \nu_1, k_1; t_2, \nu_2, k_2]$ is *good* if

$$1.977^{-\alpha_t t_1 - \alpha_\nu \nu_1/2 - \alpha_k k_1} + 1.977^{-\alpha_t t_2 - \alpha_\nu \nu_2/2 - \alpha_k k_2} < 1.$$

Standard arguments for branching algorithms show that, if in every case we perform a branching step that fulfills some good branching vector, the branching tree originated from an instance $\mathcal{I}$ has $\mathcal{O}(c^{\mu_\mathcal{I}})$ leaves for some $c < 1.977$. To simplify further exposition, we gather in the next lemma good branching vectors used in the analysis; the fact that they are good can be checked by direct calculations.

▶ **Lemma 2.5.** *The following branching vectors are good:*

$[1, 1, 0; 2, 1, 0]$	$[1, 1, 1; 1, 2, 3]$	$[1, 2, 0; 1, 3, 1]$	$[1, 1, 0; 1, 4, 3]$	$[1, 1, 2; 1, 2, 2]$
$[1, 1, 1; 1, 3, 2]$	$[1, 3, 0; 1, 3, 0]$	$[1, 1, 0; 1, 5, 2]$	$[1, 2, 1; 1, 2, 2]$	$[1, 1, 1; 1, 4, 1]$

Let us stop here to comment that the vectors in Lemma 2.5 explain our choice of constants α_t, α_ν, α_k. The constant α_t is sufficiently large to make the vector $[1, 1, 0; 2, 1, 0]$ good; intuitively speaking, we are always done when in one branch we manage to resolve or reduce at least two terminal pairs. The choice of α_ν and α_k represents a very delicate tradeoff that makes both $[1, 1, 1; 1, 2, 3]$ and $[1, 2, 0; 1, 3, 1]$ good; note that setting $\alpha_\nu = 1 - \alpha_t$ and $\alpha_k = 0$ makes the first vector not good, while setting $\alpha_\nu = 0$ and $\alpha_k = 1 - \alpha_t$ makes the second vector not good. Arguably, the possibility of a tradeoff that makes both the second and the third vector of Lemma 2.5 good at the same time is one of the critical insights in our work.

2.3 Low-excess sets

A set $A \subseteq V(G)$ is an A°-*extension* if $A^\circ \subseteq A \subseteq V(G) \setminus B^\circ$. It is *terminal-free* if $A \setminus A^\circ$ does not contain any terminal. We denote by $\Delta(A) := d(A) - d(A^\circ)$ the *excess* of an A°-extension A. An A°-extension A is *compact* if $A \setminus A^\circ$ is connected and $E(A \setminus A^\circ, A^\circ) \neq \emptyset$.

One of the main technical tools for analysis is the study of extensions of small excess. We show that their structure can be reduced to have a relatively simple picture. While in this section we focus on supersets of the set A°, by symmetry the same conclusion holds if we swap the roles of A° and B°.

First, since (A°, B°) is maximal, we have that A° is the only terminal-free A°-extension of nonpositive excess. As for excess 1, one can show the following.

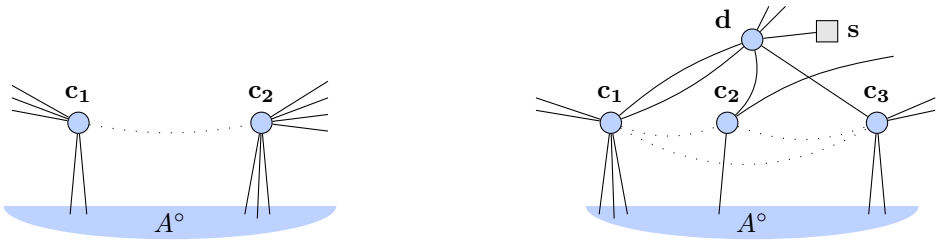


Figure 1 Examples of sets of excess 2 after reductions (dotted lines are non-edges). On the right a strict (non-null) extension A_s of $A^\circ \cup \{s\}$ with excess 1 is shown. For any such extension, $A_s \setminus \{s\}$ is a set of excess 2 in which the vertex d, obtained from contracting the set D of the decomposition, is the only neighbor of the terminal s.

▶ **Lemma 2.6.** *If A is a terminal-free A°-extension of excess 1, then there exists a minimum cost integral terminal separation (A^*, B^*) extending (A°, B°), such that $(A \setminus A^\circ)$ is either completely contained in A^* or completely contained in B^*.*

Hence, one can collapse into a single vertex the set $A \setminus A^\circ$ for every terminal-free A°-extension A of excess 1. For extensions of excess 2, one can describe them similarly, and collapse into a single vertex any set D as in the following lemma.

▶ **Lemma 2.7.** *Assume that every terminal-free A°-extension of excess 1 has been collapsed to a single vertex, and that G contains no nonterminal degree-1 vertices. If A is a terminal-free A°-extension of excess 2, then there exists a partition $A \setminus A^\circ = D \uplus C_1 \uplus C_2 \uplus \ldots \uplus C_r$ for some $r \geq 0$ ($\uplus$ meaning union of disjoint sets), such that:*

1. *there exists a minimum cost integral terminal separation (A^*, B^*) extending (A°, B°), such that one of the following holds:*
   - *$(A \setminus A^\circ) \cap A^* = \emptyset$;*
   - *$(A \setminus A^\circ) \cap A^* = C_i$ for some $1 \leq i \leq r$; or*
   - *$A \subseteq A^*$.*
2. *for every $1 \leq i \leq r$, the sets C_i and $E(C_i, A^\circ)$ are nonempty, and $A^\circ \cup C_i$ is a terminal-free A°-extension of excess 1;*
3. *if $D \neq \emptyset$, then for every $1 \leq i \leq r$ the set $E(C_i, D)$ is nonempty and $A \setminus A^\circ$ is connected;*
4. *if $D = \emptyset$, then $r = 2$;*
5. *for every $1 \leq i < j \leq r$, there are no edges between C_i and C_j.*

2.4 Basic branching step

Let $\mathcal{T}' := \mathcal{T} \setminus (A^\circ \cup B^\circ)$ be the set of unresolved terminal pairs. In the basic branching step of our algorithm we take a pair $\{s, t\} \in \mathcal{T}'$ and try to assign s and t to the different sides of the separation. That is, we apply the algorithm of Theorem 2.2 twice: once for terminal separation $(A^\circ \cup \{s\}, B^\circ \cup \{t\})$, and the second time for terminal separation $(A^\circ \cup \{t\}, B^\circ \cup \{s\})$. In this manner we obtain two maximal terminal separations (A_s, B_t) and (A_t, B_s) that extend $(A^\circ \cup \{s\}, B^\circ \cup \{t\})$ and $(A^\circ \cup \{t\}, B^\circ \cup \{s\})$ respectively. Of course, the number of unresolved pairs decreases by at least one in both (A_s, B_t) and (A_t, B_s), due to resolving $\{s, t\}$.

If the number of unresolved pairs either in (A_s, B_t) or in (A_t, B_s) decreases by more than one, then performing a branching step $(A_1, B_1) = (A_s, B_t)$ and $(A_2, B_2) = (A_t, B_s)$ leads to the branching vector $[1, 1, 0; 2, 1, 0]$ or a better one, which is good; the corresponding decrease in the measure $\nu_{\mathcal{I}}$ follows from the assumption that (A°, B°) is maximal. We can test in

$\mathcal{O}(k^{\mathcal{O}(1)}m)$ time whether this holds for any pair $\{s,t\} \in \mathcal{T}'$, and if so then we pursue the branching step.

If this is not the case, we are left with the extensive analysis of the sets A_s, B_s, A_t, and B_t. As we could always pick $A_s = A^\circ \cup \{s\}$, and similarly for the other sets, we have that the excess of any of these four sets is at most one. Furthermore, the maximality of (A°, B°) implies that also neither of these excesses is negative: if, say, $\Delta(A_s) < 0$, then since $\Delta(B_t) \leq 1$, we have $c(A_s, B_t) \leq c(A^\circ, B^\circ)$, contradicting the maximality of (A°, B°). Thus, we are left with excesses 0 and 1, giving different cases for analysis.

Let us first consider the case when $A_s = A^\circ \cup \{s\}$, $A_t = A^\circ \cup \{t\}$, $B_s = B^\circ \cup \{s\}$, and $A_t = B^\circ \cup \{t\}$, that is, the situation when both branching steps colored only the terminals. If s or t is an isolated vertex in G, it is easy to reduce the pair $\{s,t\}$ without branching. Otherwise, let s' be the unique neighbor of s and t' be the unique neighbor of t. Since both s and t are of degree one in G, it is easy to argue that there exists a minimum-cost solution (A^*, B^*) that does not cut the edge ss'. Consequently, we can strengthen the basic branch by forcing s' to be in the same side of the separation as s. More precisely, we consider branches $(A_{ss'\to A}, B_{ss'\to A})$ and $(A_{ss'\to B}, B_{ss'\to B})$ that are minimum-cost terminal separations extending $(A^\circ \cup \{s,s'\}, B^\circ \cup \{t\})$ and $(A^\circ \cup \{t\}, B^\circ \cup \{s,s'\})$, computed using Theorem 2.2. It is easy to see that, unless some simple reduction is applicable or another terminal pair gets resolved, we have $\Delta(A_{ss'\to A}) \geq 2$ and $\Delta(B_{ss'\to B}) \geq 2$ while still $\Delta(B_{ss'\to A}), \Delta(A_{ss'\to B}) \geq 1$. This gives a good branching vector $[1,3,0;1,3,0]$.

In the analysis of remaining cases we rely on our understanding of low-excess extensions in the following way. Assume that, say, A_s is a strict superset of $A^\circ \cup \{s\}$. Then A_s contains s' and $A := A_s \setminus \{s\}$ is a terminal free A°-extension of excess $\Delta(A_s) + 1 \in \{1,2\}$. Thus, Lemma 2.6 or 2.7 applies, giving us a good insight into the set $A \setminus A^\circ$, capturing the neighborhood of s. Observe that the structure of an excess-1 or excess-2 extension in particular guarantees that A lies "closely" to the set A°, giving grounds for possibly multiple Boundary Reductions in a subcase in a branching step when some vertices of A are assigned to the B-side of the separation.

An extensive case analysis, provided in the full version of the paper, shows that in all cases, if the basic branching resolves only one terminal pair, then one can gather a sufficient number of Boundary Reductions stemming from the understanding of low-excess extensions and sufficient increase in the cost of the separation (A°, B°) to obtain a good branching vector. This proves Theorem 2.4. In the remainder of this section, we illustrate how the low-excess extensions work by sketching one particular subcase of the case $\Delta(A_s) = 1$ and $\Delta(B_s) = 0$. This illustration is quite representative for the kind of reasoning we need to perform in other cases as well.

2.5 Example subcase of the case $\Delta(A_s) = 1$, $\Delta(B_s) = 0$

We define $R = V(G) \setminus (A_s \cup B_s)$, $\tilde{A} = A_s \setminus B_s$ and $\tilde{B} = B_s \setminus A_s$; note that $\tilde{A}$ and $\tilde{B}$ are terminal-free extensions of A° and B°, respectively. By posimodularity of the cuts we infer:

$$d(A_s) + d(B_s) = d(\tilde{A}) + d(\tilde{B}) + 2|E(A_s \cap B_s, R)| \geq d(A^\circ) + d(B^\circ) + 2|E(A_s \cap B_s, R)|.$$

We infer that in our case $|E(A_s \cap B_s), R)| = 0$ and $\Delta(\tilde{A}) + \Delta(\tilde{B}) = 1$. In this overview we consider the subcase $\Delta(\tilde{A}) = 1$ and $\Delta(\tilde{B}) = 0$.

By maximality of (A°, B°) we have $\tilde{B} = B^\circ$; by Lemma 2.6 we can assume $\tilde{A} = A^\circ \cup \{a\}$ for some vertex a; in particular $A_s \supsetneq A^\circ \cup \{s\}$. Let s' be the unique neighbor of s; we have $s' \in A_s$, as otherwise $A_s \setminus \{s\}$ is a nontrivial A°-extension of excess 0, a contradiction. As $\Delta(A_s) = 1$, the set $A_s \setminus \{s\}$ is a terminal-free A°-extension of excess 2: we can apply Lemma 2.7

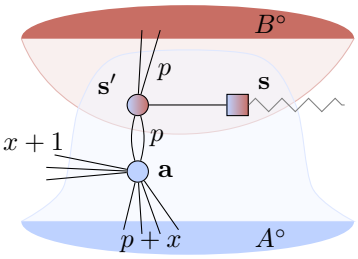


Figure 2 Subcase $(\Delta(A_s), \Delta(B_s)) = (\Delta(\tilde{A}), \Delta(\tilde{B})) = (1, 0)$. Extensions A_s, B_s are highlighted.

to obtain a decomposition $A_s \setminus \{s\} = A^\circ \uplus \{d, c_1, c_2, \ldots, c_r\}$ or $A_s \setminus \{s\} = A^\circ \uplus \{c_1, c_2\}$ (vertices d and c_i are sets D and C_i from Lemma 2.7 collapsed into single vertices due to reduction rules). From the fact that $s' \in A_s$ and (A_s, B_t) is a separation extending $(A^\circ \cup \{s\}, B^\circ \cup \{t\})$ of minimum-cost, we infer that s' is actually the vertex d: if $s' = c_i$ for some i, then $(A^\circ \cup \{s, c_i\}, B_t)$ would have strictly smaller cost. In particular, we are dealing with the decomposition of the form $A_s \setminus \{s\} = A^\circ \uplus \{d, c_1, c_2, \ldots, c_r\}$.

Since $\Delta(B_s) = 0$ but $\Delta(B^\circ \cup \{s\}) = 1$, we infer that $B_s \supsetneq B^\circ \cup \{s\}$, which implies that $s' \in B_s$. Furthermore, we can assume that $A_s \cap B_s = \{s, s'\}$: if there were more vertices in $A_s \cap B_s$, it is easy to see that we can safely reduce the graph by collapsing $(A_s \cap B_s) \setminus \{s\}$ into a single vertex. Consequently, as $\tilde{A} = A^\circ \cup \{a\}$ and $\tilde{B} = B^\circ$, we have $B_s = B^\circ \cup \{s, s'\}$ and $A_s = A^\circ \cup \{a, s', s\}$ (i.e., $r = 1$ and $c_1 = a$).

From Lemma 2.7 we have $p := |E(a, s')| \geq 1$ and $|E(a, A^\circ)| \geq 1$, thus $E(a, B^\circ) = \emptyset$ since Boundary Reduction does not apply to a. By using the assumptions on excesses of sets, we have that a is incident on: p edges to s', $x+1$ edges to $V(G) \setminus (A_s \cup B^\circ)$, $p+x$ edges to A° and no other edges, for some $x \geq 0$. Since $B_s = B^\circ \cup \{s', s\}$ is an excess-0 set and $E(s', R) = \emptyset$, we have that $|E(s', B^\circ)| = p + |E(s', A^\circ)|$. In particular $|E(s', B^\circ)| > 0$, so since Boundary Reductions do not apply to s', we have $E(s', A^\circ) = \emptyset$ and hence $|E(s', B^\circ)| = p$. See Fig. 2.

Consider first case $x = 0$. Then a has a unique edge aa' with $a' \in R$. If a' is a terminal, it is easy to either reduce the case without branching (if $a' = t$) or provide a branching step that resolves two terminal pairs (if a' belongs to other terminal pair), so assume otherwise. We claim that it is a safe reduction to contract the edge aa'; to prove this claim, it suffices to show that there exists an optimum integral terminal separation extending (A°, B°) where a and a' belong to the same side. Take any such integral terminal separation (A^*, B^*), and assume that a and a' are on opposite sides. Clearly it cannot happen that $a \in B^*$ and $a' \in A^*$, because then moving a from B^* to A^* would decrease the cost of the separation. Hence $a \in A^*$ and $a' \in B^*$. If $s' \in B^*$, then moving a from A^* to B^* would decrease the cost of the separation, so also $s' \in A^*$. Construct a new integral separation (A^*_m, B^*_m) from (A^*, B^*) by moving $\{a, s'\}$ from A^* to B^*. Then the cost of (A^*_m, B^*_m) is not larger than that of (A^*, B^*) (we could have broken the edge $s's$ instead of aa'), while both endpoints of aa' belong to A^*_m. This resolves the case $x = 0$.

In the case $x > 0$, we claim that branching on the membership of a leads to a good branch. That is, we recurse into two branches $(A_{a\to A}, B_{a\to A})$ and $(A_{a\to B}, B_{a\to B})$ that are minimum-cost maximal terminal separations extending $(A^\circ \cup \{a\}, B^\circ)$ and $(A^\circ, B^\circ \cup \{a\})$, respectively. For $X \in \{A, B\}$, let $t_{a\to X}, \nu_{a\to X}, k_{a\to X}$ be the changes of the components of the potential in respective branches, as we denote them in branching vectors.

Consider first the branch $(A_{a\to A}, B_{a\to A})$. Then p Boundary Reductions are triggered on vertex s' (regardless of whether it is added or not to one of the sets $A_{a\to A}, B_{a\to A}$). Hence $k_{a\to A} \geq p$. Moreover, the terminal pair $\{s, t\}$ either is already resolved by $(A_{a\to A}, B_{a\to A})$

or is easily reducible after applying the Boundary Reductions. Hence $t_{a\to A} \geq 1$. Finally, since (A°, B°) was maximal, we have that $\nu_{a\to A} \geq 1$. So the part of the branching vector corresponding to the branch $(A_{a\to A}, B_{a\to A})$ is $[1, 1, p]$, or better.

Consider now the second branch $(A_{a\to B}, B_{a\to B})$. Then at least $|E(a, A^\circ)| = p + x$ Boundary Reductions are triggered, hence $k_{a\to B} \geq p + x$. Since $p \geq 1$ and t is of degree 1, $s' \in B_{a\to B}$ and w.l.o.g. we can assume $s \in B_{a\to B}$ and $t \in A_{a\to B}$. Hence $t_{a\to B} \geq 1$. If actually $t_{a\to A} \geq 2$ or $t_{a\to B} \geq 2$, then we arrive at a good branching vector $[1, 1, p; 2, 1, p]$ or better, so assume that $t_{a\to A} = t_{a\to B} = 1$, that is, only the pair $\{s, t\}$ gets resolved.

We now claim that $\Delta(A_{a\to B}) \geq 1$ and $\Delta(B_{a\to B}) \geq 1$. For the latter claim, note that if $\Delta(B_{a\to B}) \leq 0$, then $B_{a\to B} \setminus \{s\}$ is a terminal-free B°-extension of excess at most one, while $a, s' \in B_{a\to B}$; a contradiction to the assumption that every terminal-free extension of excess one has been collapsed into a single vertex. For the former claim, suppose for the sake of contradiction that $d(A_{a\to B}) = d(A^\circ)$ (case $d(A_{a\to B}) < d(A^\circ)$ can easily be excluded by the maximality of (A°, B°)). Recall that also $d(B_s) = d(B^\circ)$, which means that $d(A_{a\to B}) + d(B_s) = c(A^\circ, B^\circ)$. From the posimodularity of cuts it now follows that one of the terminal separations $(A_{a\to B} \setminus B_s, B_s)$ and $(A_{a\to B}, B_s \setminus A_{a\to B})$ has cost not larger than (A°, B°), while both of them resolve the terminal pair $\{s, t\}$. This is a contradiction with the maximality of (A°, B°). Hence $\Delta(A_{a\to B}) \geq 1$ and $\Delta(B_{a\to B}) \geq 1$, and so $\nu_{a\to B} \geq 2$.

Thus, branching into separations $(A_{a\to A}, B_{a\to A})$ and $(A_{a\to B}, B_{a\to B})$ leads to a branching vector $[1, 1, p; 1, 2, p + x]$ or better. Recalling that $p, x > 0$, observe that this branching vector can be not good only if $p = x = 1$ and $\Delta(B_{a\to B}) = 1$. Let us now analyze this case.

Since $\Delta(B_{a\to B}) = 1$, we have that $B_{a\to B} \setminus \{s\}$ is a terminal-free set of excess 2, and hence we can apply Lemma 2.7 to it: assuming excess-2 sets have been reduced, we have that $B_{a\to B} \setminus \{s\}$ has a decomposition of the form $B^\circ \uplus \{c_1, c_2\}$ or $B^\circ \uplus \{d, c_1, \ldots, c_r\}$. Note that $B^\circ \cup \{s'\}$ is an excess-1 set, so it is not hard to argue that $s' = c_i$ for some i. As $a \in B_{a\to B}$, a is adjacent to s', and c_i-s are pairwise non-adjacent, we must have that $a = d$ and we are dealing with a decomposition of the form $B^\circ \uplus \{d, c_1, \ldots, c_r\}$. Observe that $B^\circ \cup \{a, s'\}$ is a B°-extension of excess at least $1 + (x + 1) = 3$ (counting edge ss' and edges between a and A°); hence $B_{a\to B} \supsetneq B^\circ \cup \{a, s', s\}$, and in particular $r > 1$. Hence there exists some vertex $c_j \neq c_i = s'$. By Lemma 2.7 we have that c_j is adjacent both to B° and to a. Hence, in the branch $(A_{a\to A}, B_{a\to A})$ at least one Boundary Reduction is applied to c_j, regardless whether c_j is assigned to $A_{a\to A}$, or $B_{a\to A}$, or neither of these sets. We did not include this Boundary Reduction in the previous calculations; this shows that we in fact pursue a branch with a branching vector $[1, 1, 2; 1, 2, 2]$ or better, which is a good branching vector.

3 Conclusions

In this work we have developed an algorithm for Edge Bipartization with running time $\mathcal{O}(1.977^k \cdot nm)$, which is the first one to achieve the dependence on the parameter better than 2^k. Thus, in the case of Edge Bipartization the constant 2 in the base of the exponent is not the ultimate answer, as is conjectured for CNF-SAT. Also, it improves some recent works where the FPT algorithm for Edge Bipartization is used as a black-box [10]. However, our work leaves some open questions that we would like to highlight.

- Reducing the dependence on the parameter from 2^k to 1.977^k can be only considered a "proof of concept" that such an improvement is possible. We put forward the question of designing a reasonably simple algorithm with significant improvement in the base of the exponent, hopefully decreasing the polynomial dependence from $\mathcal{O}(nm)$ to (near-)linear.
- Our approach can be summarized as follows: having observed that Terminal Separation admits a simple $\mathcal{O}^\star(2^{|\mathcal{T}|})$-time algorithm and an $\mathcal{O}^\star(4^k)$-time algorithm using the

CSP-guided technique of Wahlström [18], we develop an algorithm for a joint parameterization $(|\mathcal{T}|, k)$ that for $|\mathcal{T}| = k + 1$ achieves running time $\mathcal{O}^\star(1.977^k)$. Can Terminal Separation be solved in time $\mathcal{O}^\star(c^{|\mathcal{T}|})$ for some $c < 2$?

References

1 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41, 2016. doi:10.1145/2925416.

2 Marek Cygan, Łukasz Kowalik, and Marcin Pilipczuk. Open problems from the update meeting on graph separation problems, Workshop on Kernels, Warsaw, 2013. http://worker2013.mimuw.edu.pl/slides/update-opl.pdf.

3 Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. On Multiway Cut parameterized above lower bounds. *TOCT*, 5(1):3, 2013. doi:10.1145/2462896.2462899.

4 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010. doi:10.1007/978-3-642-16533-7.

5 Sylvain Guillemot. FPT algorithms for path-transversal and cycle-transversal problems. *Discrete Optimization*, 8(1):61–71, 2011. doi:10.1016/j.disopt.2010.05.003.

6 Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006.

7 Falk Hüffner. Algorithm engineering for optimal Graph Bipartization. *J. Graph Algorithms Appl.*, 13(2):77–98, 2009.

8 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.

9 Sudeshna Kolay, Pranabendu Misra, M. S. Ramanujan, and Saket Saurabh. Parameterized approximations via d-Skew-Symmetric Multicut. In *Proc. MFCS'14*, volume 8635 of *Lecture Notes in Computer Science*, pages 457–468. Springer, 2014. doi:10.1007/978-3-662-44465-8_39.

10 Sudeshna Kolay, Fahad Panolan, Venkatesh Raman, and Saket Saurabh. Parameterized algorithms on perfect graphs for deletion to (r, l)-graphs. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *Proc. MFCS'14*, volume 58 of *LIPIcs*, pages 75:1–75:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.MFCS.2016.75.

11 Vladimir Kolmogorov, Johan Thapper, and Stanislav Živný. The power of linear programming for general-valued CSPs. *SIAM J. Comput.*, 44(1):1–36, 2015.

12 Stefan Kratsch and Magnus Wahlström. Compression via matroids: A randomized polynomial kernel for Odd Cycle Transversal. *ACM Trans. Algorithms*, 10(4):20:1–20:15, 2014.

13 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

14 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.

15 Daniel Lokshtanov, Saket Saurabh, and Somnath Sikdar. Simpler parameterized algorithm for OCT. In Jirí Fiala, Jan Kratochvíl, and Mirka Miller, editors, *Proc. IWOCA 2009, Revised Selected Papers*, volume 5874 of *Lecture Notes in Computer Science*, pages 380–384. Springer, 2009. doi:10.1007/978-3-642-10217-2_37.

16 Dániel Marx. What's next? Future directions in Parameterized Complexity. In *The Multivariate Algorithmic Revolution and Beyond*, volume 7370 of *Lecture Notes in Computer Science*, pages 469–496. Springer, 2012.

17 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004.

18 Magnus Wahlström. Half-integrality, LP-branching and FPT algorithms. In *Proc. SODA'14*, pages 1762–1781. SIAM, 2014.

Cut and Count and Representative Sets on Branch Decompositions

Willem J. A. Pino[1], Hans L. Bodlaender[*2], and Johan M. M. van Rooij[3]

1 **Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands**
w.j.a.pino@students.uu.nl

2 **Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, and**
Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
H.L.Bodlaender@uu.nl

3 **Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, and**
Consultants in Quantitative Methods, Eindhoven, The Netherlands
jmmrooij@cs.uu.nl

Abstract

Recently, new techniques have been introduced to speed up dynamic programming algorithms on tree decompositions for connectivity problems: the 'Cut and Count' method and a method called the rank-based approach, based on representative sets and Gaussian elimination. These methods respectively give randomised and deterministic algorithms that are single exponential in the treewidth, and polynomial, respectively linear in the number of vertices. In this paper, we adapt these methods to branch decompositions yielding algorithms, both randomised and deterministic, that are in many cases faster than when tree decompositions would be used.

In particular, we obtain the currently fastest randomised algorithms for several problems on planar graphs. When the involved weights are $\mathcal{O}(n^{\mathcal{O}(1)})$, we obtain faster randomised algorithms on planar graphs for Steiner Tree, Connected Dominating Set, Feedback Vertex Set and TSP, and a faster deterministic algorithm for TSP. When considering planar graphs with arbitrary real weights, we obtain faster deterministic algorithms for all four mentioned problems.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory, I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases Graph algorithms, Branchwidth; Treewidth, Dynamic Programming, Planar Graphs

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.27

1 Introduction

It is well known that many problems that are NP-hard on general graphs, become polynomial or linear time solvable on graphs where the treewidth or branchwidth is bounded by a constant. More precisely, many problems are fixed parameter tractable with treewidth or branchwidth as parameter. For an overview regarding treewidth, e.g., see [1] .

* Hans L. Bodlaender was partially supported by the Networks project, funded by the Dutch Ministry of Education, Culture and Science through NWO.

Table 1 Our results using the 'Cut and Count' (randomised) and rank-based (exact) techniques.

Problem	Randomised	Deterministic
Steiner Tree	$\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$	$\mathcal{O}(n((1+2^{\omega})\sqrt{5})^{bw}bw^{\mathcal{O}(1)})$
Connected Dominating Set	$\mathcal{O}(4^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$	$\mathcal{O}(n((2+2^{\omega})\sqrt{6})^{bw}bw^{\mathcal{O}(1)})$
Feedback Vertex Set	$\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$	$\mathcal{O}(n((1+2^{\omega})\sqrt{5})^{bw}bw^{\mathcal{O}(1)})$
Hamilton Cycle / TSP	$\mathcal{O}(4^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$	$\mathcal{O}(n(5+2^{\frac{\omega+2}{2}})^{bw}bw^{\mathcal{O}(1)})$
Planar Steiner Tree	$\mathcal{O}(2^{3.991\sqrt{n}})$	$\mathcal{O}(2^{8.039\sqrt{n}})$
Planar Connected Dominating Set	$\mathcal{O}(2^{5.036\sqrt{n}})$	$\mathcal{O}(2^{8.778\sqrt{n}})$
Planar Feedback Vertex Set	$\mathcal{O}(2^{3.991\sqrt{n}})$	$\mathcal{O}(2^{8.039\sqrt{n}})$
Planar Hamilton Cycle / TSP	$\mathcal{O}(2^{5.036\sqrt{n}})$	$\mathcal{O}(2^{6.570\sqrt{n}})$

It was long known that many graph problems with a local nature (e.g., Independent Set, Dominating Set) can be solved on graphs given with a tree decomposition of width k in time, that is single exponential in k and linear in the number of vertices n, e.g., see [17]. For several problems with a global 'connectivity' property in it, it was open whether there existed $\mathcal{O}(2^{\mathcal{O}(k)}n^{\mathcal{O}(1)})$ time algorithms. This was resolved by Cygan et al. [5] with the '*Cut and Count*' method; this approach gives fast randomised algorithms that are single-exponential in the treewidth and polynomial in the number of vertices for various problems, e.g., Feedback Vertex Set, Hamiltonian Circuit, TSP, Connected Dominating Set. At the cost of a higher constant in the base of the exponential factor, Bodlaender et al. [2] gave deterministic algorithms that are single-exponential in the treewidth and linear in the number of vertices for these connectivity problems, with a technique, based on representative sets and Gaussian elimination, called the *rank-based* approach. This algorithm was experimentally evaluated by Fafianie et al. [9], showing that in the case of the Steiner Tree problem, the method gives a significant speedup over naive dynamic programming. An alternative method that gives similar time bounds, based on representative sets and matroids, was given by Fomin et al. [11]. Later, Fomin et al. [10] showed how to use matroids to speed up the computation at join nodes in these algorithms leading, for several connectivity problems with Steiner Tree as flagship example, to the currently fastest algorithms on graphs of bounded treewidth.

Branchwidth is another well studied graph parameter, with strong relations to treewidth. The branchwidth and treewidth of a graph are bounded by each other in the following way: $bw \leq tw + 1 \leq \lfloor \frac{3}{2}bw \rfloor$. The transformation from a tree decomposition to a branch decomposition or vice versa, fulfilling these bounds can be executed in linear time. This implies that a running times of the form $\mathcal{O}(c^k n^{\mathcal{O}(1)})$ for graphs of treewidth k or branchwidth k follow from each other, except for a possibly different value for the base of the exponent c.

In this paper, we show that 'Cut and Count' and the rank-based approach can be used directly on branch decompositions. As a result, we obtain, in several cases, improvements compared to using tree decompositions instead. For an overview of our results, see Table 1.

Two other techniques to speed up dynamic programming algorithms on tree and branch decompositions are the following: Dorn [6] showed how to use matrix multiplication to speed up algorithms on branch decompositions and van Rooij et al. [3, 18] showed how to speed up algorithms on tree, branch and clique decompositions using *(generalised) subset convolutions*. In this paper, we build upon these works applying these techniques where possible.

For a comparison of our results to the current best treewidth algorithms, see Table 2 and Table 3. Here, $\omega < 2.373$ [14] is the matrix multiplication exponent. Our branch decomposition based results improve known treewidth results for parts of the range $bw \leq tw + 1 \leq \lfloor \frac{3}{2}bw \rfloor$ (note that $\frac{\omega}{2} < \frac{3}{2}$). In case of deterministic algorithms for TSP with

Table 2 Comparison of our results with best known results on treewidth [5] for randomised algorithms on problems where the weights are $\mathcal{O}(n^{\mathcal{O}(1)})$.

Problem	Treewidth	Branchwidth
Steiner Tree	$\mathcal{O}(3^{tw}n^{\mathcal{O}(1)})$	$\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$
Connected Dominating Set	$\mathcal{O}(4^{tw}n^{\mathcal{O}(1)})$	$\mathcal{O}(4^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$
Feedback Vertex Set	$\mathcal{O}(3^{tw}n^{\mathcal{O}(1)})$	$\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$
Hamilton Cycle / TSP	$\mathcal{O}(4^{tw}n^{\mathcal{O}(1)})$	$\mathcal{O}(4^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$

Table 3 Comparison of our results with best known results on treewidth for deterministic algorithms on problems with arbitrary real weights.

Problem	Treewidth	Branchwidth
Steiner Tree	$\mathcal{O}(n2^{3.134tw})$ [10]	$\mathcal{O}(n2^{3.790bw})$
Connected Dominating Set	$\mathcal{O}(n2^{3.628tw})$ [2]	$\mathcal{O}(n2^{4.137bw})$
Feedback Vertex Set	$\mathcal{O}(n2^{3.134tw})$ [10]	$\mathcal{O}(n2^{3.790bw})$
Hamilton Cycle / TSP	$\mathcal{O}(n2^{3.257tw})$ [2]	$\mathcal{O}(n2^{3.257bw})$

arbitrary real weights, our algorithms even give the advantage of using lower width branch decompositions compared to tree decompositions without the additional cost of a higher constant in the base of the exponent of the running time.

As planar graphs have branchwidth at most $2.122\sqrt{n}$, and such a branch decomposition can be constructed in polynomial time [12] (or we use the ratcatcher algorithm that exactly computes the branchwidth of planar graphs in $\mathcal{O}(n^3)$ time [15, 16]), we can apply our algorithms to solve connectivity problems on planar graphs. This leads to the currently fastest algorithms on planar graphs for several problems improving upon the best known results, due to Dorn [6, 7]. When considering randomised algorithms, we improve the currently fastest algorithms for all considered problems when weights are bounded by $\mathcal{O}(n^{\mathcal{O}(1)})$. When considering determinstic algorithms, we improve the currently fastest algorithms for all considered problems with arbitrary real weights, and the currently fastest algorithm for Hamilton Cycle and TSP when weights are bounded by $\mathcal{O}(n^{\mathcal{O}(1)})$.

2 Preliminaries

Let $G(V, E)$ be a graph with $|V| = n$ vertices and $|E| = m$ edges. For a vertex set $X \subseteq V$ the induced subgraph is denoted by $G[X]$, i.e., $G[X] = G(X, E \cap (X \times X))$. Likewise, the induced subgraph of an edge set $Y \subseteq E$ is denoted as $G[Y]$, i.e., $G[Y] = G(V(Y), Y)$ where $V(Y)$ stands for all endpoints of edges in Y. A cut in a graph is a tuple of two vertex sets (X_1, X_2) for which it holds that $X_1 \cup X_2 = V$ and $X_1 \cap X_2 = \emptyset$.

Throughout the paper the Iverson bracket notation is used. This notation denotes a number that is 1 if the condition between the brackets is satisfied and 0 otherwise, e.g., $[1 = 1]42 = 42$ and $[1 = 2]42 = 0$. We also use this notation in combination with sets S, then this denotes $[True]S = S$ and $[False]S = \emptyset$.

This paper considers dynamic programming algorithms on branch decompositions.

▶ **Definition 1** (Branch decomposition). A branch decomposition of a graph G is a tree $\mathbb{T}$ in which every internal node has degree 3 together with a bijection between the leaves of $\mathbb{T}$ and the edges of G.

As such, every leaf of $\mathbb{T}$ is assigned an edge of G and every edge of G is in exactly one leaf.

Table 4 Comparison of our results on planar graphs with best known results. The column 'Dorn ($n^{\mathcal{O}(1)}$)' states deterministic results by Dorn [6] when weights are $\mathcal{O}(n^{\mathcal{O}(1)})$; the column 'Dorn ($\mathbb{R}$)' states deterministic results by Dorn [7] for arbitrary real weights; the column 'Randomised' states our randomised results when weights are $\mathcal{O}(n^{\mathcal{O}(1)})$; and the column 'Deterministic' states our deterministic results that also apply to arbitrary real weights. We note that the mentioned results by Dorn [6, 7] have not been adjusted for the recently slightly improved matrix multiplication constant ω [14].

Problem	Dorn ($n^{\mathcal{O}(1)}$)	Dorn ($\mathbb{R}$)	Randomised	Deterministic
Planar Steiner Tree	$\mathcal{O}(2^{7.16\sqrt{n}})$	$\mathcal{O}(2^{8.49\sqrt{n}})$	$\mathcal{O}(2^{3.991\sqrt{n}})$	$\mathcal{O}(2^{8.039\sqrt{n}})$
Planar Connected Dom. Set	$\mathcal{O}(2^{8.11\sqrt{n}})$	$\mathcal{O}(2^{9.82\sqrt{n}})$	$\mathcal{O}(2^{5.036\sqrt{n}})$	$\mathcal{O}(2^{8.778\sqrt{n}})$
Planar Feedback Vertex Set	$\mathcal{O}(2^{7.56\sqrt{n}})$	$\mathcal{O}(2^{9.26\sqrt{n}})$	$\mathcal{O}(2^{3.991\sqrt{n}})$	$\mathcal{O}(2^{8.039\sqrt{n}})$
Planar Hamilton Cycle/TSP	$\mathcal{O}(2^{8.15\sqrt{n}})$	$\mathcal{O}(2^{9.86\sqrt{n}})$	$\mathcal{O}(2^{5.036\sqrt{n}})$	$\mathcal{O}(2^{5.63\sqrt{n}})$

The removal of an edge x in a branch decomposition $\mathbb{T}$ divides the edges of G in two parts E_1 and E_2, namely the edges assigned to the leaves of the resulting subtrees T_1 and T_2 of $\mathbb{T}$. For an edge x in $\mathbb{T}$, the associated *middle set* is the vertex subset $B_x \subseteq V$ consisting of all vertices both in $G[E_1]$ and in $G[E_2]$, i.e., $B_e = V_1 \cap V_2$ where V_1 and V_2 are the vertices in $G[E_1]$ and $G[E_2]$, respectively. The *width* assigned to the edge x is the size of the middle set B_x. The width of a branch decomposition $\mathbb{T}$ is the maximum width over all edges of the decomposition, and the branchwidth of a graph G is the minimum width over all possible branch decompositions of G.

To simplify the presentation, we only consider *rooted* branch decompositions. One obtains a rooted branch decomposition by splitting an arbitrary edge (u, v) in the branch decomposition into (u, w) and (w, v), adding a root node r, and adding the edge (w, r). The middle sets of these three edges are defined to be $B_{(u,w)} = B_{(w,v)} = B_{(u,v)}$ and $B_{(w,r)} = \emptyset$. On rooted branch decompositions, we can define a *leaf edge* to be an edge of $\mathbb{T}$ connected to a leaf of $\mathbb{T}$, the *root edge* to be the edge (w, r) to the root r, and an *internal edge* to be any other edge of $\mathbb{T}$. Additionally, for a non-leaf edge x of $\mathbb{T}$, we can now define its *left child* y and *right child* z in $\mathbb{T}$ by ordering the two edges below x in $\mathbb{T}$.

A dynamic programming algorithm on branch decompositions typically computes a table A_x for every edge x of the branch decomposition $\mathbb{T}$ in a bottom-up fashion. Such a table A_x usually contains a set of partial solutions (or the number of partial solutions) on $G[E_x]$ where E_x is the set of the edges assigned to the leaves below the edge x in $\mathbb{T}$. In the case that x is the root edge, the table A_x contains (the number of) complete solutions.

When considering a non-leaf edge x of a branch decomposition $\mathbb{T}$, it is convenient to define a well-known partitioning on the three middle sets involved.

▶ **Definition 2** (Partioning of middle sets)**.** Consider a non-leaf edge x in a branch decomposition $\mathbb{T}$. Let x have left child y and right child z, and let the associated middle sets be B_x, B_y, and B_z. We now define the following partitioning of $B_x \cup B_y \cup B_z$ (see Figure 1):

- *Intersection vertices:* $I = B_x \cap B_y \cap B_z$.
- *Forget vertices:* $F = (B_y \cap B_z) \setminus B_x$.
- *Left vertices:* $L = (B_x \cap B_y) \setminus B_z$.
- *Right vertices:* $R = (B_x \cap B_z) \setminus B_y$.

▶ **Lemma 3** (Constraints on size of middle set partitions)**.** *Given a branch decomposition $\mathbb{T}$ of width bw, the following inequalities on the sizes of the middle-set partitions hold for all non-leaf edges in $\mathbb{T}$:*

- $|I| + |L| + |R| \leq bw$.
- $|I| + |L| + |F| \leq bw$.
- $|I| + |F| + |R| \leq bw$.

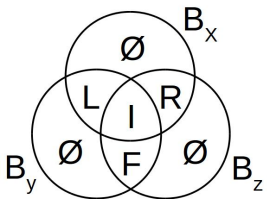


Figure 1 The partitioning of the middle sets.

Finally, to obtain our results on planar graphs, we need the following lemma that relates planar graphs to branch decompositions:

▶ **Lemma 4** (Branch decompositions of planar graphs [8, 12, 15, 16]). *Given a planar graph G, a branch decomposition $\mathbb{T}$ of G of minimal width can be computed in $\mathcal{O}(n^3)$ time. Furthermore, the computed branch decomposition $\mathbb{T}$ has width at most $2.122\sqrt{n}$, and for every non-leaf edge x in $\mathbb{T}$ the middle set partitions satisfy $|I| \leq 2$.*

3 Cut and Count and Branch Decompositions

In this section, we will discuss how to use 'Cut and Count' [5] on branch decompositions. We will illustrate this approach using the *unweighted* variant of the Steiner Tree problem. Our results on other problems use the same ideas, however these proofs are omit due to space restrictions.

The 'Cut and Count' technique of Cygan et al. [5] has two parts, the cut part and the count part. In the cut part, the problem is reformulated and transformed into a counting problem on consistently-cut candidate solutions where the connectivity constraint is relaxed. In the count part, this counting problem is solved using dynamic programming. In this paper, we summarise the cut part for Steiner Tree in Lemma 5 and refer to [5] for more details.

For a subset $X \subseteq V$, Cygan et al. [5] define a *consistent cut* of $G[X]$ to be a cut (X_1, X_2) such that there is no edge (u, v) in $G[X]$ with $u \in X_1$ and $v \in X_2$. Since we consider the unweighted version of Steiner Tree, we can let a solution be a subset of *vertices* $X \subseteq V$ such that $T \subseteq X$ and $G[X]$ is connected. A consistently-cut (possibly disconnected) candidate solution then is a pair $(X, (X_1, X_2))$ consisting of a candidate solution X and a consistent cut (X_1, X_2) of $G[X]$.

▶ **Lemma 5** (based on [5]). *Suppose we are given an algorithm* `Count` *that, given a graph G, a terminal set T, some fixed terminal $t_0 \in T$, and a weight function $w : V \to [0, ..., W]$, computes the values $A(i, w)$ defined below, for all $0 \leq i \leq k$ and $0 \leq w \leq kW$:*

$$A(i,w) = \left| \left\{ (X,(X_1,X_2)) \;\middle|\; \begin{array}{l} X \subseteq V, (X_1,X_2) \text{ a consistent cut of } G[X], \\ T \subseteq X, t_0 \in X_1, |X| = i, w(X) = w \end{array} \right\} \right| \pmod 2$$

Then, there exists a Monte-Carlo algorithm that solves Steiner Tree *on G, that cannot give false-positives and may give false negatives with probability at most $1/2$. The running time of this algorithm is dominated by the running time of the* `Count` *algorithm with $W = \mathcal{O}(n)$.*

We will omit the modulo two in the description of our counting algorithms and take the modulus afterwards, doing all computations modulo two requires slightly less time and space.

For easier exposition, we first prove the following theorem. Next, we will improve this using fast matrix multiplication in Theorem 7.

▶ **Theorem 6.** *There exist a Monte-Carlo algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw, solves* Steiner Tree *in time $\mathcal{O}(3^{\frac{3}{2}bw}n^{\mathcal{O}(1)})$.*

Proof. The result follows from Lemma 5 if we can give an algorithm that computes the required values $A(i,w)$ in $\mathcal{O}(3^{\frac{3}{2}bw}n^{\mathcal{O}(1)})$ time. We give this algorithm below.

We compute $A(i,w)$ by bottom-up dynamic programming on the branch decomposition $\mathbb{T}$. For each edge x of $\mathbb{T}$, we count partial-solution-cut pairs $(X,(X_1,X_2))$, where we call X a partial solution in $G[E_x]$ if all terminals in $G[E_x]$ are in X, and where the cut (X_1,X_2) is a consistent cut of the subgraph of $G[E_x]$ induced by X (i.e., a cut in $(G[E_x])[X]$) with additionally that if $t_0 \in X$ then $t_0 \in X_1$. To count these pairs, we define a labelling using labels 0, 1_1 and 1_2 on the vertices in the middle set B_x associated to an edge x of $\mathbb{T}$. These labels identify the situation of the vertex in a partial-solution-cut pair $(X,(X_1,X_2))$: label 0 means not in X, and labels 1_1 and 1_2 mean in X and on side X_1 and X_2 of the cut, respectively.

In a bottom-up fashion, we associate to each edge x of $\mathbb{T}$ a table $A_x(i,w,s)$ with entries for all $0 \le i \le k$, $0 \le w \le kW$, and $s \in \{0,1_1,1_2\}^{B_x}$. Such an entry $A_x(i,w,s)$ counts the number of partial-solution-cut pairs $(X,(X_1,X_2))$ as defined above that satisfy the constrains imposed by the states s on B_x and that satisfy $|X| = i$ and $w(X) = w$.

For a leaf edge x of the branch decomposition $\mathbb{T}$, we have that $B_x = \{u,v\}$ for some edge (u,v) in E. The table A_x associated to x can be filled as follows (all other entries are zero):

$$\begin{aligned}
A_x(0,0,0\ 0) &= 1[u \notin T \wedge v \notin T]\\
A_x(1,w(u),1_1\ 0) &= 1[v \notin T]\\
A_x(1,w(v),0\ 1_1) &= 1[u \notin T]\\
A_x(1,w(u),1_2\ 0) &= 1[u \neq t_0 \wedge v \notin T]\\
A_x(1,w(v),0\ 1_2) &= 1[u \notin T \wedge v \neq t_0]\\
A_x(2,w(u)+w(v),1_1\ 1_1) &= 1\\
A_x(2,w(u)+w(v),1_2\ 1_2) &= 1[u \neq t_0 \wedge v \neq t_0]
\end{aligned}$$

Here, we enforce that the cut is consistent, that every terminal $t \in T$ is in the partial solution X, that t_0 is on the correct side of the cut ($t_0 \in X_1$), and that $|X| = i$ and $w(X) = w$.

For an internal edge x of the branch decomposition $\mathbb{T}$ with children y and z, we fill the table A_x by combining the counted number of partial-solution-cut pairs from the tables for y and z. For this, we say that labellings s_x of B_x, s_y of B_y, and s_z of B_z are compatible if and only if $s_x^L = s_y^L \wedge s_x^R = s_z^R \wedge s_y^F = s_z^F \wedge s_x^I = s_y^I = s_z^I$ (where we denote by s_x^L the labelling s_x restricted to middle set partition L; for the middle set partitions see Definition 2).

We fill A_x by means of the following formula, where i^Z denotes the number of vertices with state 1 in middle set partition Z, and w^Z denotes the sum of the weights of the vertices with state 1 in middle set partition Z (for Z equals F, or I):

$$A_x(i_x,w_x,s_x) = \sum_{\substack{s_x,\,s_y,\,s_z\\ \text{compatable}\\ \text{labellings}}} \sum_{i_x = i_y + i_z - i^I - i^F} \sum_{w_x = w_y + w_z - w^I - w^F} A_y(i_y,w_y,s_y)\cdot A_z(i_z,w_z,s_z)\,.$$

This counts the total number of partial-solution-cut pairs $(X,(X_1,X_2))$ that satisfy the constraints as the summations combine all compatible entries from A_y and A_z and the multiplication combines the individual counts. To see that exactly these entries are compatible,

note that the consistency of the cut, the fact that $T \subseteq X$, and that $t_0 \in X_1$ are all enforced at the leaves and maintained by enforcing compatible labels. Furthermore, the partial-solution size i and weight w is the sum of both underlying partial solutions minus the doubling on the middle set partitions F and I.

By computing A_x for all edges in the branch decomposition $\mathbb{T}$ in the above way, we can find the required values $A(i, w)$ at the root edge r of $\mathbb{T}$ where $B_r = \emptyset$.

Consider the time required for computing table A_x. This table has at most $3^{|L|}3^{|R|}3^{|I|}k^2W$ entries, and for each entry we have to inspect at most $3^{|F|}k^2W$ combinations of entries from A_y and A_z, thus requiring $\mathcal{O}(3^{|L|+|R|+|I|+|F|}k^4n^2)$ time using $W = \mathcal{O}(n)$. This leads to a worst-case running time of $\mathcal{O}(3^{\frac{3}{2}bw}n^{\mathcal{O}(1)})$ under the constraints in Lemma 3. ◀

This result can be improved by using fast matrix multiplication similar to Dorn et al. [6].

▶ **Theorem 7.** *There exist a Monte-Carlo algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw, solves* Steiner Tree *in time $\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$, where ω is the matrix multiplication exponent.*

Proof. The algorithm is similar to the proof of Theorem 6, however, we evaluate the formula for the table $A_i(i_x, w_x, s_x)$ associated to an internal edge of the branch decomposition in a more efficient way. Instead of first fixing all labellings, we now first only fix compatible a labelling on I and fix i_x, i_y, w_x and w_y. Then, we can compute the contribution to $A_i(i_x, w_x, s_x)$, given the fixed values and fixed partial state, for all compatible states $s_x \in \{0, 1_1, 1_2\}^{B_x}$ using a single matrix multiplication.

To do so, we construct two matrices B and C. In matrix B there is a row for each labelling of L and a column for each labelling of F, and in matrix C there is a row for each labelling of F and a column for each labelling of R. As the labellings on I are fixed, each entry in B can be associated to a full labelling of the middle set B_y, and each entry in C can be associated to a complete labelling of the middle set B_z. Moreover, each entry in the matrix product BC can be associated to a full labelling of B_x, corresponding to the row of B (labelling of L) and column of C (labelling of R). If we fill matrix B with the corresponding values $A_y(i_y, w_y, s_y)$, and matrix C with the corresponding values $A_z(i_z, w_z, s_z)$ (note that we did not fix i_z and w_z, but these follow from all other fixed values and labellings), then matrix BC holds the contribution to $A_x(i_x, w_x, s_x)$ given the fixed labellings and values.

In the above way, we perform $3^{|I|}k^2W^2$ matrix multiplications of a $3^{|L|} \times 3^{|F|}$ matrix and a $3^{|F|} \times 3^{|R|}$ matrix. These rectangular matrices can be multiplied in $\mathcal{O}(3^{(\omega-1)|L|}3^{|F|}n^{\mathcal{O}(1)})$ time (see also [6, 13]), where we use that we can assume $|L| = |R|$ in a worst-case analysis for symmetry reasons. Under the constraints of Lemma 3, the worst-case arises when $|L| = |R| = |F| = \frac{1}{2}bw$ resulting in a running time of $\mathcal{O}(3^{\frac{\omega}{2}bw}n^{\mathcal{O}(1)})$. ◀

▶ **Corollary 8.** *There exist a Monte-Carlo algorithm that, given a planar graph G, solves* Planar Steiner Tree *in time $\mathcal{O}(2^{3.991\sqrt{n}})$.*

Proof. Combine Theorem 7 with Lemma 4 and use $\omega < 2.373$ [14]. ◀

The other randomised results in Table 1 follow in a similar fashion and are omitted due to space restrictions. However, for some problems we need to use (generalised forms of) subset convolution to obtain the claimed time bounds. For the generalised subset convolution, we refer the reader to [18], and for an exposition on how to apply this in the setting of branch decompositions to [3].

4 Representative Sets and Branch Decompositions

In this section, we will discuss how to use the *rank-based approach* based on representative sets and Gaussian elimination [2] on branch decompositions. We will illustrate this approach using the *weighted* variant of the Steiner Tree problem. Our results on other problems use the same ideas, however the proofs are omitted due to space restrictions.

We need some definitions and notion regarding partitions. The set of all partitions of a set U is denoted by $\Pi(U)$. An element of a partition is also called a block. For $p \in \Pi(U)$, the term $|p|$ denotes the amount of blocks in the partition, where we let the empty partition in $\Pi(\emptyset)$ have zero blocks. For $p, q \in \Pi(U)$, $p \sqcup q$ is obtained from p and q by iteratively merging blocks in p that contain elements that are in the same block in q and vice versa. Also, $p \sqcap q$ is the partition that contains all blocks that are a non-empty intersection of a block in p and a block in q. If $X \subseteq U$, then $p_{\downarrow X} \in \Pi(X)$ is formed by removing all elements not in X from the partition p and possibly removing empty blocks. In the same way, if $U \subseteq X$, then $p_{\uparrow X} \in \Pi(X)$ is formed by adding a singleton to p for every element in $X \setminus U$.

A set of weighted partitions over U is a set $\mathcal{F} \subseteq (\Pi(U) \times \mathbb{N})$, i.e., a set of pairs consisting of a partition of U and a non-negative integer that is the weight of the partition. We use the following operators from [2] on a set of weighted partitions $\mathcal{F} \subseteq (\Pi(U) \times \mathbb{N})$:

- **Remove:** Define $\texttt{rmc}(\mathcal{F}) = \{(p, w) \in \mathcal{F} \mid \nexists (p, w') \in \mathcal{F} \wedge w' < w\}$. This operator removes non-minimal weight copies.
- **Union:** For $\mathcal{G} \subseteq (\Pi(U) \times \mathbb{N})$, define $\mathcal{F} \uplus \mathcal{G} = \texttt{rmc}(\mathcal{F} \cup \mathcal{G})$. This operator combines the two sets of weighted partitions and discards non-minimal weight copies.
- **Project:** For $X \subseteq U$, let $\bar{X} = U \setminus X$ and define $\texttt{proj}(X, \mathcal{F}) \subseteq \Pi(\bar{X}) \times \mathbb{N}$ as

$$\texttt{proj}(X, \mathcal{F}) = \texttt{rmc}(\{(p_{\downarrow \bar{X}}, w) \mid (p, w) \in \mathcal{F}, |p_{\downarrow \bar{X}}| = |p| \vee (X = \emptyset \wedge |p| = 1)\}) .$$

  This operator removes all elements from X from each partition and discards a partition if the amount of blocks in it decreases because of this, unless there is only one partition which is projected upon the empty set.
- **Join:** For a set U' and $\mathcal{G} \subseteq \Pi(U')$, let $\hat{U} = U \cup U'$, we define any pair of partitions $(p, w_1) \in \mathcal{F}$, $(q, w_2) \in \mathcal{G}$ to be compatible, unless (p, w_1) or (q, w_1) is the empty partition with non-zero weight. In that case, the pair is compatible, if and only if, the other partition is the empty partition with zero weight.
  Now we define $\texttt{join}(\mathcal{F}, \mathcal{G}) \subseteq (\Pi(\hat{U}) \times \mathbb{N})$ as:

$$\texttt{join}(\mathcal{F}, \mathcal{G}) = \texttt{rmc}(\{(p_{\uparrow \hat{U}} \sqcup q_{\uparrow \hat{U}}, w_1 + w_2) \mid (p, w_1) \in \mathcal{F}, (q, w_2) \in \mathcal{G} \text{ compatible}\}) .$$

  This operator extends all partitions to the same base set $\hat{U}$. It then combines each compatible pair of partitions by means of the $\sqcup$ operator and assigns the sum of the weights as a new weight. We will need pairs to be compatible to keep solutions connected.

We will start by giving a naive algorithm for weighted Steiner Tree on branch decompositions. Thereafter, we will show how to use representative sets and Gaussian elimination to improve the time complexity. We note that, different from Section 3, we now let (partial) solutions be sets of *edges* connecting the terminals T.

The Naive Algorithm for Steiner Tree on Branch Decompositions

In a bottom-up fashion, the naive algorithm computes a table A_x for each edge x of the branch decomposition $\mathbb{T}$. This table keeps track of all possible partial solutions $Y \subseteq E_x$ on $G[E_x]$ that can be extended to a minimal weight solution on G. These partial solutions

are subsets $Y \subseteq E_x$ such that all terminals in $G[E_x]$ are incident to an edge in Y, and all connected components in $G[Y]$ either contain an edge incident to B_x or connect all terminals T in G.

Each entry $A_x(s)$ in the table is indexed by a labelling $s \in \{0,1\}^{B_x}$ on the vertices in B_x and contains a *set* of weighted partitions. The label 1 means that the vertex will be incident to the solution edge set, which is the case when the vertex is a terminal or when the vertex is incident to an edge in the partial solution Y. The label 0 means that it will not be incident to the solution edge set. The set of weighted partitions $A_x(s)$ is a set of weighted partitions on all vertices with label 1 in s. $A_x(s)$ represents all partial solutions on $G[E_x]$ consistent with the labelling s in the following way: the weight of the partition corresponds to the weight of the partial-solution Y; and vertices are in the same block of the partition p that represents that solution Y, if and only if, the vertices are in the same connected component in $G[Y]$.

For a leaf edge x of the branch decomposition $\mathbb{T}$, we have that $B_x = \{u, v\}$ for an edge (u, v) in E. The table A_x associated to x can be filled as follows:

$$A(0\ 0) = \{(\emptyset, 0)\}[u \notin T \wedge v \notin T]$$
$$A(1\ 0) = \{(\{\{u\}\}, 0)\}[v \notin T]$$
$$A(0\ 1) = \{(\{\{v\}\}, 0)\}[u \notin T]$$
$$A(1\ 1) = \{(\{\{u\}, \{v\}\}, 0), \{\{u\ v\}\}, w((u, v)))\}$$

Here, we make sure that terminal vertices in T correspond to 1 labels, and that vertices incident to an edge in the partial solution correspond to 1 labels. We also make sure that the partition corresponds to the connected components on the vertices with a 1 label, and that the weight of the partition equals the weight of the partial solution.

For an internal edge x of the branch decomposition $\mathbb{T}$ with children y and z, we fill the table A_x by means of the following formula:

$$A_x(s_x) = \biguplus_{s_F \in \{0,1\}^F} \texttt{proj}\big(F, \texttt{join}(A_y(s_x^L s_x^I s_F), A_z(s_x^R s_x^I s_F))\big)\,.$$

Here $s_x^L s_x^I s_F$ stands for the concatenation of the labelling s_x restricted to L, the labelling s_x restricted to I, and the labelling s_F on F (note that this gives a valid labelling on B_y).

For every labelling s_F on F, the above formula combines all entries with compatible weighted partitions from $A_y(s_x^L s_x^I s_F)$ and $A_z(s_x^R s_x^I s_F)$. Partitions in the computed set $A_x(s)$ now correspond to the connected components of the partial solution, by definition of the `join` and `proj` operations. This is because, the resulting entries in which vertices in F are in separate blocks (separate connected components in $G[E_x]$) are discarded by the project operation. Also, we require compatible weighted partitions in the join operation to make sure that no connected components that do not contain vertices in B_x are combined, i.e., these do not result in new non-empty partitions. The weights of the partitions in $A_x(s)$ correspond to the weights of the partial solutions, as we choose edges from G in a partial solution in leaf edges of the branch decomposition and the `join` operation sums up the weights.

By computing A_x for all edges in the branch decomposition $\mathbb{T}$ in the above way, we can find the weight of the minimum weight solution to Steiner Tree at the root edge r of $\mathbb{T}$ where $B_r = \emptyset$ as the weight of the empty partition.

Using Representative Sets

The essence of the rank-based approach lies in the `Reduce` procedure from [2]. This procedure reduces the size of the tables used in the dynamic program without loss of representation.

▶ **Definition 9** (Representation [2]). For sets of weighted partitions $\mathcal{F}, \mathcal{F}' \subseteq (\Pi(U) \times \mathbb{N})$ and a partition $q \in \Pi(U)$, define:

$$\texttt{opt}(q, \mathcal{F}) = \min\{w \mid (p, w) \in \mathcal{F} \wedge p \sqcup q = \{U\}\}.$$

We say that $\mathcal{F}'$ *represents* $\mathcal{F}$, if for all $q \in \Pi(U)$, it is the case that $\texttt{opt}(q, \mathcal{F}') = \texttt{opt}(q, \mathcal{F})$.

▶ **Theorem 10** ([2]). *There exists an algorithm* `Reduce` *that, given a set of weighted partitions $\mathcal{F} \subseteq (\Pi(U) \times \mathbb{N})$, outputs a set of weighted partitions $\mathcal{F}' \subseteq \mathcal{F}$, such that $\mathcal{F}'$ represents $\mathcal{F}$ and $|\mathcal{F}'| \leq 2^{|U|-1}$, in $\mathcal{O}(|\mathcal{F}|2^{(\omega-1)|U|}|U|^{\mathcal{O}(1)})$ time.*

We apply the above theorem at each step of the naive algorithm for Steiner Tree and carefully analyse the resulting running time to obtain the following result.

▶ **Theorem 11.** *There exist an algorithm that, given a graph G and a branch decomposition $\mathbb{T}$ of G of width bw, solves* Steiner Tree *in time $\mathcal{O}(n((1+2^\omega)\sqrt{5})^{bw} bw^{\mathcal{O}(1)})$.*

Proof. The algorithm computes the tables A_x in a bottom-up fashion over the branch decomposition $\mathbb{T}$ according to the formulae in the description of the naive algorithm. Directly after the algorithm finishes computing a table A_x for any edge x in the branch decomposition, the `Reduce` algorithm is applied to each entry $A_x(s_x)$ of the table to control the sizes of the sets of weighted partitions. Because the naive algorithm is correct and the `Reduce` procedure maintains representation (Theorem 10), we conclude that the new algorithm is correct also.

To prove the running time, consider a non-leaf edge x in the branch decomposition $\mathbb{T}$ with left child y and right child z. The operations in the naive algorithm used to compute, for a labelling $s_x \in \{0,1\}^{B_x}$, the set of weighted partitions $A_x(s_x)$ can be implemented in $\mathcal{O}(bw^{\mathcal{O}(1)})$ time times the number of combinations of entries from A_y and A_z involved. This can be done using the straightforward implementations (see also [2]). As each combination of entries from A_y and A_z can lead to an entry in $A_x(s_x)$ before the `Reduce` step is applied, the running time is dominated by the time required by the `Reduce` algorithm.

For a fixed $s_x \in \{0,1\}^{B_x}$, let j be the amount of vertices in s_x with label 1, which we will also denote by $j = |s_x^{-1}(1)|$. For the set of weighted partitions $A_x(s_x)$, `Reduce` takes time:

$$\mathcal{O}\big(|A_x(s_x)|2^{(\omega-1)j} j^{\mathcal{O}(1)}\big).$$

The size of $A_x(s_x)$ is the result of combining, for every labelling $s_F \in \{0,1\}^F$, every entry of $A_y(s_x^L s_x^I s_F)$ with every entry of $A_z(s_x^R s_x^I s_F)$. Using $s_y = s_x^L s_x^I s_F$ and $s_z = s_x^R s_x^I s_F$, the sizes of $A_y(s_y)$ and $A_z(s_z)$ are bounded by $2^{|s_y^{-1}(1)|}$ and $2^{|s_z^{-1}(1)|}$, respectively, since these table were reduced after computing A_y and A_z. Therefore, the total time it takes to reduce the sets of partitions for all entries in $A_x$is:

$$\mathcal{O}\Big(\sum_{j=0}^{|I\cup R\cup L|} \binom{|I \cup R \cup L|}{j} 2^{(\omega-1)j} |A_x(s_j)| j^{\mathcal{O}(1)}\Big).$$

The sum and the binomial coefficient consider all possible labellings using j for the number of 1 labels. This is the only information needed about the labellings. As such, we will slightly abuse notation and denote any labelling with j vertices with label 1 as s_j. Also, we will denote by $s_{i,l,f}$ any labelling with i vertices with label 1 on I, l vertices with label 1 on L, and f vertices with label 1 on F.

We can now expand the sum, differentiating between I, L and R, and use that $A_y(s_y)$ and $A_z(s_z)$ are bounded by $2^{|s_y^{-1}(1)|}$ and $2^{|s_z^{-1}(1)|}$, respectively:

$$\mathcal{O}\Big(\sum_{j=0}^{|I\cup R\cup L|}\binom{|I\cup R\cup L|}{j}2^{(\omega-1)j}|A_x(s_j)|j^{\mathcal{O}(1)}\Big) =$$
$$\mathcal{O}\Big(\sum_{i=0}^{|I|}\sum_{r=0}^{|R|}\sum_{l=0}^{|L|}\binom{|I|}{i}\binom{|R|}{r}\binom{|L|}{l}2^{(\omega-1)(i+r+l)}|A_x(s_{i,r,l})|(i+r+l)^{\mathcal{O}(1)}\Big) =$$
$$\mathcal{O}\Big(\sum_{i=0}^{|I|}\sum_{r=0}^{|R|}\sum_{l=0}^{|L|}\binom{|I|}{i}\binom{|R|}{r}\binom{|L|}{l}2^{(\omega-1)(i+r+l)}\sum_{f=0}^{|F|}\binom{|F|}{f}|A_y(s_{i,l,f})||A_z(s_{i,r,f})|bw^{\mathcal{O}(1)}\Big) \leq$$
$$\mathcal{O}\Big(\sum_{i=0}^{|I|}\sum_{r=0}^{|R|}\sum_{l=0}^{|L|}\binom{|I|}{i}\binom{|R|}{r}\binom{|L|}{l}2^{(\omega-1)(i+r+l)}\sum_{f=0}^{|F|}\binom{|F|}{f}2^{i+l+f}2^{i+r+f}bw^{\mathcal{O}(1)}\Big).$$

Next, we rearrange the terms and repeatedly apply the binomial theorem to obtain a more simple expression:

$$\mathcal{O}\Big(\sum_{i=0}^{|I|}\binom{|I|}{i}2^{(\omega+1)i}\sum_{r=0}^{|R|}\binom{|R|}{r}2^{\omega r}\sum_{l=0}^{|L|}\binom{|L|}{l}2^{\omega l}\sum_{f=0}^{|F|}\binom{|F|}{f}2^{2f}bw\Big)^{\mathcal{O}(1)}\Big) \leq$$
$$\mathcal{O}\big((1+2^{\omega+1})^{|I|}(1+2^{\omega})^{|R|}(1+2^{\omega})^{|L|}5^{|F|}bw^{\mathcal{O}(1)}\big).$$

If we maximize this under the constraints in Lemma 3, then we find a worst-case running time of:

$$\mathcal{O}(((1+2^{\omega})\sqrt{5})^{bw}bw^{\mathcal{O}(1)}).$$

In this case $|R| = |L| = |F| = \frac{1}{2}bw$ and $|I| = 0$. Taking into consideration that this must be done for every edge in the branch decomposition, we find the time-complexity from the statement of theorem. ◀

The other deterministic result in Table 1 follow in a similar fashion but are omitted due to space restrictions. These omitted proofs, use besides (generalised forms of) subset convolution, also the fact that the `Reduce` procedure can be modified to output a set of weighted partitions of size at most $2^{|U|/2}$ in case of the Hamilton Cycle and TSP problems [4].

5 Conclusion

In this paper, we have shown two things. First of all, we have shown that cut and count and the rank-based approach can be used not only on tree decompositions but also on branch decompositions. This means the techniques are more powerful than they were known to be. Perhaps these techniques can also be used in combination with other width measures.

We have also given fast algorithms, especially on planar graphs, for several connectivity problems. These algorithms use branch decompositions and therefore affirm the use of this type of decomposition as a solid foundation for algorithms.

References

1 Hans L. Bodlaender. Treewidth: Structure and algorithms. In *Proceedings of the 14th International Colloquium on Structural Information and Communication Complexity, SIROCCO 2007*, volume 4474 of *Lecture Notes in Computer Science*, pages 11–25. Springer Verlag, 2007.

2 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015.

3 Hans L. Bodlaender, Erik Jan van Leeuwen, Johan M. M. van Rooij, and Martin Vatshelle. Faster algorithms on branch and clique decompositions. In *Proceedings of the 35th International Symposium on Mathematical Foundations of Computer Science, MFCS 2010*, volume 6281 of *Lecture Notes in Computer Science*, pages 174–185. Springer Verlag, 2010.

4 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity checking via bases of perfect matchings. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 301–310. ACM, 2013.

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

6 Frederic Dorn. Dynamic programming and fast matrix multiplication. In *Proceedings of the 14th Annual European Symposium on Algorithms, ESA 2006*, volume 4168 of *Lecture Notes in Computer Science*, pages 280–291. Springer Verlag, 2006.

7 Frederic Dorn. *Designing Subexponential Algorithms: Problems, Techniques & Structures.* PhD thesis, Institutt for informatikk, Universitetet i Bergen, 2007.

8 Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. Efficient exact algorithms on planar graphs: exploiting sphere cut decompositions. *Algorithmica*, 58:790–810, 2010.

9 Stefan Fafianie, Hans L. Bodlaender, and Jesper Nederlof. Speeding up dynamic programming with representative sets: an experimental evaluation of algorithms for steiner tree on tree decompositions. *Algorithmica*, 71(3):636–660, 2015.

10 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Representative sets of product families. In *Algorithms – ESA 2014 – 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 443–454. Springer, 2014.

11 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 142–151, 2014.

12 Fedor V. Fomin and Dimitrios M. Thilikos. New upper bounds on the decomposability of planar graphs. *Journal of Graph Theory*, 51:53–81, 2006.

13 François Le Gall. Faster algorithms for rectangular matrix multiplication. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 514–523. IEEE Computer Society, 2012.

14 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC*, pages 296–303, 2014.

15 Qian-Ping Gu and Hisao Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ time. *ACM Trans. Algorithms*, 4(3), 2008.

16 Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

17 Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Discr. Math.*, 10:529–550, 1997.

18 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Proceedings of the 17th Annual European Symposium on Algorithms, ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer Verlag, 2009.

A Fast Parameterized Algorithm for Co-Path Set*

Blair D. Sullivan[1] and Andrew van der Poel[2]

1 **Dept. of Computer Science, North Carolina State University, Raleigh, USA**
`blair_sullivan@ncsu.edu`
2 **Dept. of Computer Science, North Carolina State University, Raleigh, USA**
`ajvande4@ncsu.edu`

Abstract

The k-Co-Path Set problem asks, given a graph G and a positive integer k, whether one can delete k edges from G so that the remainder is a collection of disjoint paths. We give a linear-time, randomized fpt algorithm with complexity $O^*(1.588^k)$ for deciding k-Co-Path Set, significantly improving the previously best known $O^*(2.17^k)$ of Feng, Zhou, and Wang (2015). Our main tool is a new $O^*(4^{tw(G)})$ algorithm for Co-Path Set using the Cut&Count framework, where $tw(G)$ denotes treewidth. In general graphs, we combine this with a branching algorithm which refines a $6k$-kernel into reduced instances, which we prove have bounded treewidth.

1998 ACM Subject Classification G.2.2 Graph Algorithms

Keywords and phrases co-path set, parameterized complexity, treewidth, fixed-parameter tractable complexity

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.28

1 Introduction

We study parameterized versions of Co-Path Set [3, 16], an NP-complete problem asking for the minimum number of edges whose deletion from a graph results in a collection of disjoint paths (the deleted edges being a *co-path set* – see Figure 1). Specifically, we are concerned with k-Co-Path Set, which uses the natural parameter of the number of edges deleted.

> k-Co-Path Set
> **Input:** A graph $G = (V, E)$ and a non-negative integer k.
> **Parameter:** k
> **Problem:** Does there exist $F \subseteq E$ of size exactly k such that $G[E \setminus F]$ is a set of disjoint paths?

These problems are naturally motivated by determining the ordering of genetic markers in DNA using fragment data created by breaking chromosomes with gamma radiation (a technique known as *radiation hybrid mapping*) [4, 13, 15]. Unfortunately, human error in distinguishing markers often means the constraints implied by markers' co-occurrence on fragments are incompatible with all possible linear orderings, necessitating an algorithm to find the "best" ordering (that violates the fewest constraints). Co-Path Set solves the

* This work supported in part by the Gordon & Betty Moore Foundation under DDD Investigator Award GBMF4560 and the DARPA GRAPHS program under SPAWAR Grant N66001-14-1-4063. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of DARPA, SSC Pacific, or the Moore Foundation.

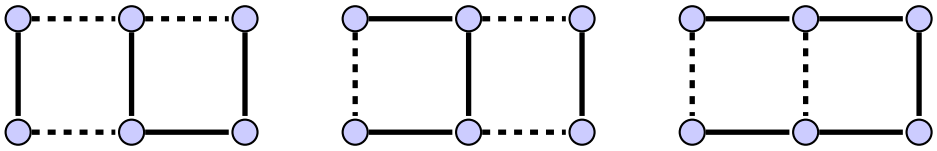


Figure 1 Three co-path sets (dashed edges), including one of minimum size (rightmost).

special case where each DNA fragment contains exactly two genetic markers (corresponding to an edge in the graph); any linear ordering of the markers must correspond to some set of paths, and we minimize the number of unsatisfied constraints (edges in the co-path set).

Recent algorithmic results related to Co-Path Set include a (10/7)-approximation algorithm [2], and two parameterized algorithms deciding k-Co-Path Set [6, 7], the faster of which [7] has time complexity[1] $O^*(2.17^k)$. However, as written, both parameterized results [6, 7] contain a flaw in their analysis which invalidates their probability of a correct solution in the given time[2]. The best known bound prior to [6] is an $O^*(2.45^k)$ algorithm [16]. In this paper, we prove:

▶ **Theorem 1.** *k-Co-Path Set is decidable in $O^*(1.588^k)$ linear-fpt time with probability at least 2/3.*

We note that standard amplification arguments apply, and Theorem 1 holds for any success probability less than 1. Further, if f is an increasing function with $\lim_{n\to\infty} f(n) = 1$, we can solve k-Co-Path Set with success probability at least $f(n)$ in $O(1.588^k n\mathsf{polylog}(n))$.

The remainder of this paper is organized as follows: after essential definitions and notation in Section 2, we start in Section 3 by giving a new $O^*(4^{tw(G)})$ algorithm `tw-copath` for solving Co-Path Set parameterized by treewidth (tw) using the Cut&Count framework [5]. Finally, Section 4 describes the linear-fpt algorithm referenced in Theorem 1, which solves k-Co-Path Set on general graphs in $O^*(1.588^k)$ by applying `tw-copath` to a set of "reduced instances" generated via kernelization and a branching procedure[3] `deg-branch`.

2 Preliminaries

Let $G(V, E)$ be the graph with vertex set V and edge set E. Unless otherwise noted, we assume $|V| = n$ and $|E| = m$; we let $N(v)$ denote the set of neighbors of a vertex v, and let $\deg(v) = |N(v)|$. Given a graph $G(V, E)$ and $F \subseteq E$, we write $G[F]$ for the graph $G(V, F)$.

Our `tw-copath` algorithm in Section 3 uses dynamic programming over a *tree decomposition*, and its running time depends on the related measure of *treewidth* [14], which we denote $tw(G)$. To simplify the dynamic programming, we will use a variant of *nice tree decompositions* [10, 5] where each node in the tree has one of five specific types: leaf, introduce

[1] Throughout this paper, we use the notation $O^*(f(k))$ for the fpt (fixed-parameter tractable) complexity $O(f(k)n^{O(1)})$; we say an algorithm is *linear-fpt* if the complexity is $O(f(k)n)$.

[2] Step 2.11 in both versions of Algorithm R-MCP checks if a candidate co-path set F has size $\leq k_1$ (as they are sweeping over all possible sizes of candidates and want to restrict the size accordingly). If F is too large, the algorithm discards it and continues to the next iteration. However, in order for their analysis to hold, the probability that the candidate is contained in a co-path set must be $\geq (1/2.17)^{k_1}$ (or $(1/2.29)^{k_1}$ in [6]) for *every* iteration. Candidates which are too large may have significantly smaller probability of containment, yet are counted in the exponent of the analysis.

[3] The properties of our reduced instances guarantee we can find a tree decomposition of small width in poly(k) time.

vertex, introduce edge, forget vertex, or join. The "introduce edge" nodes are labelled with an edge uv and have one child (with an identical bag); we require that each edge in E is introduced exactly once. Additionally, we enforce that the root node is of type "forget vertex" (and thus has an empty bag). A tree decomposition can be transformed into a nice decomposition of the same width in time linear in the size of the input graph [5].

When describing the dynamic programming portion of the algorithm we use Iverson's bracket notation: if p is a predicate we let $[\![p]\!]$ be 1 if p is true and 0 otherwise. We also use the shorthand $f[x \to y]$ to denote updating a function f so that $f(x) = y$ and all other values are unchanged.

Finally, we use fast subset convolution [1] to reduce the complexity of handling join nodes in the nice tree decomposition (Section 3). This technique maps functions of the vertices in a join bag to values in $\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$ (where p is chosen based on the application). The key complexity result we rely on uses the $\mathbb{Z}_p$ product, which is defined below. We write $\mathbb{Z}_p^B$ for the set of all vectors t of length $|B|$ assigning a value $t(b) \in \mathbb{Z}_p$ to each element of $b \in B$.

▶ **Definition 2** ($\mathbb{Z}_p$ product)**.** Let $p \geq 2$ be a fixed integer and let B be a finite set. For $t_1, t_2, t \in \mathbb{Z}_p^B$ we say that $t_1 + t_2 = t$ if $t_1(b) + t_2(b) = t(b)$ (in $\mathbb{Z}_p$) for all $b \in B$. For a ring R and functions $f, g : \mathbb{Z}_p^B \to R$, define the $\mathbb{Z}_p$ product, $*_x^p$ as

$$(f *_x^p g)(t) = \sum_{t_1+t_2=t} f(t_1)g(t_2)\,.$$

Fast subset convolution guarantees that certain $\mathbb{Z}_p$ products can be computed quickly.

▶ **Lemma 3** (Cygan et al. [5])**.** *Let $R = \mathbb{Z}$ or $R = \mathbb{Z}_q$ for some constant q. The $\mathbb{Z}_4$ product of functions $f, g : \mathbb{Z}_4^B \to R$ can be computed in $4^{|B|}|B|^{O(1)}$ time and ring operations.*

3 An $O^*(4^{tw(G)})$ Algorithm via Cut&Count

We start by giving an fpt algorithm for Co-Path Set parameterized by treewidth. Our primary tool is the Cut&Count framework, which enables $c^{tw}n^{O(1)}$ one-sided Monte Carlo algorithms for connectivity-type problems with constant probability of a false negative. Cut&Count has previously been used to improve the best-known bounds for several well-studied problems, including Connected Vertex Cover, Hamiltonian Cycle, and Feedback Vertex Set [5]. Pilipczuk showed that an $O^*(c^{tw})$ algorithm for some constant c can be designed with the Cut&Count approach for Co-Path Set because the problem can be expressed in the specialized graph logic known as ECML+C [12]. However, since our end goal is to improve on existing algorithms for k-Co-Path Set in general graphs using a bounded treewidth kernel, we need to develop a specialized dynamic programming algorithm with a small value of c. We show:

▶ **Theorem 4.** *There exists a one-sided fpt Monte Carlo algorithm* `tw-copath` *deciding k-Co-Path Set for all k in a graph G in $O^*(4^{tw(G)})$ time with failure probability $\leq 1/3$, when a tree decomposition of width tw is given as input.*

The Cut&Count technique has two main ingredients: an algebraic approach to counting which uses arithmetic in $\mathbb{Z}_2$ (enabling faster algorithms) alongside a guarantee that undesirable objects are seen an even number of times (so a non-zero result implies a desired solution has been seen), and the idea of defining the problem's connectivity requirement through consistent cuts. In this context, a *consistent cut* is a partitioning (V_1, V_2) of the vertices of a graph into two sets such that no edge uv has $u \in V_1$ and $v \in V_2$ and all vertices of degree 0

are in V_1 . Since each connected component must lie completely on one side of any consistent cut, we see that a graph G has exactly $2^{cc(G)-n_I(G)}$ such cuts, where $cc(G)$ is the number of connected components and $n_I(G)$ is the number of isolates (vertices with degree 0). In order to utilize parity with the number of consistent cuts, we introduce *markers*, which create even numbers of consistent cuts for graphs that are not collections of disjoint paths. Our counting algorithm `tw-copath`, which computes the parity of the size of the collection of subgraphs with consistent cuts which adhere to specific properties pertaining to Co-Path Set, employs dynamic programming over a nice tree decomposition. We further use weights and the Isolation Lemma to bound the probability of a false negative arising from multiple valid markings of a solution. We use fast subset convolution [1] to reduce the complexity required for handling join bags in the dynamic programming. In the remainder of this section, we present the specifics for applying these techniques to solve Co-Path Set.

3.1 Cutting

We first provide formal definitions of markers and marked consistent cuts, which we use to ensure that sets of disjoint paths are counted exactly once during our dynamic programming.

▶ **Definition 5.** A triple (V_1, V_2, M) is a **marked consistent cut** of a graph G if (V_1, V_2) is a consistent cut and $M \subseteq E(G[V_1])$. We refer to the edges in M as the **markers**. A marker set is **proper** if it contains at least one edge in each connected component of G which is not an isolate.

Note that if a marked consistent cut contains a proper marker set, all vertices are on the V_1 side of the cut. This is because by the definition of a consistent cut, all isolates are on the V_1 side, and if every connected component contains a marker then all connected components must fall entirely on the V_1 side as well. Therefore for any proper marker set there exists exactly one consistent cut, while all marker sets which are not proper will be paired with an even number of consistent cuts because unmarked components may lie in V_1 *or* V_2. We use proper marker sets to distinguish desired subgraphs by assigning markers in such a way that when we prune the dynamic programming table for solutions (as described later in the section), the only subgraphs we consider which may have a proper marker set are collections of disjoint paths. We know because the marker set is proper that the subgraph has a unique consistent cut, and thus these collections of disjoint paths will only be counted once in some entry of the dynamic programming table, while all other subgraphs will be counted an even number of times. Note that we are not claiming that all collections of disjoint paths will have proper marker sets.

We refer to the complement of a co-path set (the edges in the disjoint paths) as a *cc-solution*, and call it a *marked-cc-solution* when paired with a proper marker set of size exactly equal to its number of non-isolate connected components. While cc-solutions can be viewed as solutions due to their complementary nature, being marked is crucial in our counting algorithm and thus subgraphs which are marked-cc-solutions are what correspond to solutions in the dynamic programming table.

We now describe our use of the Isolation Lemma, which guarantees we are able to use parity to distinguish solutions. Let $f(X)$ denote $\sum_{x \in X} f(x)$.

▶ **Isolation Lemma** ([11]). *Let $\mathcal{F} \subseteq 2^U$ be a non-empty set family over universe U. A function $\omega \colon U \to \mathbb{Z}$ is said to* isolate *$\mathcal{F}$ if there is a unique $S \in \mathcal{F}$ with $\omega(S) = \min_{F \in \mathcal{F}} \omega(F)$. Assign weights $\omega \colon U \to \{1, 2, ..., N\}$ uniformly at random, where the value of N is of the reader's choice. Then the probability that ω isolates $\mathcal{F}$ is at least $1 - |U|/N$.*

Intuitively, if $\mathcal{F}$ is the set of solutions (or complements of solutions) to an instance of Co-Path Set and $|\mathcal{F}|$ is even, then `tw-copath` would return a false negative. This is because while each solution is counted an odd number of times in `tw-copath`, because there are an even number of solutions the total count of solutions is even, making the combined count of solutions and non-solutions even and the algorithm would incorrectly determine a solution does not exist (a false negative). The Isolation Lemma allows us to partition $\mathcal{F}$ based on the weight of each solution (as assigned by ω), and guarantees at least one of the partition's blocks has odd size with constant probability. We let U contain two copies of every edge $e \in E$: one representing e as a marker and one as an edge in the cc-solution. Then 2^U denotes all pairs of edge subsets (potential marked-cc-solutions), and we set $N = 3|U| = 6E$ (selected to achieve success probability in Theorem 1). Each copy of an edge is assigned a weight in $[1, N]$ uniformly at random by ω and the probability of finding an isolating ω is thus $2/3$. We denote the values assigned by ω to the set of marker copies by ω_M, and likewise to the set of edge-in-cc-solution copies by ω_E.

3.2 Counting

A marked-cc-solution C of a graph G corresponds to a co-path set of size k when the number of edges and markers in C match specific values which depend on k and $|E(G)|$. These values are easily deduced because we know the deletion of a co-path set solution of size k will leave $|E(G)| - k$ edges in a cc-solution. Furthermore, because a forest has $n - m$ connected components, the number of markers in C needs to be at most $|V(C)| - |E(G)| + k$. All isolates from a forest can be removed and the resulting graph is still a forest, and thus the actual number of markers necessary in C is $|V(C)| - n_I(C) - |E(G)| + k$.

We now describe a dynamic programming (DP) algorithm over a nice tree decomposition which returns mod 2 the number of appropriately sized marked-cc-solutions in the root's subtree (for a fixed k). Since no-instances have no appropriately sized marked-cc-solutions, and yes-instances have at least one, odd parity for the number of marked-cc-solutions of size corresponding to k implies a solution to the k-Co-Path Set instance must exist.

During the DP algorithm we actually count (for all values (m, e)) the number of *cc-candidates*, which are subgraphs $G' \subseteq G$ with maximum degree 2, exactly e edges, and a marked consistent cut with m markers. The following lemma justifies counting cc-candidates in place of marked-cc-solutions. Note that the weight of a marked-cc-solution or a cc-candidate is equal to the sum of its marker weights and its edge weights.

▶ **Lemma 6.** *The parity of the number of marked-cc-solutions in G with e edges and weight w is the same as the parity of the number of cc-candidates $G' \subseteq G$ with e edges, $|V(G')| - e - n_I(G')$ markers, and weight w.*

Proof. Consider a subgraph $G' \subseteq G$ with maximum degree 2 and e edges. Let M' be a marking of G' such that $\omega_E(E(G')) + \omega_M(M') = w$. Assume first that G' is a collection of paths. We know that G' has $|V(G')| - e - n_I(G')$ non-isolate connected components. If M' is a proper marker set of G', then $|M'| = |V(G')| - e - n_I(G')$ and (G', M') has exactly one consistent cut. Therefore (G', M') contributes one to both the number of marked-cc-solutions and the number of cc-candidates, respectively.

If otherwise M' is not a proper marker set, then (G', M') contains an unmarked connected component and has an even number of consistent cuts, and therefore contributes an even number to the count of cc-candidates and zero to the number of marked-cc-solutions. Finally, if G' contains at least one cycle then $cc(G') > |V(G')| - e - n_I(G')$. Therefore at least one connected component does not contain a marker, and the number of consistent cuts is even, so the contribution to the count of cc-candidates is again even and the contribution

Table 1 Dynamic programming table parameters and upper bounds.

Variable	Parameter	Maximum value
a	# of non-isolated vertices	n
e	# of edges	n^2
m	# of markers	n^2
w	weight of edges and markers	$4n^4$

to the count of marked-cc-solutions is zero. We conclude that the parity of the number of marked-cc-solutions and the parity of the number of cc-candidates is the same. ◀

Our dynamic programming algorithm is a bottom-up approach over a nice tree decomposition. We build cc-candidates for all values of m and e (encoding the option to add/not add edges and select/not select edges as markers), and keep track of various parameters ensuring that when pruning the DP table we only consider cc-candidates which could be valid solutions to the k-Co-Path Set instance. We use the number of edges to ensure our solution is of the correct size, and the number of markers and non-isolate vertices to determine when a subgraph is acyclic. The weight parameter allows us to distinguish between solutions and decreases the likelihood of a false negative occurring via the Isolation Lemma.

Finally, we need a parameter that encodes the degree information required to properly combine cc-candidates as we iterate up the tree. We call this parameter a *degree-function* and define it on the vertices V of a bag as $f : V \to \Sigma = \{0, 1_1, 1_2, 2\}$, where $f(v)$ corresponds to v's degree in the associated cc-candidates of the table entry — for vertices of degree 1, their value 1_j denotes which side of the partition (V_1, V_2) they are on. Vertices with degree 0 are on the V_1 side of the cut by definition and degree 2 vertices cannot gain additional incident edges, so we need not keep track of their side of the cut. In summary, we have table entries $A_x(a, e, m, w, s)$ counting the number of cc-candidates with a non-isolated vertices, e edges, m markers, weight w, and degree-function s, where all vertices which have been introduced in the subtree rooted at x are present and only edges which have been introduced in this subtree may be present.

In the following description of the dynamic programming algorithm over a nice tree decomposition T, we let z_1, z_2 denote the children of a join node; otherwise, the unique child is denoted y.

Leaf:

$A_x(0, 0, 0, 0, \emptyset) = 1$; $A_x(a, e, m, w, s) = 0$ for all other inputs.

Introduce vertex v:

$$A_x(a, e, m, w, s[v \to 0]) = A_y(a, e, m, w, s);\ A_x(a, e, m, w, s[v \to i]) = 0,\ \forall i \neq 0.$$

Introduce edge uv:

$$A_x(a, e, m, w, s) = A_y(a, e, m, w, s) + \sum_{\substack{\alpha_t \in subs(s(t)) \\ t \in \{u,v\}}} [\![\phi_2(\alpha_u, \alpha_v)]\!] A_y(a', e-1, m, w', s')$$

$$+ \sum_{\substack{\alpha_t \in subs(s(t)) \\ t \in \{u,v\}}} [\![\phi_1(\alpha_u, \alpha_v)]\!] \Big(A_y(a', e-1, m, w', s') + A_y(a', e-1, m-1, w'', s') \Big),$$

where $\phi_j(\alpha_u, \alpha_v) = (\alpha_u = 1_j \vee s(u) = 1_j) \wedge (\alpha_v = 1_j \vee s(v) = 1_j)$, $a' = a - (|\{1_1, 1_2\} \cap \{s(u), s(v)\}|)$, $w' = w - \omega_E(uv)$, $w'' = w - \omega_E(uv) - \omega_M(uv)$, $s' = s[u \to \alpha_u, v \to \alpha_v]$, and the *subs* function returns all the values the degree-function in child node y could have assigned to vertices u and v based on current degree-function s (summarized below).

$s(v)$	0	1_1	1_2	2
$subs(s(v))$	$\emptyset$	0	0	$\{1_1, 1_2\}$

We now argue this formula's correctness. The term $A_y(a, e, m, w, s)$ handles the case when uv is excluded from the cc-solution. We handle the case when uv is added to the cc-solution by iterating over all possible *subs* values for each endpoint, only considering counts in child y's entries where u and v have the appropriate *subs* values (preventing us from ever having a vertex with degree greater than 2). Note that we use the ϕ_j function to guarantee that if s labels u or v as an isolate, we do not use the introduced edge. We have a summation for both possible j values in order to consider uv falling on either side of the cut. The formulation of a' assures that each endpoint of degree 1 is now included in the count of non-isolates (i.e. when u and/or v had degree 0 in y). We utilize the marker weight of uv to distinguish when we choose it as a marker (only if on V_1 side of cut), and increment m accordingly. In either case, we update w appropriately (with w' if no marker, w'' if marker introduced).

Forget vertex h:

$$A_x(a, e, m, w, s) = \sum_{\alpha \in \{0, 1_1, 1_2, 2\}} A_y(a, e, m, w, s[h \to \alpha]).$$

As a forgotten vertex can have degree 0, 1 or 2 in a cc-candidate, we must consider all possible values that s assigns to h in child bag y. Note that cc-candidates in which h is both not an isolate and not a member of a connected component that contains a marker will cancel mod 2, as h can be on either side of the cut and all parameters will be identical.

Join: We compute A_x from A_{z_1} and A_{z_2} via fast subset convolution [1] taking care to only combine table entries whose degree-functions are *compatible*, ensuring that only joins which preserve the constraints of the degree-functions of the children nodes occur.

▶ **Definition 7.** At a join node x with children z_1 and z_2, the degree-functions s_1 from A_{z_1}, s_2 from A_{z_2}, and s from A_x are **compatible** if one of the following holds for every vertex v in x: (i) $s_i(v) = 0$ and $s_l(v) = s(v), i \neq l$ or (ii) $s_1(v) = s_2(v) = 1_j$ and $s(v) = 2$ for $i, j, l \in [1, 2]$.

In order to apply Lemma 3, we let B be the bag at x, and transform the values assigned by the degree function s to values in $\mathbb{Z}_4$. Let $\phi\colon \{0, 1_1, 1_2, 2\} \to \mathbb{Z}_4$ and $\rho\colon \{0, 1_1, 1_2, 2\} \to \mathbb{Z}$ be defined as in the table below, extending to vectors by component-wise application.

	0	1_1	1_2	2
ϕ	0	1	3	2
ρ	0	1	1	2

We use ϕ to apply Lemma 3, while the function ρ (which corresponds to a vertex's degree) is used in tandem to ensure the compatibility requirements are met: if $\phi(s_1) + \phi(s_2) = \phi(s)$,

then necessarily $\rho(s_1) + \rho(s_2) \geq \rho(s)$. From the above table it is easy to verify that $\phi(s_1)+\phi(s_2) = \phi(s)$ and $\rho(s_1)+\rho(s_2) = \rho(s)$ together imply that s_1, s_2 and s are compatible. We sum over both functions when computing values for join nodes, to make sure that solutions from the children are combined only when there is compatibility.

Assign $t_1 = \phi(s_1)$, $t_2 = \phi(s_2)$, and $t = \phi(s)$ in accordance with Lemma 3. Let $\rho(s) = \sum_{v \in B} \rho(s(v))$; that is $\rho(s)$ is the sum of the degrees of all the vertices in the join node, as assigned by degree-function s. By defining functions f and g as follows:

$$f^{\langle d,a,e,m,w\rangle}(\phi(s)) = [\![\rho(s) = d]\!] A_{z_1}(a, e, m, w, s),$$
$$g^{\langle d,a,e,m,w\rangle}(\phi(s)) = [\![\rho(s) = d]\!] A_{z_2}(a, e, m, w, s),$$

and writing $\vec{r_i}$ for the vector $\langle d_i, a_i, e_i, m_i, w_i\rangle$ in order to consider all ways to split the parameter values of x between the two children nodes, we can now compute

$$A_x(a, e, m, w, s) = \sum_{\vec{r_1}+\vec{r_2}=\langle\rho(s),a',e,m,w\rangle} (f^{\vec{r_1}} *_x^4 g^{\vec{r_2}})(\phi(s))$$

where $a' = a + |s_1^{-1}\{1_1, 1_2\} \cap s_2^{-1}\{1_1, 1_2\}|$. We point out that

$$\sum_{\vec{r_1}+\vec{r_2}=\langle\rho(s),a',e,m,w\rangle} (f^{\vec{r_1}} *_x^4 g^{\vec{r_2}})(\phi(s)) = 1$$

only if both $\phi(s_1) + \phi(s_2) = \phi(s)$ and $\rho(s_1) + \rho(s_2) = \rho(s)$; that is, exactly when s_1, s_2 and s are compatible.

We conclude this section by describing how we search the DP table for marked-cc-solutions at the root node r. By Lemma 6, the parity of the number of marked-cc-solutions with $|E|-k$ edges and weight w is the same as the parity of the number of cc-candidates G' with $|E|-k$ edges, $|V(G')| - (|E|-k) - n_I(G')$ markers and weight w. These candidates are recorded in the table entries $A_r(a, |E|-k, a-|E|+k, w, \emptyset)$, where a is the number of non-isolates. Therefore, if there exists some a and w so that $A_r(a, |E|-k, a-|E|+k, w, \emptyset) = 1$, then we have a yes-instance of k-Co-Path Set. Note that the degree-function is $\emptyset$ in this entry because there are no vertices contained in the root node by definition.

By Lemma 3, the time complexity of `tw-copath` for a join node B is $O^*(4^{|B|})$, which is $O^*(4^{tw})$. Note that for the other four types of bags, as we only consider one instance of s per table entry, the complexity for each is $O^*(4^{tw})$. We point out that the size of the table is polynomial in n because there are a linear number of bags and a polynomial number of entries (combinations of parameters) for each bag. Since the nice tree decomposition has size linear in n, the bottom-up dynamic programming runs in total time $O^*(4^{tw})$. This complexity bound combined with the correctness of `tw-copath` discussed above proves Theorem 4.

4 Achieving $O^*(1.588^k)$ in General Graphs

In order to use `tw-copath` to solve k-Co-Path Set in graphs with unbounded treewidth, we combine kernelization and a branching procedure to generate a set of *reduced instances* – bounded treewidth subgraphs of the input graph G. Specifically, we begin by constructing a kernel of size at most $6k$ as described in [7]. Our reduced instances are bounded degree subgraphs of the kernel given by a branching technique. We prove that (1) at least one reduced instance is an equivalent instance; (2) we can bound the number of reduced instances; and (3) each reduced instance has bounded treewidth. Finally, we analyze the overall computational complexity of this process.

Algorithm 1: Generating reduced instances

```
1  Algorithm deg-branch(G, k, ℓ, D, b)
2      Let v be a vertex of maximum degree in G
3      if deg(v) ≥ D + 1 and b ≥ D − 1 then
4          Arbitrarily select vertices u_1, ... u_{D+1} from N(v)
5          R = ∅, E_v = {{v, u_i} | i ∈ [1, D + 1]}
6          for e_1, e_2 ∈ E_v, e_1 ≠ e_2 do
7              E'_v = E_v \ {e_1, e_2}
8              R = R ∪ deg-branch(G \ E'_v, k, ℓ, D, b − (D − 1))
9          return R
10     else if b = 0 and deg(v) ≤ D then return {(G, k − ℓ)}
11     else return ∅                                   // Discard G
```

4.1 Kernelization and Branching

We start by describing our branching procedure `deg-branch` (Algorithm 1), which uses a degree-bounding technique similar to that of Zhang et al. [16]. Our implementation takes an instance (G, k) of Co-Path Set and two non-negative integers ℓ and D, and returns a set of reduced instances $\{(G_i, k - \ell)\}$ so that (1) each G_i is a subgraph of G with exactly $|E| - \ell$ edges and maximum degree at most D; and (2) at least one $(G_i, k-\ell)$ is an equivalent instance to (G, k). The size of the output (and hence the running time) of `deg-branch` depends on both input parameters ℓ and D. We will select D to achieve the desired complexity in `copath` in Section 4.3. We also make use of a budget parameter b, which keeps track of how many more edges can be removed per the constraints of ℓ (b is initially set to ℓ).

Our branching procedure leverages the observation that if a co-path set S exists, then every vertex has at most two incident edges not in S. Specifically, for every vertex of degree greater than D, we branch on pairs of incident edges which could remain after removing a valid co-path set (calling each pair a *candidate*), creating a search tree of subgraphs.

Algorithm 1 returns a set of reduced instances which have had exactly ℓ edges removed. The size of the set is at most the number of leaves in the search tree of the branching process (inequality can result from the algorithm discarding branches in which the number of edits necessary to branch on a vertex exceeds the number of allowed deletions remaining). We now give an upper bound on the size of this set.

▶ **Lemma 8.** *Let T be a search tree formed by* `deg-branch`(G, ℓ, D, k, b). *The number of leaves of T is at most $\binom{D+1}{2}^{\ell/(D-1)}$.*

Proof of Lemma 8. The number of children of each interior node of T is $\binom{D+1}{2}$, resulting in at most $\binom{D+1}{2}^{depth(T)}$ leaves. The depth of T is limited by the second condition of the `if` on line 3 of Algorithm 1. For each recursive call, b is decremented by $(D - 1)$, until $b \leq D - 1$. As b is initially set to ℓ, this implies $depth(T) \leq \ell/(D - 1)$, proving the claim. ◀

Finally, we argue that at least one member of the set of reduced instances returned by `deg-branch` is equivalent to the original. Consider a solution F to k-Co-Path Set in the original instance (G, k). Every vertex has at most two incident edges in $G[E\backslash F]$, and since all candidates are considered at every high-degree vertex, at least one branch correctly keeps all of these edges.

Table 2 Numerically obtained constants c_d, $3 \leq d \leq 17$, used in Lemma 9; originally given in Table 6.1 of [9].

d	3	4	5	6	7	8	9	10
c_d	0.1667	0.3334	0.4334	0.5112	0.5699	0.6163	0.6538	0.6847
d	11	12	13	14	15	16	17	
c_d	0.7105	0.7325	0.7514	0.7678	0.7822	0.7949	0.8062	

4.2 Treewidth of Reduced Instances

Our algorithm `deg-branch` produces reduced instances with bounded degree; in order to bound their treewidth, we make use of the following result, which originated from Lemma 1 in [8] and was extended in [9].

▶ **Lemma 9.** *For $\epsilon > 0$, there exists $n_\epsilon \in \mathbb{Z}^+$ s.t. for every graph G with $n > n_\epsilon$ vertices,*

$$tw(G) \leq \left(\sum_{i=3}^{17} c_i n_i \right) + n_{\geq 18} + \epsilon n,$$

where n_i is the number of vertices of degree i in G for $i \in \{3, \ldots, 17\}$, $n_{\geq 18}$ is the number of vertices of degree at least 18, and c_i is given in Table 2. Moreover, a tree decomposition of the corresponding width can be constructed in polynomial time in n.

Since the structure of k-Co-Path Set naturally provides some constraints on the degree sequence of yes-instances, we are able to apply Lemma 9 to our reduced instances to effectively bound treewidth. We first find an upper bound on the number of degree-3 vertices in any yes-instance of k-Co-Path Set.

▶ **Lemma 10.** *Let n_i be the number of vertices of degree i in a graph G for any $i \in \mathbb{Z}^+$, and Δ be the maximum degree of G. If (G, k) is a yes-instance of k-Co-Path Set, then $n_3 \leq 2k - (\sum_{i=4}^{\Delta} (i-2)n_i)$.*

Proof. Since (G, k) is a yes-instance, removing some set of at most k edges results in a graph of maximum degree 2. For a vertex of degree $j \geq 3$, at least $j - 2$ incident edges must be removed. Thus, $n_3 + 2n_4 + 3n_5 + \ldots + (\Delta - 2)n_\Delta \leq 2k$ (each removed edge counts twice – once for each endpoint). ◀

▶ **Lemma 11.** *Let (G, k) be an instance of k-Co-Path Set such that G has n vertices and max degree at most $\Delta \in \{3, \ldots, 17\}$. If (G, k) is a yes-instance, then the treewidth of G is upper bounded by $k/3 + \epsilon n + c$, for any $\epsilon > 0$ and constant $c = n_\epsilon$ as defined in Lemma 9. A tree decomposition of the corresponding width can be constructed in polynomial time in n.*

Proof. Let n_ϵ be defined as in Lemma 9. Let G' be the graph formed by adding $N = n_\epsilon$ isolates to G. By Lemma 9, because G' has maximum degree at most Δ, $tw(G') \leq (1/6)n_3 + (1/3)n_4 + \ldots + c_\Delta n_\Delta + \epsilon(N + n)$. We can substitute the bound for n_3 from Lemma 10, which yields:

$$\begin{aligned} tw(G') &\leq \frac{2k - (\sum_{i=4}^{\Delta} (i-2)n_i)}{6} + \frac{n_4}{3} + \ldots + c_\Delta n_\Delta + \epsilon(N + n) \\ &\leq \frac{k}{3} + \epsilon(n + N). \end{aligned}$$

Algorithm 2: Deciding k-Co-Path Set

```
1 Algorithm copath (G,k)
2     (G', k') = 6k-kernel(G, k)
3     for k_1 ← 0 to k' do
4         Q_{k_1} = deg-branch(G', k', k_1, 10, k_1)
5         foreach (G_i, k_2) ∈ Q_{k_1} do
6             if tw-copath(G_i, k_2) then return true
7     return false
```

Note that the inequality holds because we can pair the negative terms of $(\sum_{i=4}^{\Delta}(i-2)n_i)/6$ with the corresponding terms of $n_4/3 + \ldots + c_\Delta n_\Delta$ and the value of $c_j n_j - (j-2)(n_j)/6$ is non-positive for all $j \in [4, 17]$. Since $N = n_\epsilon$ is a constant, we have $tw(G') \leq k/3 + \epsilon n + c$. Since $G \subseteq G'$ and treewidth is monotone under subgraph inclusion, this proves the claim. ◀

We point out that when applying Lemma 11 to reduced instances, computing the desired tree decomposition is polynomial in k (since they are subgraphs of a $6k$-kernel).

4.3 The Algorithm `copath`

This section describes how we combine the above techniques to prove Theorem 1. As shown in Algorithm 2, we start by applying `6k-kernel` [7] to find G', a kernel of size at most $6k$; this process deletes $k - k'$ edges. We then guess the number of edges $k_1 \in [0, k']$ to remove during branching, and use `deg-branch` to create a set of reduced instances Q_{k_1}, each of which have $k' - k_1$ edges. Note that `deg-branch` considers *all* possible reduced instances, and thus if a (cc-)solution exists, it is contained in at least one reduced instance. To ensure the complexity of finding the reduced instances does not dominate the running time, we set the degree bound D of the reduced instances to be 10 (any choice of $10 \leq D \leq 17$ is valid). By considering all possible values of k_1, we are assured that if (G, k) is a yes-instance, some Q_{k_1} contains a yes-instance. Each reduced instance is then passed to `tw-copath`, which correctly decides the problem with probability 2/3.

Proof of Theorem 1. We now analyze the running time of `copath`, as given in Algorithm 2. By Lemma 8, the size of each Q_{k_1} is $O(1.561^{k_1})$. For each reduced instance (G_i, k_2) in Q_{k_1}, we have $tw(G_i) \leq k_2/3 + \epsilon(6k) + c$ by Lemma 11.

Applying Theorem 4, `tw-copath` runs in time $O^*(4^{k_2/3+\epsilon 6k})$ for each reduced instance (G_i, k_2) in Q_{k_1} (with success probability at least 2/3). Each iteration of the outer `for` loop can then be completed in time

$$O^*(1.561^{k_1} 4^{k_2/3+\epsilon 6k}) = O^*(4^{k/3+\epsilon 6k}) = O^*(1.588^k),$$

where we use that $k_1 + k_2 = k' \leq k$, and choose $\epsilon < 10^{-5}$. Since this loop runs at most $k+1$ times, this is also a bound on the overall computational complexity of `copath`. Additionally `copath` is linear-fpt, as the kernelization of [7] is $O(n)$, and the kernel has size $O(k)$, avoiding any additional poly(n) complexity from the `tw-copath` subroutine. Note that by Lemma 9 the tree decomposition can be found in polynomial time in the size of the reduced instance. Since reduced instances are subsets of $6k$-kernels, the linearity is unaffected because the graph has size polynomial in k. ◀

5 Conclusion

This paper gives an $O^*(4^{tw})$ fpt algorithm for Co-Path Set. By coupling this with kernelization and branching, we derive an $O^*(1.588^k)$ linear-fpt algorithm for deciding k-Co-Path, significantly improving the previous best-known result of $O^*(2.17^k)$. We believe that the idea of combining a branching algorithm which guarantees equivalent instances with bounds on the degree sequence from the problem's constraints can be applied to other problems in order to obtain a bound on the treewidth (allowing treewidth-parameterized approaches to be extended to general graphs).

One natural question is whether similar techniques extend to the generalization of Co-Path Set to k-uniform hypergraphs (as treated in Zhang et al. [16]). It is also open whether the combined parameterization asking for a co-path set of size k resulting in ℓ disjoint paths is solvable in sub-exponential fpt time.

Acknowledgements. We thank two anonymous reviewers for providing a simplification of our previous branching algorithm and pointing out the result from [9] enabling us to branch on vertices with degree greater than 7. We also thank Felix Reidl for helpful suggestions on an earlier draft that significantly improved the presentation of the results.

References

1 A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of STOC*, pages 67–74, 2007. doi:10.1145/1250790.1250801.

2 Z. Chen, G. Lin, and L. Wang. An approximation algorithm for the minimum co-path set problem. *Algorithmica*, 60(4):969–986, 2011.

3 Y. Cheng, Z. Cai, R. Goebel, G. Lin, and B. Zhu. The radiation hybrid map construction problem: recognition, hardness, and approximation algorithms. Unpublished Manuscript, 2008.

4 D. Cox, M. Burmeister, E. Price, S. Kim, and R. Myers. Radiation hybrid mapping: a somatic cell genetic method for constructing high-resolution maps of mammalian chromosomes. *Science*, 250:245–50, 1990.

5 M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. van Rooij, and J. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS*, pages 150–159. IEEE, 2011.

6 Q. Feng, Q. Zhou, and S. Li. Randomized parameterized algorithms for co-path set problem. In *FAW*, pages 82–93. Springer, 2014.

7 Q. Feng, Q. Zhou, and J. Wang. Kernelization and randomized parameterized algorithms for co-path set problem. *J. Comb. Optim.*, 2015. in press, DOI 10.1007/s10878-015-9901-y. URL: http://dx.doi.org/10.1007/s10878-015-9901-y.

8 F. Fomin, S. Gaspers, S. Saurabh, and A. Stepanov. On two techniques of combining branching and treewidth. *Algorithmica*, 54(2):181–207, April 2009. doi:10.1007/s00453-007-9133-3.

9 S. Gaspers. *Exponential Time Algorithms – Structures, Measures, and Bounds*. VDM, 2010. URL: http://bora.uib.no/bitstream/handle/1956/3436/Dr.thesis_Serge_Gaspers.pdf.

10 T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994. doi:10.1007/BFb0045375.

11 K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of STOC*, pages 345–354. ACM, 1987. doi:10.1145/28395.383347.

12 M. Pilipczuk. Solving connectivity problems parameterized by treewidth in single exponential time. In *MFCS*, pages 520–531, 2011.

13 C. Richard III, D. Withers, T. Meeker, S. Maurer, G. Evans, R. Myers, and D. Cox. A radiation hybrid map of the proximal long arm of human chromosome 11 containing the multiple endocrine neoplasia type 1 (men-1) and bcl-1 disease loci. *Am. J. Hum. Genet.*, 49(6):1189–1196, 1991.

14 N. Robertson and P. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

15 D. Slonim, L. Kruglyak, L. Stein, and E. Lander. Building human genome maps with radiation hybrids. *J. Comp. Biol.*, 4(4):487–504, 1997.

16 C. Zhang, H. Jiang, and B. Zhu. Radiation hybrid map construction problem parameterized. *Journal of Combinatorial Optimization*, 27(1):3–13, 2014.

Clifford Algebras Meet Tree Decompositions*

Michał Włodarczyk

University of Warsaw, Faculty of Mathematics, Informatics, and Mechanics, Warsaw, Poland
m.wlodarczyk@mimuw.edu.pl

Abstract

We introduce the Non-commutative Subset Convolution – a convolution of functions useful when working with determinant-based algorithms. In order to compute it efficiently, we take advantage of Clifford algebras, a generalization of quaternions used mainly in the quantum field theory.

We apply this tool to speed up algorithms counting subgraphs parameterized by the treewidth of a graph. We present an $O^*((2^\omega + 1)^{tw})$-time algorithm for counting Steiner trees and an $O^*((2^\omega + 2)^{tw})$-time algorithm for counting Hamiltonian cycles, both of which improve the previously known upper bounds. The result for Steiner Tree also translates into a deterministic algorithm for Feedback Vertex Set. All of these constitute the best known running times of deterministic algorithms for decision versions of these problems and they match the best obtained running times for pathwidth parameterization under assumption $\omega = 2$.

1998 ACM Subject Classification F.2.2 [Nonnumerical Algorithms and Problems] Computations on Discrete Structures, G.2.2 [Graph Theory] Graph Algorithms

Keywords and phrases fixed-parameter tractability, treewidth, Clifford algebra, algebra isomorphism

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.29

1 Introduction

The concept of *treewidth* has been introduced by Robertson and Seymour in their work on graph minors [13]. The treewidth of a graph is the smallest possible width of its *tree decomposition*, i.e. a tree-like representation of the graph. Its importance follows from the fact that many NP-hard graph problems become solvable on trees with a simple dynamical programming. A similar idea of *pathwidth* captures the width of a graph in case we would like to have a *path decomposition*. Formal definitions can be found in Section 2.2.

A bound on the graph's treewidth allows to design efficient algorithms using *fixed-parameter tractability*. An algorithm is called fixed-parameter tractable (FPT) if it works in time complexity $f(k)n^{O(1)}$ where k is a parameter describing hardness of the instance and f is a computable function. We also use notation $O^*(f(k))$ that suppresses polynomial factors with respect to the input size. Problems studied in this work are parameterized by the graph's pathwidth or treewidth. To distinguish these cases we denote the parameter respectively pw or tw.

It is natural to look for a function f that is growing relatively slow. For problems with a local structure, like Vertex Cover or Dominating Set, there are simple FPT algorithms with single exponential running time. They usually store c^{tw} states for each node of the

* This work is partially supported by Foundation for Polish Science grant HOMING PLUS/2012-6/2 and a 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).

decomposition and take advantage of the Fast Subset Convolution [2] to perform the *join* operation in time $O^*(c^{tw})$. As a result, time complexities for pathwidth and treewidth parameterizations remain the same. The Fast Subset Convolution turned out helpful in many other problems, e.g. Chromatic Number, and enriched the basic toolbox used for exponential and parameterized algorithms.

Problems with connectivity conditions, like Steiner Tree or Hamiltonian Cycle, were conjectured to require time $2^{\Omega(tw \log tw)}$ until the breakthrough work of Cygan et al. [8]. They introduced the randomized technique Cut & Count working in single exponential time. The obtained running times were respectively $O^*(3^{tw})$ and $O^*(4^{tw})$. Afterwards, a faster randomized algorithm for Hamiltonian Cycle parameterized by the pathwidth was presented with running time $O^*((2+\sqrt{2})^{pw})$ [7]. This upper bound as well as $O^*(3^{pw})$ for Steiner Tree are tight modulo subexponential factors under the assumption of *Strong Exponential Time Hypothesis* [7, 8].

The question about the existence of single exponential deterministic methods was answered positively by Bodlaender et al. [4]. What is more, presented algorithms count the number of Steiner trees or Hamiltonian cycles in a graph. However, in contrast to the Cut & Count technique, a large gap emerged between the running times for pathwidth and treewidth parameterizations – the running times were respectively $O^*(5^{pw})$, $O^*(10^{tw})$ for Steiner Tree and $O^*(6^{pw})$, $O^*(15^{tw})$ for Hamiltonian Cycle. This could be explained by a lack of efficient algorithms to perform the *join* operation, necessary only for tree decompositions. Some efforts have been made to reduce this gap and the deterministic running time for Steiner Tree has been improved to $O^*((2^{\omega-1}\cdot 3+1)^{tw})$ [9].

1.1 Our contribution

The main contribution of this work is creating a link between Clifford algebras, objects not being used in algorithmics to the best of our knowledge, and fixed-parameter tractability. As the natural dynamic programming approach on tree decompositions uses the Fast Subset Convolution (FSC) to perform efficiently the *join* operation, there was no such a tool for algorithms based on the determinant approach.

Our first observation is that the FSC technique can be regarded as an isomorphism theorem for some associative algebras. To put it briefly, a Fourier-like transform is being performed in the FSC to bring computations to a simpler algebra. Interestingly, this kind of transform is just a special case of the Artin-Wedderburn theorem [1], which seemingly is not widely reported in computer science articles. The theorem provides a classification of a large class of associative algebras, not necessarily commutative (more in Appendix A). We use this theory to introduce the Non-commutative Subset Convolution (NSC) and speed up multiplication operations in an algebra induced by the *join* operation in determinant-based dynamic programming on tree decomposition. An important building block is a fast Fourier-like transform for a closely related algebra [11]. We hope our work will encourage researchers to investigate further algorithmic applications of the Artin-Wedderburn theorem.

1.2 Our results

We apply our algebraic technique to determinant approach introduced by Bodlaender et al. [4]. For path decomposition, they gave an $O^*(5^{pw})$-time algorithm for counting Steiner trees and an $O^*(6^{pw})$-time algorithm for counting Hamiltonian cycles. The running times for tree decomposition were respectively $O^*(10^{tw})$ and $O^*(15^{tw})$. These gaps can be explained by

the appearance of the *join* operation in tree decompositions which could not be handled efficiently so far.

By performing NSC in time complexity $O^*(2^{\frac{\omega n}{2}})$ we partially solve an open problem about the *different convolution* from [6]. Our further results may be considered similar to those closing the gap between time complexities for pathwidth and treewidth parameterizations for Dominating Set by switching between representations of states in dynamic programming [14]. We improve the running times to $O^*((2^\omega+1)^{tw})$ for counting Steiner trees and $O^*((2^\omega+2)^{tw})$ for counting Hamiltonian cycles, where ω denotes the matrix multiplication exponent (currently it is established that $\omega < 2.373$ [15]). These are not only the fastest known algorithms for counting these objects, but also the fastest known deterministic algorithms for the decision versions of these problems. The deterministic algorithm for Steiner Tree can be translated into a deterministic algorithm for Feedback Vertex Set [4] so our technique provides an improvement also in this case.

Observe that the running times for pathwidth and treewidth parameterizations match under the assumption $\omega = 2$. Though we do not hope for settling the actual value of ω, this indicates there is no further space for significant improvement unless pure combinatorial algorithms (i.e. not based on matrix multiplication) are invented or the running time for pathwidth parameterization is improved.

1.3 Organization of the paper

Section 3 provides a brief introduction to Clifford algebras. The bigger picture of the employed algebraic theory can be found in Appendix A. In Section 4 we define the NSC and design efficient algorithms for variants of the NSC employing the algebraic tools. Sections 5 and 6 present how to apply the NSC in counting algorithms for Steiner Tree and Hamiltonian Cycle. They contain main ideas improving the running times, however in order to understand the algorithms completely one should start from Section 4 (*Determinant approach*) in [4]. The algorithm for Hamiltonian Cycle is definitely more complicated and its details, formulated as two isomorphism theorems, are placed in Appendix C.

2 Preliminaries

We will start with notation conventions.

1. $A \uplus B = C$ stands for $(A \cup B = C) \wedge (A \cap B = \emptyset)$.
2. $A \triangle B = (A \setminus B) \cup (B \setminus A)$.
3. $[\alpha]$ equals 1 if condition α holds and 0 otherwise.
4. For permutation f of a linearly ordered set U
$$\operatorname{sgn}(f) = (-1)^{|\{(a,b) \in U \times U \wedge a<b \wedge f(a)>f(b)\}|}.$$
5. For A, B being subsets of a linearly ordered set
$$I_{A,B} = (-1)^{|\{(a,b) \in A \times B \wedge a>b\}|}. \tag{1}$$

Let us note two simple properties of I.

▶ **Claim 1.** *For disjoint A, B*
$$I_{A,B} I_{B,A} = (-1)^{|A||B|}.$$

▶ **Claim 2.** *For $A \cap B = \emptyset$ and $C \cap D = \emptyset$*
$$I_{A \cup B, C \cup D} = I_{A,C} I_{A,D} I_{B,C} I_{B,D}.$$

2.1 Fast Subset Convolution

Let us consider a universe U of size n and functions $f, g : 2^U \longrightarrow \mathbb{Z}$.

▶ **Definition 3.** The Möbius transform of f is function $\hat{f}$ defined as

$$\hat{f}(X) = \sum_{A \subseteq X} f(A).$$

▶ **Definition 4.** Let $f * g$ denote a *subset convolution* of f, g defined as

$$(f * g)(X) = \sum_{A \uplus B = X} f(A)g(B).$$

▶ **Theorem 5** (Björklund et al. [2])**.** *The Möbius transform, its inverse, and the subset convolution can be computed in time* $O^*(2^n)$.

2.2 Pathwidth and treewidth

▶ **Definition 6.** A *tree (path) decomposition* of a graph G is a tree $\mathbb{T}$ (path $\mathbb{P}$) in which each node x is assigned a bag $B_x \subseteq V(G)$ such that

1. for every edge $uv \in E(G)$ there is a bag B_x containing u and v,
2. for every vertex v the set $\{x \mid v \in B_x\}$ forms a non-empty subtree (subpath) in the decomposition.

The width of the decomposition is defined as $\max_x |B_x| - 1$ and the treewidth (pathwidth) of G is a minimum width over all possible tree (path) decompositions.

If a graph admits a tree decomposition of width t then it can be found in time $n \cdot 2^{O(t^3)}$ [3] and a decomposition of width at most $4t + 1$ can be constructed in time $poly(n) \cdot 2^{O(t)}$ [10]. We will assume that a decomposition of the appropriate type and width is given as a part of the input.

▶ **Definition 7.** A *nice tree (path) decomposition* is a decomposition with one special node r called the root and in which each bag is one of the following types:

1. *Leaf bag:* a leaf x with $B_x = \emptyset$,
2. *Introduce vertex v bag:* a node x having one child y for which $B_x = B_y \uplus \{v\}$,
3. *Forget vertex v bag:* a node x having one child y for which $B_y = B_x \uplus \{v\}$,
4. *Introduce edge uv bag:* a node x having one child y for which $u, v \in B_x = B_y$,
5. *Join bag:* **(only in tree decomposition)** a node x having two children y, z with condition $B_x = B_y = B_z$.

We require that every edge from $E(G)$ is introduced exactly once and B_r is an empty bag. For each x we define V_x and E_x to be sets of respectively vertices and edges introduced in the subtree of the decomposition rooted at x.

Given a tree (path) decomposition we can find a nice decomposition in time $n \cdot tw^{O(1)}$ [8, 10] and we will work only on these. When analyzing running time of algorithms working on tree decompositions we will estimate the bag sizes from the above assuming $|B_x| = tw$.

2.3 Problems definitions

Steiner Tree
Input: graph G, set of terminals $K \subseteq V(G)$, integer k
Decide: whether there is a subtree of G with at most k edges connecting all vertices from K

Hamiltonian Cycle
Input: graph G
Decide: whether there is a cycle going through every vertex of G exactly once

Feedback Vertex Set
Input: graph G, integer k
Decide: whether there is a set $Y \subseteq V$ of size at most k such that every cycle in G contains a vertex from Y

In the counting variants of problems we ask for a number of structures satisfying the given conditions. This setting is at least as hard as the decision variant.

3 Clifford algebras

Some terms used in this section originate from advanced algebra. For better understanding we suggest reading Appendix A.

▶ **Definition 8.** The Clifford algebra $Cl_{p,q}(R)$ is a 2^{p+q}-dimensional associative algebra over a ring R. It is generated by $x_1, x_2 \ldots, x_{p+q}$.

These are rules of multiplication of generators:

1. e is a neutral element of multiplication,
2. $x_i^2 = e$ for $i = 1, 2, \ldots, p$,
3. $x_i^2 = -e$ for $i = p+1, p+2, \ldots, p+q$,
4. $x_i x_j = -x_j x_i$ if $i \neq j$.

All 2^{p+q} products of ordered sets of generators form a basis of $Cl_{p,q}(R)$ (e is treated as a product of an empty set). We provide a standard addition and we extend multiplication for all elements in an associative way.

We will be mainly interested only in $Cl_{n,0}(\mathbb{Z})$ [1] and its natural embedding into $Cl_{n,0}(\mathbb{R})$. As $q = 0$, we can neglect condition 3 when analyzing these algebras.

For $A = \{a_1, a_2, \ldots, a_k\} \subseteq [1 \ldots n]$ where $a_1 < a_2 < \cdots < a_k$ let $x_A = x_{a_1} x_{a_2} \cdots x_{a_k}$. Each element of $Cl_{n,0}(\mathbb{R})$ can be represented as $\sum_{A \subseteq [1 \ldots n]} a_A x_A$, where a_A are real coefficients. Using condition 4 we can deduce a general formula for multiplication in $Cl_{n,0}(\mathbb{R})$:

$$\left(\sum_{A \subseteq [1 \ldots n]} a_A x_A \right) \left(\sum_{B \subseteq [1 \ldots n]} b_B x_B \right) = \sum_{C \subseteq [1 \ldots n]} \left(\sum_{A \triangle B = C} a_A b_B I_{A,B} \right) x_C \tag{2}$$

where the meaning of $I_{A,B}$ is explained in (1).

As a Clifford algebra over $\mathbb{R}$ is semisimple, it is isomorphic to a product of matrix algebras by the Artin-Wedderburn theorem (see Theorem 31). However, it is more convenient to first embed $Cl_{n,0}(\mathbb{R})$ in a different Clifford algebra that is isomorphic to a single matrix algebra. As a result, we obtain a monomorphism $\phi : Cl_{n,0}(\mathbb{R}) \longrightarrow \mathbb{M}_{2^m}(\mathbb{R})$ (see Definition 28) where $m = \frac{n}{2} + O(1)$ and the following diagram commutes ($*$ stands for multiplication).

$$\begin{array}{ccc} Cl_{n,0}(\mathbb{R}) & \xrightarrow{\phi} & \mathbb{M}_{2^m}(\mathbb{R}) \\ \downarrow * & & \downarrow * \\ Cl_{n,0}(\mathbb{R}) & \xrightarrow{\phi} & \mathbb{M}_{2^m}(\mathbb{R}) \end{array} \tag{3}$$

[1] Clifford algebras with $q = 0$ appear also in geometric literature as *exterior algebras*.

Thus, we can perform multiplication in the structure that is more convenient for us. For $a, b \in Cl_{n,0}(\mathbb{Z})$ we can treat them as elements of $Cl_{n,0}(\mathbb{R})$, find matrices $\phi(a)$ and $\phi(b)$, multiply them efficiently, and then revert the ϕ transform. The result always exists and belongs to $Cl_{n,0}(\mathbb{Z})$ because $Cl_{n,0}(\mathbb{Z})$ is closed under multiplication. The monomorphism $\phi : Cl_{n,0}(\mathbb{R}) \longrightarrow \mathbb{M}_{2^m}(\mathbb{R})$ can be performed and reverted (within the image) in $O^*(2^n)$ time [11]. However, the construction in [11] is analyzed in the infinite precision model. For the sake of completeness, we revisit this construction and prove the following theorem in Appendix B.

▶ **Theorem 9.** *The multiplication in $Cl_{n,0}(\mathbb{Z})$, with coefficients having poly(n) number of bits, can be performed in time $O^*(2^{\frac{\omega n}{2}})$.*

In order to unify the notation we will represent each element of $Cl_{n,0}(\mathbb{Z})$, that is $\sum_{A\subseteq[1\ldots n]} a_A x_A$, as a function $f : 2^{[1\ldots n]} \longrightarrow \mathbb{Z}$, $f(A) = a_A$. We introduce $\diamond_S$ convolution as an equivalence of multiplication in $Cl_{n,0}(\mathbb{Z})$. The equation (2) can be now rewritten in a more compact form

$$(f \diamond_S g)(X) = \sum_{A \triangle B = X} f(A)g(B)I_{A,B}. \tag{4}$$

4 Non-commutative Subset Convolution

We consider a linearly ordered universe U of size n and functions $f, g : 2^U \longrightarrow \mathbb{Z}$.

▶ **Definition 10.** Let $f \diamond g$ denote *Non-commutative Subset Convolution* (NSC) of functions f, g defined as

$$(f \diamond g)(X) = \sum_{A \uplus B = X} f(A)g(B)I_{A,B}.$$

▶ **Theorem 11.** *NSC on an n-element universe can be performed in time $O^*(2^{\frac{\omega n}{2}})$.*

Proof. Observe that condition $A \uplus B = X$ is equivalent to $A \triangle B = X \wedge |A| + |B| = |X|$ so

$$(f \diamond g)(X) = \sum_{\substack{i+j=|X| \\ i,j\geq 0}} \sum_{A\triangle B = X} f(A)\Big[|A| = i\Big]g(B)\Big[|B| = j\Big]I_{A,B}.$$

Alternatively, we can write

$$(f \diamond g)(X) = \sum_{\substack{i+j=|X| \\ i,j\geq 0}} (f_i \diamond_S g_j)(X),$$

where $f_i(X) = f(X)\Big[|X| = i\Big]$ and likewise for g. The $\diamond_S$ convolution, introduced in (4), is equivalent to multiplication in $Cl_{n,0}(\mathbb{R})$. This means we reduced NSC to $O(n^2)$ multiplications in $Cl_{n,0}(\mathbb{R})$ which could be performed in time $O(2^{\frac{\omega n}{2}})$ according to Theorem 9. ◀

▶ **Observation 12.** *The technique of paying polynomial factor for grouping the sizes of sets will turn useful in further proofs. We will call it size-grouping.*

In our applications we will need to compute a slightly more complex convolution.

▶ **Definition 13.** When f, g are of type $2^U \times 2^U \longrightarrow \mathbb{Z}$ we can define $f \diamond_2 g$ (NSC2) as follows

$$(f \diamond_2 g)(X,Y) = \sum_{\substack{X_1 \uplus X_2 = X \\ Y_1 \uplus Y_2 = Y}} f(X_1,Y_1) g(X_2,Y_2) I_{X_1,X_2} I_{Y_1,Y_2}.$$

▶ **Theorem 14.** *NSC2 on an n-element universe can be performed in time $O^*(2^{\omega n})$.*

Proof. Let us introduce a new universe $U' = U_X \cup U_Y$ of size $2n$ consisting of two copies of U with an order so each element of U_Y is greater than any element of U_X. To underline that $X \subseteq U_X, Y \subseteq U_X$ we will use $\uplus$ notation when summing subsets of U_X and U_Y. In order to reduce NSC2 to NSC on the universe U' we need to replace factor $I_{X_1,X_2} I_{Y_1,Y_2}$ with $I_{X_1 \uplus Y_1, X_2 \uplus Y_2}$. The latter term can be expressed as $I_{X_1,X_2} I_{Y_1,Y_2} I_{X_1,Y_2} I_{Y_1,X_2}$ due to Claim 2. As all elements from $X_i \subseteq U_X$ compare less to elements from $Y_i \subseteq U_Y$ then $I_{X_1,Y_2} = 1$ and I_{Y_1,X_2} depends only on the sizes of Y_1 and X_2. To summarize,

$$I_{X_1,X_2} I_{Y_1,Y_2} = I_{X_1 \uplus Y_1, X_2 \uplus Y_2} (-1)^{|Y_1||X_2|}.$$

To deal with factor $(-1)^{|Y_1||X_2|}$ we have to split the convolution into 4 parts for different parities of $|Y_1|$ and $|X_2|$. We define functions $f', f'_0, f'_1, g', g'_0, g'_1 : 2^{U'} \longrightarrow \mathbb{Z}$ as

$$\begin{aligned}
f'(X \uplus Y) &= f(X,Y), \\
f'_0(X \uplus Y) &= f(X,Y)\Big[\,|Y| \equiv 0 \bmod 2\Big], \\
f'_1(X \uplus Y) &= f(X,Y)\Big[\,|Y| \equiv 1 \bmod 2\Big], \\
g'(X \uplus Y) &= g(X,Y), \\
g'_0(X \uplus Y) &= g(X,Y)\Big[\,|X| \equiv 0 \bmod 2\Big], \\
g'_1(X \uplus Y) &= g(X,Y)\Big[\,|X| \equiv 1 \bmod 2\Big].
\end{aligned}$$

Now we can reduce NSC2 to 4 simpler convolutions.

$$(f \diamond_2 g)(X,Y) = \sum_{\substack{X_1 \uplus X_2 = X \\ Y_1 \uplus Y_2 = Y}} f'(X_1 \uplus Y_1) g'(X_2 \uplus Y_2) I_{X_1 \uplus Y_1, Y_2 \uplus X_2} (-1)^{|Y_1||X_2|} =$$
$$= (f'_0 \diamond g'_0)(X \uplus Y) + (f'_0 \diamond g'_1)(X \uplus Y) + (f'_1 \diamond g'_0)(X \uplus Y) - (f'_1 \diamond g'_1)(X \uplus Y)$$

We have shown that computing NSC2 is as easy as NSC on a universe two times larger. Using Theorem 11 directly gives us the desired complexity. ◀

5 Counting Steiner trees

We will revisit the theorem stated in the aforementioned work.

▶ **Theorem 15** (Bodlaender et al. [4])**.** *There exist algorithms that given a graph G count the number of Steiner trees of size i for each $1 \leq i \leq n-1$ in $O^*(5^{pw})$ time if a path decomposition of width pw is given, and in $O^*(10^{tw})$ time if a tree decomposition of width tw is given.*

Both algorithms use dynamic programming over tree or path decompositions. We introduce some decomposition-based order on V and fix vertex v_1. Let $A = (a_{v,e})_{v \in V, e \in E}$ be an incidence matrix, i.e. for $e = uv$, $u < v$ we have $a_{u,e} = 1$, $a_{v,e} = -1$ and $a_{w,e} = 0$ for

any other vertex w. For each node x of the decomposition we define a function A_x with arguments $0 \leq i \leq n-1$, $s_Y, s_1, s_2 \in \{0,1\}^{B_x}$. The idea is to express the number of Steiner trees with exactly i edges as $A_r(i+1, \emptyset, \emptyset, \emptyset)$.

$$A_x(i, s_Y, s_1, s_2) =$$
$$= \sum_{\substack{Y \subseteq V_x \\ |Y|=i \\ (K \cap V_x) \subseteq Y \\ Y \cap B_x = s_Y^{-1}(1)}} \sum_{X \subseteq E(Y,Y) \cap E_x} \sum_{\substack{f_1 : X \overset{1-1}{\to} Y \setminus \{v_1\} \setminus s_1^{-1}(0) \\ f_2 : X \overset{1-1}{\to} Y \setminus \{v_1\} \setminus s_2^{-1}(0)}} \operatorname{sgn}(f_1)\operatorname{sgn}(f_2) \prod_{e \in X} a_{f_1(e),e} a_{f_2(e),e} \tag{5}$$

As observed in [4] condition $s_Y(v) = 0$ implies that either $s_1(v) = s_2(v) = 0$ or $A_x(i, s_Y, s_1, s_2) = 0$. This means there are at most $n5^{tw}$ triples for which A_x returns a nonzero value.

If a node x has a child y and is of type *introduce vertex*, *introduce edge*, or *forget vertex*, then the function A_x can be computed from A_y in linear time with respect to the number of non-trivial states. Saying this is just a reformulation of Theorem 15 for path decompositions. The only thing that is more difficult for tree decompositions is that they include also *join* nodes having two children each. Here is the recursive formula[2] for A_x for a *join* node x having children y, z.

$$A_x(i, s_Y, s_1, s_2) = \sum_{\substack{i_y + i_z = i + |s_Y^{-1}(1)| \\ s_{1,y} + s_{1,z} = s_1 \\ s_{2,y} + s_{2,z} = s_2}} \begin{matrix} A_y(i_y, s_Y, s_{1,y}, s_{2,y}) A_z(i_z, s_Y, s_{1,z}, s_{2,z}) \\ I_{s_{1,y}^{-1}(1), s_{1,z}^{-1}(1)} I_{s_{2,y}^{-1}(1), s_{2,z}^{-1}(1)} \end{matrix} \tag{6}$$

The next lemma, however not stated explicitly in the discussed work, follows from the proof of Theorem 15 (Theorem 4.4 in [4]).

▶ **Lemma 16.** *Assume there is an algorithm computing all nonzero values of A_x given by* (6) *with running time $f(tw)$. Then the number of Steiner trees of size i in a graph G can be counted in $O^*(\max(f(tw), 5^{tw}))$ time if a tree decomposition of width tw is given.*

We will change notation for our convenience. Each function s_i will be matched with a set $s_i^{-1}(1)$. Let us replace functions A_x, A_y, A_z with h_i, f_i, g_i having first argument fixed and operating on triples of sets. In this setting, the convolution can we written as

$$h_i(A, B, C) = \sum_{\substack{i_y + i_z = i + |A| \\ B_y \uplus B_z = B \\ C_y \uplus C_z = C}} f_{i_y}(A, B_y, C_y) g_{i_z}(A, B_z, C_z) I_{B_y, B_z} I_{C_y, C_z}. \tag{7}$$

Observe that size-grouping allows us to sacrifice a polynomial factor and neglect the restrictions for i, i_y, i_z. Hence, we can work with a simpler formula

$$h(A, B, C) = \sum_{\substack{B_y \uplus B_z = B \\ C_y \uplus C_z = C}} f(A, B_y, C_y) g(A, B_z, C_z) I_{B_y, B_z} I_{C_y, C_z}. \tag{8}$$

The only triples $(s_Y(v), s_1(v), s_2(v))$ allowed for each vertex v are $(0,0,0)$, $(1,0,0)$, $(1,0,1)$, $(1,1,0)$, $(1,1,1)$. In terms of set notation we can say that if $f(A,B,C) \neq 0$ then $B \cup C \subseteq A$. Let $f_A : 2^A \times 2^A \longrightarrow \mathbb{Z}$ be f with the first set fixed, i.e. $f_A(B,C) = f(A,B,C)$.

[2] As confirmed by the authors [5], the formula in [4] for the join node is missing the first argument to the A_x function tracking the number of vertices of a Steiner tree, hence we present a corrected version of this formula.

▶ **Lemma 17.** *For fixed A all values $h(A,B,C)$ can be computed in time $O^*(2^{\omega|A|})$.*

Proof. We want to compute

$$h_A(B,C) = \sum_{\substack{B_y \uplus B_z = B \\ C_y \uplus C_z = C}} f_A(B_y,C_y) g_A(B_z,C_z) I_{B_y,B_z} I_{C_y,C_z} = (f_A \diamond_2 g_A)(B,C),$$

what can be done in time $O^*(2^{\omega|A|})$ according to Theorem 14. ◀

▶ **Lemma 18.** *The convolution (7) can be performed in time $O^*((2^\omega+1)^{tw})$.*

Proof. We use size-grouping to reduce the problem to computing (8). Then we iterate through all possible sets A and take advantage of Lemma 17. The total number of operations (modulo polynomial factor) is bounded by

$$\sum_{A \subseteq U} 2^{\omega|A|} = \sum_{k=0}^{tw} \binom{tw}{k} 2^{\omega k} = (2^\omega+1)^{tw}.$$ ◀

Keeping in mind that (6) and (7) are equivalent and combining Lemmas 16, 18, we obtain the following result.

▶ **Theorem 19.** *The number of Steiner trees of size i in a graph G can be computed in $O^*((2^\omega+1)^{tw})$ time if a tree decomposition of width tw is given.*

▶ Remark. The space complexity of the algorithm is $O^*(5^{tw})$.

Solving the decision version of Feedback Vertex Set can be reduced to the Maximum Induced Forest problem [4]. As observed in [4] the *join* operation for Maximum Induced Forest is analogous to (6).

▶ **Corollary 20.** *The existence of a feedback vertex set of size at most i in a graph G can be determined in $O^*((2^\omega+1)^{tw})$ time if a tree decomposition of width tw is given.*

6 Counting Hamiltonian cycles

Likewise in the previous section, we will start with a previously known theorem.

▶ **Theorem 21** (Bodlaender et al. [4])**.** *There exist algorithms that given a graph G count the number of Hamiltonian cycles in $O^*(6^{pw})$ time if a path decomposition of width pw is given, and in $O^*(15^{tw})$ time if a tree decomposition of width tw is given.*

For each node x of the decomposition a function A_x is defined with arguments $s_1, s_2 \in \{0,1\}^{B_x}$ and $s_{deg} \in \{0,1,2\}^{B_x}$. The idea and notation is analogous to (5). The number of Hamiltonian cycles can be expressed as $A_r(\emptyset,\emptyset,\emptyset)/n$.

$$A_x(s_{deg}, s_1, s_2) =$$
$$= \sum_{\substack{X \subseteq E_x \\ \forall_{v \in (V_x \setminus B_x)} deg_X(v)=2 \\ \forall_{v \in B_x} deg_X(v)=s_{deg}(v)}} \sum_{S \subseteq X} \sum_{\substack{f_1 : S^{1-1}_{\to} V_x \setminus \{v_1\} \setminus s_1^{-1}(0) \\ f_2 : S^{1-1}_{\to} V_x \setminus \{v_1\} \setminus s_2^{-1}(0)}} \operatorname{sgn}(f_1)\operatorname{sgn}(f_2) \prod_{e \in S} a_{f_1(e),e} a_{f_2(e),e} \qquad (9)$$

As observed in [4] we can restrict ourselves only to some subspace of states. When $s_Y(v) = 0$ then all non-zero summands in the (9) satisfy $s_1(v) = s_2(v) = 0$. When $s_Y(v) = 2$ then we can neglect all summands except for those satisfying $s_1(v) = s_2(v) = 1$.

This time there are at most 6^{tw} triples for which A_x returns a nonzero value. We again argue that *introduce vertex*, *introduce edge*, and *forget vertex* nodes can be handled the same way as for the path decomposition and the only bottleneck is formed by *join* nodes. We present a formula for A_x if x is a *join* node with children y, z.

$$A_x(s_{deg}, s_1, s_2) = \sum_{\substack{s_{deg,y}+s_{deg,z}=s_{deg} \\ s_{1,y}+s_{1,z}=s_1 \\ s_{2,y}+s_{2,z}=s_2}} A_y(s_{deg,y}, s_{1,y}, s_{2,y}) A_z(s_{deg,z}, s_{1,z}, s_{2,z}) \, I_{s_{1,y}^{-1}(1), s_{1,z}^{-1}(1)} I_{s_{2,y}^{-1}(1), s_{2,z}^{-1}(1)} \tag{10}$$

Analogously to the algorithm for Steiner Tree, we formulate our claim as a lemma following from the proof of Theorem 21 (Theorem 4.3 in [4]).

▶ **Lemma 22.** *Assume there is an algorithm computing all nonzero values of A_x given by* (10) *with running time $f(tw)$. Then the number of Hamiltonian cycles in a graph G can be counted in $O^*(\max(f(tw), 6^{tw}))$ time if a tree decomposition of width tw is given.*

The only allowed triples of $(s_{deg}(v), s_1(v), s_2(v))$ for each vertex v are $(0,0,0)$, $(1,0,0)$, $(1,0,1)$, $(1,1,0)$, $(1,1,1)$, $(2,1,1)$.

▶ **Lemma 23.** *Assume the equation* (10) *holds. Then it remains true after the following translation of the set of allowed triples $(s_{deg}(v), s_1(v), s_2(v))$.*

$0,0,0 \longrightarrow 0,0,0$
$1,0,0 \longrightarrow 1,0,0$
$1,0,1 \longrightarrow 1,0,1$
$1,1,0 \longrightarrow 0,1,0$
$1,1,1 \longrightarrow 0,1,1$
$2,1,1 \longrightarrow 1,1,1$

Proof. The $I_{\cdot,\cdot}$ factors do not change as we do not modify the coordinates given by functions s_1, s_2. Triples that match in (10) translate into matching triples as the transformation keeps their additive structure. This fact can be seen on the tables below.

	000	100	101	110	111	211
000	000	100	101	110	111	211
100	100	X	X	X	211	X
101	101	X	X	211	X	X
110	110	X	211	X	X	X
111	111	211	X	X	X	X
211	211	X	X	X	X	X

	000	100	101	010	011	111
000	000	100	101	010	011	111
100	100	X	X	X	111	X
101	101	X	X	111	X	X
010	010	X	111	X	X	X
011	011	111	X	X	X	X
111	111	X	X	X	X	X

◀

Therefore we can treat s_{deg} functions as binary ones. We start with unifying the notation binding functions s_i with sets $s_i^{-1}(1)$. Let us replace functions A_x, A_y, A_z with their equivalences h, f, g operating on triples of sets. In this setting, the convolution looks as follows.

$$h(A,B,C) = \sum_{\substack{A_1 \uplus A_2 = A \\ B_1 \uplus B_2 = B \\ C_1 \uplus C_2 = C}} f(A_1, B_1, C_1) g(A_2, B_2, C_2) I_{B_1,B_2} I_{C_1,C_2} \tag{11}$$

Performing convolution (11) within the space of allowed triples is noticeably more complicated than computations in Section 5. Therefore the proof of the following lemma is placed in Appendix C.

▶ **Lemma 24.** *The convolution* (11) *can be computed in time* $O^*((2^\omega + 2)^{tw})$.

This result, together with Lemmas 22 and 23, leads to the main theorem of this section.

▶ **Theorem 25.** *The number of Hamiltonian cycles in a graph* G *can be computed in* $O^*((2^\omega + 2)^{tw})$ *time if a tree decomposition of width* tw *is given.*

▶ Remark. The space complexity of the algorithm is $O^*(6^{tw})$.

7 Conclusions

We have presented the Non-commutative Subset Convolution, a new algebraic tool in algorithmics based on the theory of Clifford algebras. This allowed us to construct faster deterministic algorithms for Steiner Tree, Feedback Vertex Set, and Hamiltonian Cycle, parameterized by the treewidth. As the determinant-based approach applies to all problems solvable by the Cut & Count technique [4, 8], the NSC can improve running times for a larger class of problems.

The first open question is whether the gap between time complexities for the decision and counting versions of these problems could be closed. Or maybe one can prove this gap inevitable under a well-established assumption, e.g. SETH?

The second question asked is if it is possible to prove a generic theorem so the lemmas like 18 or 24 would follow from it easily. It might be possible to characterize convolution algebras that are semisimple and algorithmically construct isomorphisms with their canonical forms described by the Artin-Wedderburn theorem.

The last question is what other applications of Clifford algebras and Artin-Wedderburn theorem can be found in algorithmics.

Acknowledgements. I would like to thank Marek Cygan for pointing out the bottleneck of the previously known algorithms and for the support during writing this paper. I would also like to thank Paul Leopardi for helping me understand the fast Fourier-like transform for Clifford algebras.

References

1 John A. Beachy. *Introductory lectures on rings and modules*, volume 47. Cambridge University Press, 1999.

2 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: Fast subset convolution. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, STOC'07, pages 67–74, New York, NY, USA, 2007. ACM. doi:10.1145/1250790.1250801.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

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

5 Marek Cygan. Private communication, 2016.

6 Marek Cygan, Fedor Fomin, Bart MP Jansen, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Open problems for fpt school 2014. URL: `http://fptschool.mimuw.edu.pl/opl.pdf`.

7 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, pages 301–310. ACM, 2013.

8 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 150–159. IEEE, 2011.

9 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Representative sets of product families. In *Algorithms – ESA 2014*, pages 443–454. Springer, 2014.

10 Ton Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994.

11 Paul Leopardi. A generalized FFT for Clifford algebras. *Bulletin of the Belgian Mathematical Society*, 11(5):663–688, 03 2005.

12 David K. Maslen and Daniel N. Rockmore. Generalized ffts – a survey of some recent results. In *Groups and Computation II*, volume 28, pages 183–287. American Mathematical Soc., 1997.

13 Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

14 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. *Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution*, pages 566–577. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. `doi:10.1007/978-3-642-04128-0_51`.

15 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing*, pages 887–898. ACM, 2012.

A Associative algebras

This section is not crucial to understanding the paper but it provides a bigger picture of the applied theory. We assume that readers are familiar with basic algebraic structures like rings or fields. More detailed introduction can be found, e.g. in [1].

▶ **Definition 26.** A linear space A over a field K (or, more generally, a module over a ring K) is called an *associative algebra* if it admits a multiplication operator $A \times A \to A$ satisfying the following conditions:

1. $\forall_{a,b,c \in A}\ a(bc) = (ab)c$,
2. $\forall_{a,b,c \in A}\ a(b+c) = ab + ac,\ (b+c)a = ba + ca$,
3. $\forall_{a,b \in A, k \in K}\ k(ab) = (ka)b = a(kb)$.

A set $W \subseteq A$ is called a *generating set* if every element of A can be obtained from W by addition and multiplication. The elements of W are called *generators*. It is easy to see that multiplication defined on a generating set extends in an unambiguous way to the whole algebra. We will often abbreviate the term *associative* as we will study only such algebras.

▶ **Definition 27.** The product of algebras $A_1, A_2, \ldots, A_m$ is an algebra $A_1 \otimes A_2 \otimes \cdots \otimes A_m$ with multiplication performed independently on each coordinate.

▶ **Definition 28.** For algebras A, B over a ring K, function $\phi : A \to B$ is called a *homomorphism of algebras* if it satisfy the following conditions:

1. $\forall_{a,b\in A}\ \phi(a+b)=\phi(a)+\phi(b)$,
2. $\forall_{a,b\in A}\ \phi(ab)=\phi(a)\phi(b)$,
3. $\forall_{a\in A,k\in K}\ \phi(ka)=k\phi(a)$.

If ϕ is reversible within its image then we call it a *monomorphism* and if additionally $\phi(A)=B$ then we call ϕ an *isomorphism*

Monomorphisms of algebras turn out extremely useful when multiplication in algebra B is simpler than multiplication in A, because we can compute ab as $\phi^{-1}\big(\phi(a)\phi(b)\big)$. This observation is used in Theorem 9 and Lemmas 24, 32. For a better intuition, we depict the various ways of performing multiplication on diagrams (3), (14).

▶ **Definition 29.** A subset M of algebra A is called a *simple left module* if

1. $\forall_{a\in A,b\in M}\ ab\in M$,
2. $\forall_{b,c\in M}\ b+c\in M$,

and the only proper subset of M with these properties is $\{0\}$.

The next definition is necessary to exclude some cases of obscure algebras.

▶ **Definition 30.** An algebra A is called *semisimple* if there is no non-zero element a so for every simple left module $M\subseteq A$ the set $aM=\{ab\,|\,b\in M\}$ is $\{0\}$.

The theorem below was proven in full generality for algebras over arbitrary rings but we will formulate its simpler version for fields.

▶ **Theorem 31** (Artin-Wedderburn [1])**.** *Every finite-dimensional associative semisimple algebra A over a field K is isomorphic to a product of matrix algebras*

$$A\cong M_{n_1}(K_1)\otimes M_{n_2}(K_2)\otimes\cdots\otimes M_{n_m}(K_m),$$

where K_i are fields containing K.

The related isomorphism is called a generalized Fourier transform (GFT) for A. If we are able to perform GFT efficiently then we can reduce computations in A to matrix multiplication. For some classes of algebras, e.g. abelian group algebras [12], there are known algorithms for GFT with running time $O(n\log n)$ where $n=\dim A$.

If the field K is algebraically closed (e.g. $\mathbb{C}$) then all $K_i=K$ and $\sum_{i=1}^m n_i^2$ equals the dimension of A. If the algebra A is commutative then all $n_i=1$ and A is isomorphic to a product of fields. This is actually the case in the Fast Subset Convolution [2] where the isomorphism is given by the Möbius transform.

B Proof of Theorem 9

Proof. The transformation ϕ can be computed and reverted (within the image) in time $O^*(2^n)$ assuming infinite precision and $O(1)$ time for any arithmetic operation [11]. In order to compute ϕ accurately, we need to look inside the paper [11].

Transformation ϕ can be represented as $\phi=\gamma\circ\upsilon$ where υ is a monomorphic embedding into another Clifford algebra and γ is an isomorphism with the matrix algebra. We modify isomorphism diagram (3) to show these mappings in more detail.

$$\begin{array}{ccccccc}
Cl_{n,0}(\mathbb{Z}) & \hookrightarrow & Cl_{n,0}(\mathbb{R}) & \xrightarrow{\upsilon} & Cl_{m,m}(\mathbb{R}) & \xrightarrow{\gamma} & \mathbb{M}_{2^m}(\mathbb{R}) \\
\downarrow * & & \downarrow * & & \downarrow * & & \downarrow * \\
Cl_{n,0}(\mathbb{Z}) & \hookrightarrow & Cl_{n,0}(\mathbb{R}) & \xrightarrow{\upsilon} & Cl_{m,m}(\mathbb{R}) & \xrightarrow{\gamma} & \mathbb{M}_{2^m}(\mathbb{R})
\end{array}$$

We begin with embedding $\upsilon : Cl_{n,0}(\mathbb{R}) \longrightarrow Cl_{m,m}(\mathbb{R})$ where $m = \frac{n}{2} + O(1)$ (see Definition 4.4 in [11]). Transformation υ is just a translation of basis so no arithmetic operations are required.

For the sake of disambiguation, we indicate the domain of the function γ with a lower index: $\gamma_k : Cl_{k,k}(\mathbb{R}) \longrightarrow \mathbb{M}_{2^k}(\mathbb{R})$. In the k-th step, we construct a matrix representation of $y \in Cl_{k,k}(\mathbb{R})$. Let y^+, y^- denote the projections of y onto subspaces spanned by products of respectively even and odd number of generators. Of course, $y = y^+ + y^-$ and $\gamma_k(y) = \gamma_k(y^+) + \gamma_k(y^-)$. Such an element y can be represented as $y = a + b\mathbf{x}_- + c\mathbf{x}_+ + d\mathbf{x}_-\mathbf{x}_+$ for $\mathbf{x}_+, \mathbf{x}_-$ being the first and the last generator ($\mathbf{x}_+^2 = e$, $\mathbf{x}_-^2 = -e$) and $a, b, c, d \in Cl_{k-1,k-1}(\mathbb{R})$. Now we can apply the recursive formula from Theorem 5.2 in [11]:

$$\gamma_k(y^+) = \gamma_{k-1}\left(\begin{bmatrix} a^+ - d^+ & -b^- - c^- \\ -b^- - c^- & a^+ + d^+ \end{bmatrix}\right), \quad \gamma_k(y^-) = \gamma_{k-1}\left(\begin{bmatrix} a^- - d^- & -b^+ + c^+ \\ b^+ + c^+ & -a^- - d^- \end{bmatrix}\right),$$

where $\gamma_{k-1}(M)$ stands for a block matrix with γ_{k-1} applied to each element of M.

We see that computing $\big(\gamma_k(y^+), \gamma_k(y^-)\big)$ can be reduced to computing 4 analogous pairs for $k-1$ and combining them using addition and subtraction. Hence, the coefficients of the obtained matrix will also be integers with $poly(n)$ number of bits and the total number of arithmetic operations is $O(m4^m) = O(n2^n)$.

The inverse transform γ^{-1} is also computed in m steps and we continue using lower index to indicate the domain alike for the forward transform. Let $Y \in \mathbb{M}_{2^k}(\mathbb{Z})$ and

$$Y = \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix}, \quad y_{ij} = \gamma_{k-1}^{-1}(Y_{ij}).$$

Then from Theorem 7.1 in [11] we know that

$$\gamma_k^{-1}(Y) = \frac{1}{2}\big((\hat{y_{22}} + y_{11}) + (\hat{y_{21}} - y_{12})\mathbf{x}_- + (\hat{y_{21}} + y_{12})\mathbf{x}_+ + (\hat{y_{22}} - y_{11})\mathbf{x}_-\mathbf{x}_+\big),$$

where $\hat{y} = y^+ - y^-$ and the rest of notation is as above. We can reduce computing γ_k^{-1} to 4 queries from $(k-1)$-th step so the total number of arithmetic operations is $O(m4^m) = O(n2^n)$.

This time the coefficients at each step are given as sums of elements from the previous step divided by 2. We do not need to prove that they remain integer at all steps because we can postpone the division until the last step. As long as $\gamma^{-1}(Y)$ is a product of two elements from $Cl_{m,m}(\mathbb{Z})$, it is guaranteed that the numbers in the last step would be divisible by 2^m. What is more, if we know that $\gamma^{-1}(Y) \in \upsilon(Cl_{n,0}(\mathbb{Z}))$ then we can revert the υ transform and obtain $\phi^{-1}(Y)$.

We have proven that we can switch representation between $Cl_{n,0}(\mathbb{Z})$ and $\mathbb{M}_{2^m}(\mathbb{Z})$ in time $O^*(2^n)$. The multiplication in $\mathbb{M}_{2^m}(\mathbb{Z})$ for inputs of $poly(n)$ size can be performed in time complexity $O^*(2^{\omega m}) = O^*(2^{\frac{\omega n}{2}})$ and the resulting matrix also contains only $poly(n)$-bits integers. This proves that the multiplication in $Cl_{n,0}(\mathbb{Z})$ admits an algorithm with running time $O^*(2^{\frac{\omega n}{2}})$. ◀

C Proof of Lemma 24

This section reduces the complicated algorithm for Hamiltonian Cycle to two isomorphism theorems and we suggest reading Appendix A first. Our goal is to compute values of h for the allowed triples assuming that non-zero values of f, g also occur only for the allowed triples.

$$h(A,B,C) = \sum_{\substack{A_1 \uplus A_2 = A \\ B_1 \uplus B_2 = B \\ C_1 \uplus C_2 = C}} f(A_1,B_1,C_1)g(A_2,B_2,C_2)I_{B_1,B_2}I_{C_1,C_2} \tag{12}$$

Taking advantage of the size-grouping technique (see Observation 12) we can replace condition $A_1 \uplus A_2 = A$ with $A_1 \cup A_2 = A$ and focus on the following convolution.

$$(f \odot g)(A,B,C) = \sum_{\substack{A_1 \cup A_2 = A \\ B_1 \uplus B_2 = B \\ C_1 \uplus C_2 = C}} f(A_1,B_1,C_1) g(A_2,B_2,C_2) I_{B_1,B_2} I_{C_1,C_2} \tag{13}$$

Let Ham be a subspace of $2^U \times 2^U \times 2^U \longrightarrow \mathbb{Z}$ given by functions admitting only the allowed triples (see Lemma 23), i.e. $f \in Ham \wedge f(A,B,C) \neq 0$ implies $A \cap (B \triangle C) = C \backslash B$. Observe that Ham is closed under the $\odot$ operation so it can be regarded as a 6^{tw}-dimensional algebra. Let H_D be an algebra over space $2^{U \backslash D} \times 2^D \times 2^D \longrightarrow \mathbb{Z}$ with multiplication given by the $\oslash$ operator defined as

$$(f \oslash g)(E,B,C) = \sum_{\substack{E_1 \uplus E_2 = E \\ B_1 \uplus B_2 = B \\ C_1 \uplus C_2 = C}} f(E_1,B_1,C_1) g(E_2,B_2,C_2) I_{B_1,B_2} I_{C_1,C_2} (-1)^{|E_1|(|B_2|+|C_2|)}.$$

We want to show that Ham is isomorphic (see Definition 28) with a product of all H_D for $D \subseteq U$ (see Definition 27). In particular, diagram (14) commutes.

$$\begin{array}{ccc} Ham & \xrightarrow{\quad\tau\quad} & \bigotimes_{D \subseteq U} H_D \\ \downarrow \odot & & \downarrow \oslash \\ Ham & \xrightarrow{\quad\tau\quad} & \bigotimes_{D \subseteq U} H_D \end{array} \tag{14}$$

where $\tau_D : Ham \longrightarrow H_D$ is given as

$$(\tau_D f)(E,B,C) = I_{B,E} I_{C,E} \sum_{A \subseteq D} f(A, B \cup E, C \cup E).$$

▶ **Lemma 32.** *Transform τ and its inverse can be performed in time $O^*(6^{tw})$.*

▶ **Corollary 33.** *Transformation τ is reversible.*

▶ **Lemma 34.** *Given $f, g \in H_D$ we can compute $f \oslash g$ in time $O^*(2^{\omega|D|} 2^{|U \backslash D|})$.*

▶ **Lemma 35.** *Diagram (14) commutes, i.e. τ is a homomorphism of algebras.*

▶ **Corollary 36.** *Transformation τ is an isomorphism of algebras.*

As for the Clifford algebras, we can switch the representation of the algebra to perform multiplication in the simpler one, and then revert the isomorphism to get the result. The most time consuming part of the algorithm is performing the $\oslash$ convolutions. Total number of operations modulo polynomial factor can be bounded with Lemma 34 by

$$\sum_{D \subseteq U} 2^{\omega|D|} 2^{|U \backslash D|} = \sum_{k=0}^{tw} \binom{tw}{k} 2^{\omega k} 2^{tw-k} = (2^\omega + 2)^{tw}. \tag{15}$$

The rest of the appendix is devoted to proving Lemmas 32, 34, 35.

Proof of Lemma 32. For fixed sets B, C let $H = B \cap C$, $F = B \triangle C$, $B_1 = B \backslash C$, $C_1 = C \backslash B$. Observe that every allowed triple (A,B,C) must satisfy $A \cap F = C_1$. Therefore we can represent Ham as a union of sets

$$T_{B_1,C_1,H} = \Big\{ (A_1 \cup C_1, B_1 \cup H, C_1 \cup H) \;\Big|\; A_1 \subseteq U \backslash (B_1 \cup C_1) \Big\}.$$

for all pairwise disjoint triples $B_1, C_1, H \subseteq U$. Functions over $T_{B_1,C_1,H}$ can be parameterized with only the A_1 argument. Consider following transformation over function space on $T_{B_1,C_1,H}$.

$$(\gamma_{B_1,C_1,H} f)(A_1) = \sum_{A_0 \subseteq A_1} f(A_0 \cup C_1, B_1 \cup H, C_1 \cup H)$$

Transform $\gamma_{B_1,C_1,H}$ is just the Möbius transform, therefore it can be performed and reverted in time $O^*(2^{|U\setminus(B_1\cup C_1)|})$ (see Theorem 5). Values of γf correspond directly to values of τf.

$$\begin{aligned}(\tau_D f)(E,B,C) &= I_{B,E} I_{C,E} \sum_{A \subseteq D} f(A, B \cup E, C \cup E) = \\ &= I_{B,E} I_{C,E} \sum_{A \subseteq D} f(A, B_1 \cup H \cup E, C_1 \cup H \cup E) = \\ &= I_{B,E} I_{C,E} \sum_{A_0 \subseteq D\setminus F} f(A_0 \cup C_1, B_1 \cup H \cup E, C_1 \cup H \cup E) = \\ &= I_{B,E} I_{C,E} (\gamma_{B_1,C_1,H\cup E} f)(D\setminus F)\end{aligned}$$

$$\begin{aligned}(\gamma_{B_1,C_1,H} f)(A_1) &= \sum_{A_0 \subseteq A_1} f(A_0 \cup C_1, B_1 \cup H, C_1 \cup H) = \\ &= \sum_{A_0 \subseteq A_1 \cup C_1} f(A_0, B_1 \cup H, C_1 \cup H) = \\ &= \sum_{A_0 \subseteq A_1 \cup C_1} f\big(A_0, B_2 \cup (H\setminus A_1), C_2 \cup (H\setminus A_1)\big) = \\ &= (\tau_{A_1\cup C_1} f)(E, B_2, C_2) I_{B_2,E} I_{C_2,E}\end{aligned}$$

where $E = H\setminus A_1$, $B_2 = B_1 \cup (H \cap A_1)$, $C_2 = C_1 \cup (H \cap A_1)$ are valid arguments of $\tau_{A_1\cup C_1}$.

To estimate the total number of operations consider all choices of F. The partition into $F = B_1 \uplus C_1$ can be done in $2^{|F|}$ ways, the set H can be chosen in $2^{|U\setminus F|}$ ways, and for such triple we have to perform the $\gamma_{B_1,C_1,H}$ transform (or its inverse) what involves $O^*(2^{|U\setminus F|})$ operations. Hence, the total running time (modulo polynomial factors) is

$$\sum_{F \subseteq U} 2^{|F|} 4^{|U\setminus F|} = \sum_{k=0}^{tw} \binom{tw}{k} 2^k 4^{tw-k} = 6^{tw}. \qquad \blacktriangleleft$$

Proof of Lemma 34. Applying the size-grouping (see Observation 12) allows us to neglect the $(-1)^{|E_1|(|B_2|+|C_2|)}$ factor and replace condition $E_1 \uplus E_2 = E$ with $E_1 \cup E_2 = E$. Therefore it suffices to perform the $\odot$ convolution on H_D (the same as in (13)).

$$(f \odot g)(E,B,C) = \sum_{\substack{E_1\cup E_2 = E \\ B_1 \uplus B_2 = B \\ C_1 \uplus C_2 = C}} f(E_1, B_1, C_1) g(E_2, B_2, C_2) I_{B_1,B_2} I_{C_1,C_2}.$$

Let us denote

$$(\mu_E f)(B,C) = \sum_{F \subseteq E} f(F,B,C).$$

Transform μ and its inverse can be computed using Möbius transform (see Theorem 5) in time $O^*(2^{|U\setminus D|})$ for all E and a fixed pair of sets B, C. We perform it for all $4^{|D|}$ such pairs.

It turns out that μ is an isomorphism between $(H_D, \odot)$ and a product of all algebras given by images of μ_E for $E \subseteq U \setminus D$ (see Definitions 27, 28) with multiplication given by NSC2, i.e. $(\mu_E f) \diamond_2 (\mu_E g) = \mu_E(f \odot g)$. We can again switch the representation of the algebra, multiply the elements, and then revert the isomorphism. The computations below show that μ is a homomorphism of algebras and we know already that μ is reversible.

$$\begin{aligned}
\big((\mu_E f) \diamond_2 (\mu_E g)\big)(B, C) = \\
&= \sum_{\substack{B_1 \uplus B_2 = B \\ C_1 \uplus C_2 = C}} (\mu_E f)(B_1, C_1)(\mu_E g)(B_2, C_2) I_{B_1,B_2} I_{C_1,C_2} = \\
&= \sum_{\substack{E_1, E_2 \subseteq E \\ B_1 \uplus B_2 = B \\ C_1 \uplus C_2 = C}} f(E_1, B_1, C_1) g(E_2, B_2, C_2) I_{B_1,B_2} I_{C_1,C_2} = \\
&= \sum_{F \subseteq E} \sum_{\substack{E_1 \cup E_2 = F \\ B_1 \uplus B_2 = B \\ C_1 \uplus C_2 = C}} f(E_1, B_1, C_1) g(E_2, B_2, C_2) I_{B_1,B_2} I_{C_1,C_2} = \\
&= \big(\mu_E(f \odot g)\big)(B, C)
\end{aligned}$$

To perform multiplication of $\mu(a)$ and $\mu(b)$, where $a, b \in H_D$, we have to perform NSC2 $\big(O^*(2^{\omega|D|})$ time complexity, see Theorem 14$\big)$ for each $E \subseteq U \setminus D$, what results in desired running time. ◀

Proof of Lemma 35. We need to show that for each $B, C \subseteq D, D \cap E = \emptyset$ it is $(\tau_D(f \odot g))(E, B, C) = ((\tau_D f) \oslash (\tau_D g))(E, B, C)$. Let us start with unrolling the formula for $\tau_D(f \odot g)$. Keeping in mind that $B \cap E = C \cap E = \emptyset$ we can see that

$$\begin{aligned}
&(\tau_D(f \odot g))(E, B, C) = \\
&= \sum_{A \subseteq D} (f \odot g)(A, B \cup E, C \cup E) I_{B,E} I_{C,E} = \\
&= \sum_{\substack{A_1, A_2 \subseteq D \\ B_1 \uplus B_2 = B \\ E_1 \uplus E_2 = E \\ C_1 \uplus C_2 = C \\ F_1 \uplus F_2 = E}} \begin{aligned}[t] & f(A_1, B_1 \cup E_1, C_1 \cup F_1) g(A_2, B_2 \cup E_2, C_2 \cup F_1) \\ & I_{B_1 \cup E_1, B_2 \cup E_2} I_{C_1 \cup F_1, C_2 \cup F_2} I_{B,E} I_{C,E}. \end{aligned}
\end{aligned} \tag{16}$$

On the other hand, we have

$$\begin{aligned}
&((\tau_D f) \oslash (\tau_D g))(E, B, C) = \\
&= \sum_{\substack{E_1 \uplus E_2 = E \\ B_1 \uplus B_2 = B \\ C_1 \uplus C_2 = C}} (\tau_D f)(E_1, B_1, C_1)(\tau_D g)(E_2, B_2, C_2) I_{B_1,B_2} I_{C_1,C_2} (-1)^{|E_1|(|B_2|+|C_2|)} = \\
&= \sum_{\substack{A_1, A_2 \subseteq D \\ E_1 \uplus E_2 = E \\ B_1 \uplus B_2 = B \\ C_1 \uplus C_2 = C}} \begin{aligned}[t] & f(A_1, B_1 \cup E_1, C_1 \cup E_1) g(A_2, B_2 \cup E_2, C_2 \cup E_2) \\ & I_{B_1,B_2} I_{C_1,C_2} I_{B_1,E_1} I_{C_1,E_1} I_{B_2,E_2} I_{C_2,E_2} (-1)^{|E_1|(|B_2|+|C_2|)}. \end{aligned}
\end{aligned} \tag{17}$$

We want to argue that all non-zero summands of (16) satisfy $E_1 = F_1, E_2 = F_2$. Indeed, let us assume $v \in F_1 \setminus E_1$. As $v \in E$ so $v \notin D \supseteq A, B, C$ and $\big([v \in A_1], [v \in B_1 \cup E_1], [v \in C_1 \cup F_1]\big) = (0, 0, 1)$ which is not a valid triple what implies $f(A_1, B_1 \cup E_1, C_1 \cup F_1) = 0$.

Assumption $v \in E_1 \backslash F_1$ leads to $\big([v \in A_1], [v \in B_1 \cup E_1], [v \in C_1 \cup F_1]\big) = (0,1,0)$ but $v \in E = E_1 \uplus E_2 = F_1 \uplus F_2$ so $\big([v \in A_2], [v \in B_2 \cup E_2], [v \in C_2 \cup F_2]\big) = (0,0,1)$ and $g(A_2, B_2 \cup E_2, C_2 \cup F_1) = 0$. The same arguments can be used if $v \in E_2 \triangle F_2$.

Now we just need to prove that for $E_1 = F_1, E_2 = F_2$ the I factors in (16) and (17) are equivalent. We apply Claim 2 to $I_{B_1 \cup E_1, B_2 \cup E_2} I_{C_1 \cup E_1, C_2 \cup E_2}$. We can omit factor $I^2_{E_1,E_2} = 1$ as well as $I_{B_1,B_2} I_{C_1,C_2}$ appearing also in (17). What is left to prove is that

$$I_{B_1,E_2} I_{E_1,B_2} I_{B,E} = I_{B_1,E_1} I_{B_2,E_2} (-1)^{|E_1||B_2|},$$
$$I_{C_1,E_2} I_{E_1,C_2} I_{C,E} = I_{C_1,E_1} I_{C_2,E_2} (-1)^{|E_1||C_2|}.$$

According to Claim 1 we can replace $I_{E_1,B_2}(-1)^{|E_1||B_2|}$ with I_{B_2,E_1} what reduces the formula in the first row to Claim 2 for $B = B_1 \uplus B_2$, $E = E_1 \uplus E_2$. Applying analogous observation to the second row finishes the proof. ◀

The First Parameterized Algorithms and Computational Experiments Challenge

Holger Dell[1], Thore Husfeldt[2], Bart M. P. Jansen[3], Petteri Kaski[4], Christian Komusiewicz[5], and Frances A. Rosamond[6]

2 **Saarland University, Saarbrücken, Germany; and**
Cluster of Excellence "Multimodal Computing and Interaction" (MMCI), Saarbrücken, Germany
hdell@mmci.uni-saarland.de
2 **ITU Copenhagen, Denmark, and Lund University, Sweden**
thore@itu.dk
3 **Eindhoven University of Technology, The Netherlands**
b.m.p.jansen@tue.nl
4 **Aalto University, Finland**
petteri.kaski@aalto.fi
5 **Friedrich-Schiller-University Jena, Germany**
christian.komusiewicz@uni-jena.de
6 **University of Bergen, Norway**
frances.rosamond@uib.no

Abstract

In this article, the steering committee of the Parameterized Algorithms and Computational Experiments challenge (PACE) reports on the first iteration of the challenge. Where did PACE come from, how did it go, who won, and what's next?

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, K.7.2 Organizations

Keywords and phrases treewidth, feedback vertex set, contest, implementation challenge, FPT

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.30

1 Introduction

The Parameterized Algorithms and Computational Experiments Challenge (PACE) was conceived in Fall 2015 when many FPT researchers gathered at the Simons Institute for the Theory of Computing at UC Berkeley. A talk there [14] explored the practical implementability of theoretically tight FPT results, which seemed to offer an area for further investigation. PACE was born from a belief that a challenge could help deepen the relationship between parameterized algorithmics and practice. It was partially inspired by the success of SAT-solving competitions in neighboring communities. The goal of PACE is to investigate the applicability of algorithmic ideas studied and developed in the subfields of multivariate, fine-grained, parameterized, or fixed-parameter tractable algorithms. In particular, it aims to:

1. Bridge between algorithm design and analysis theory and algorithm engineering practice.
2. Inspire new theoretical developments.
3. Investigate the competitiveness of analytical and design frameworks developed in the communities.

4. Produce universally accessible libraries of implementations and repositories of benchmark instances.
5. Encourage dissemination of the findings in scientific papers.

Discussions throughout the community led to a website [18], a steering committee, and two challenge tracks for 2016 with program committees: Track A (Treewidth) and Track B (Feedback Vertex Set). The winners were announced at IPEC 2016 in Aarhus.

2 Competition track A: Treewidth

The treewidth of a graph is an important graph parameter, the theory and complexity of which has been intensely study in graph minor theory and fixed-parameter tractability (FPT). Given a graph G and an integer k, it is NP-complete to determine whether the treewidth of G is at most k, but there is an $\mathcal{O}(n^{k+2})$-time algorithm [1]. The problem can also be solved in FPT-time $2^{\mathcal{O}(k^3)}n$ [3], and a factor-5 approximation can be obtained in time $2^{\mathcal{O}(k)}n$ [4]. It is unknown whether the problem has a polynomial-time approximation scheme (PTAS).

Treewidth implementations are used in various contexts. For example, compilers allocate registers by computing proper colorings on control flow graphs, which turn out to have small treewidth in practice (e.g. [24]). Data structures for shortest path queries can use tree decompositions in a preprocessing phase (e.g. [5]). Graph theory can be guided by treewidth implementations when attempting to rigorously determine the treewidth of specific graph families (e.g. [15]). Finally, many problems in probabilistic inference use tree decompositions in a preprocessing phase (e.g. [13]).

While some treewidth implementations existed before PACE 2016, they were not easily accessible and sometimes buggy (as in the case of the Python SAGE implementation, which can produce non-optimal solutions), and their performances have never been compared in public. For PACE, we imposed a unified input/output format for the challenge and required all implementations to be made available on GitHub. Moreover, the details and raw data of all results mentioned in this document can be found in the GitHub repository [25] that we published. Using the tools and benchmark instances in the repository, it is a trivial matter to reproduce the results.

2.1 Submissions

The list of all implementation submissions is available online [16]. Two researchers submitted real-world instances for benchmarking:

1. Johannes Fichte (TU Wien) submitted transit networks.
2. Ben Strasser (Karlsruhe Institute of Technology) submitted road graphs.

Track A was subdivided into two tracks, based on the distinction between exact algorithms and heuristics. Each track was further subdivided into two subchallenges, based on the distinction between parallel and sequential algorithms. Since the best sequential exact algorithm outperformed the best parallel exact algorithm, the exact parallel subchallenge was discarded.

2.2 Sequential algorithms for computing treewidth exactly

The goal of this challenge was to compute a tree decomposition of minimum width. Three teams participated in this track, and two PACE co-organizers jointly contributed a further implementation. Figure 1 summarizes the results. In total, we used 200 instances in the

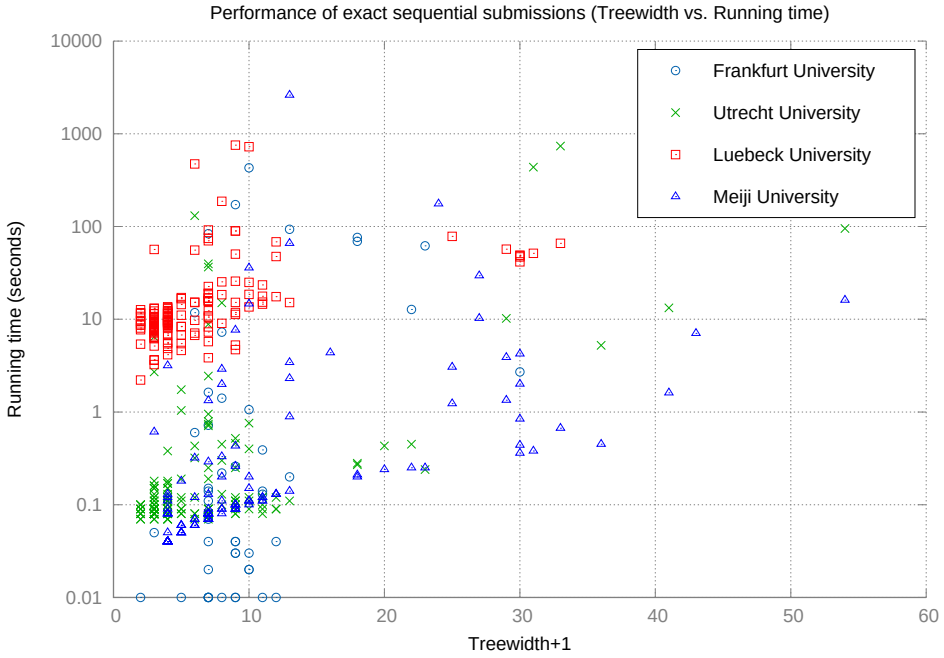


Figure 1 Results for exact sequential algorithms for computing treewidth. This plot shows one data point per solver-instance pair. The x-coordinate corresponds to the treewidth of the instance and the y-coordinate corresponds to the running time. We aborted the computation after a timeout of 100 seconds for most instances; some instances had a timeout of 1000 and 3600 seconds.

exact competition. The instances are samples of named graphs [21], control flow graphs [20], and DIMACS graph coloring instances [7]. The outcome of this challenge is:

1st prize, 350 €: Hisao Tamaki (Meiji University) solved 199 of 200 instances. The submission is written in C++ and is based on a modified version of the brute force approach of Arnborg et al. [1]. [`https://github.com/TCS-Meiji/treewidth-exact`]

2nd prize, 125 €: Hans Bodlaender and Tom Van der Zanden (Utrecht University) solved 173 of 200 instances. The submission is written in C# / Mono and relies on balanced separators as well as dynamic programming. [`https://github.com/TomvdZanden/BZTreewidth`]

3rd prize, 75 €: Max Bannach, Sebastian Berndt, and Thorsten Ehlers (Luebeck University) solved 166 of 200 instances. The submission is written in Java 8 and relies on a SAT-solver to find the optimal elimination order. [`https://github.com/maxbannach/Jdrasil`]

The implementation by Larisch and Salfelder from the track A program committee (Frankfurt University) solved 171 of 200 instances.

2.3 Heuristic algorithms for computing treewidth

The goal of this challenge was to compute a good tree decomposition in a given fixed timeout. We set the timeout to 100 seconds for every instance. We used the 200 instances from the exact competition and 81 additional, harder instances from the same sources; arguably, the instances were generally too easy for the heuristic challenge. Seven teams participated in this track. In total, 6 sequential programs and 3 parallel programs were submitted. Some teams submitted multiple programs, in which case we only kept the best-performing submission

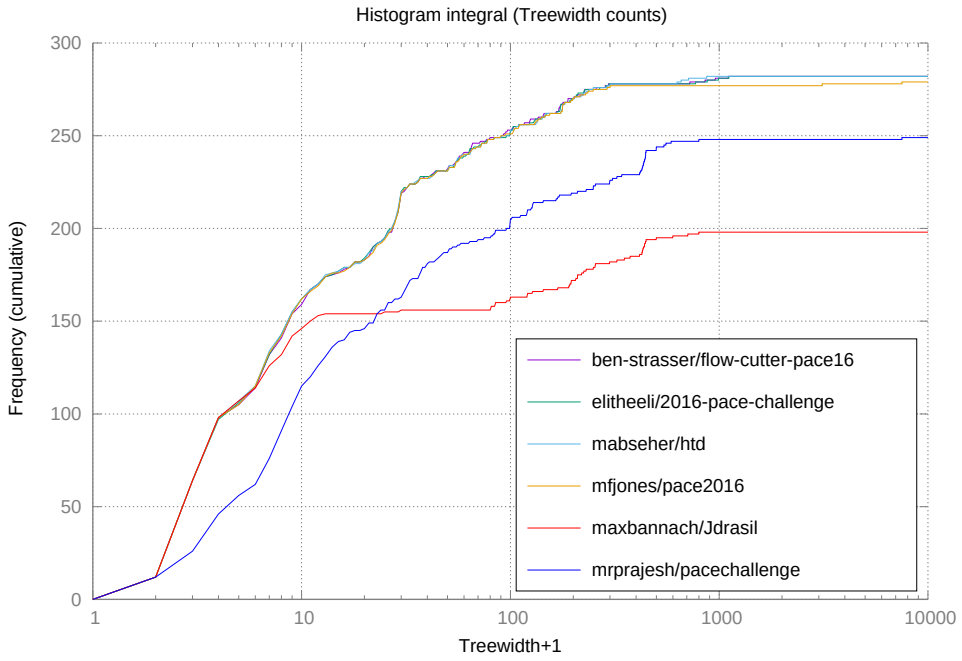


Figure 2 Results for heuristic sequential algorithms for computing treewidth. This plot shows, for every sequential submission and every x, the number y of instances for which the submission computed a tree decomposition of width at most x within the given time limit.

of each team in the ranking. For the evaluation, we viewed each instance as a voter and determined its ranking for the implementations from the width of the tree decomposition produced by the implementation after 100 seconds. We combined these votes using the Schulze method [23]. This process can be inspected in the testbed repository [22].

2.3.1 Sequential heuristic algorithms for computing treewidth

Figure 2 shows the results for sequential heuristic algorithms. As can be seen, the top three submissions nearly converge on the employed metric; this is perhaps explained by the fact that all three implement the basic minimum fill-in heuristic (tweaked in different ways). The other three submissions use more interesting techniques from FPT. The final ranking is:

1st prize, 350 €: Ben Strasser (Karlsruhe Institute of Technology) [`https://github.com/ben-strasser/flow-cutter-pace16`]
2nd prize, 125 €: Eli Fox-Epstein (Brown University) [`https://github.com/elitheeli/2016-pace-challenge`]
3rd prize, 75 €: Michael Abseher, Nysret Musliu, and Stefan Woltran (TU Wien) [`https://github.com/mabseher/htd`]
4th place: Serge Gaspers, Joachim Gudmundsson, Mitchell Jones, Julian Mestre, and Stefan Rümmele (UNSW and University of Sidney) [`https://github.com/mfjones/pace2016`]
5th place: Max Bannach, Sebastian Berndt, and Thorsten Ehlers (Luebeck University) [`https://github.com/maxbannach/Jdrasil`]
6th place: Kaustubh Joglekar, Akshay Kamble, and Rajesh Pandian (IIT Madras) [`https://github.com/mrprajesh/pacechallenge`]

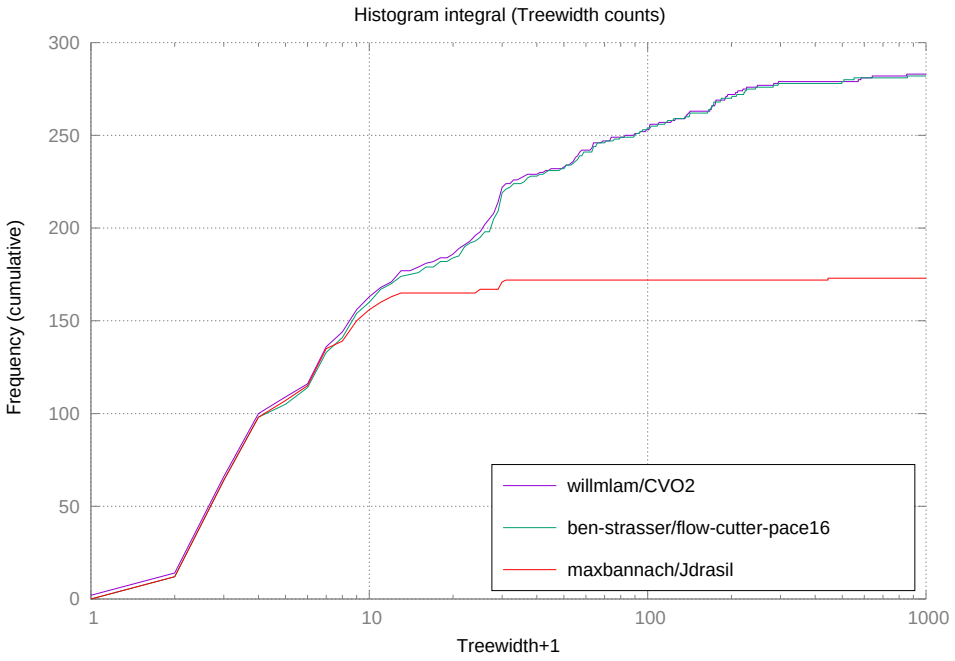


Figure 3 Results for heuristic parallel algorithms for computing treewidth. This plot shows, for every sequential submission and every x, the number y of instances for which the submission computed a tree decomposition of width at most x within the given time limit.

2.3.2 Parallel heuristic algorithms for computing treewidth

The results for parallel heuristic algorithms can be found in Figure 3. It leads to the following ranking:

1st prize, 350 €: Kalev Kask and William Lam (University of California at Irvine) [`https://github.com/willmlam/CVO2`]

2nd prize, 125 €: Ben Strasser (Karlsruhe Institute of Technology) [`https://github.com/ben-strasser/flow-cutter-pace16`]

3rd prize, 75 €: Max Bannach, Sebastian Berndt, and Thorsten Ehlers (Luebeck University) [`https://github.com/maxbannach/Jdrasil`]

3 Competition track B: Feedback Vertex Set

In the (undirected) Feedback Vertex Set problem we are given an undirected graph G and want to compute a smallest vertex set S such that removing S from G results in a forest, that is, a graph without cycles. Feedback Vertex Set is NP-complete [12] and one of the most prominent problems in parameterized algorithmics. Most fixed-parameter algorithms use the parameter solution size $k = |S|$.

Virtually all fixed-parameter algorithms make use of the fact that vertices of degree at most two can be easily removed from the graph. After this initial removal, a range of different techniques were used in the fixed-parameter algorithms. The first constructive fixed-parameter algorithm branches on a shortest cycle in the resulting graph. This cycle has length at most $2k$ in a yes-instance, which results in an overall running time of $(2k)^k n^{\mathcal{O}(1)}$ [8]. By using

a randomized approach on the resulting graph, a running time of $4^k n^{\mathcal{O}(1)}$ can be obtained [2]. The first deterministic approaches to achieve running times of the form $2^{\mathcal{O}(k)} n^{\mathcal{O}(1)}$ use the iterative compression technique. It iteratively builds up the graph by adding one vertex at a time, and makes use of the fact that a size-k solution can be stored during this computation [6, 9]. Other fixed-parameter algorithms for this problem can be obtained by branching on a vertex of maximum degree or by LP-based techniques [11].

3.1 Challenge setup and participation

We collected 230 graphs, which were mostly from various application fields such as social networks, biological networks, road networks, or incidence graphs of CNF-SAT formulas. The graphs were selected so that there was a steady progression from easy to hard instances. The set of public training and hidden test instances is available in a GitHub repository [19].

To determine the winners, we counted the number of instances that could be solved within the given time limit. To avoid overemphasizing low-level improvements of the algorithms, we set the time limit to 30 minutes per instance. To identify programs that report non-optimal solutions we precomputed the optimal solutions for some instances using an ILP that was given at least 30 minutes on each instance. This ILP is based on cycle constraints. More precisely, we add constraints enforcing that for each cycle at least one vertex must be deleted by any solution. Since the number of constraints is usually exponential, they are added in a lazy fashion, that is, we compute a solution with only some initial constraints and check whether the solution is a feedback vertex set. If this is the case, then we have found an optimal solution, otherwise we add constraints for some of the remaining cycles and compute a new solution until a feedback vertex set is found.

Overall, 14 teams registered out of which seven eventually submitted a program. From those teams that submitted a program, three were from Germany, one from India, one from Japan, one from Poland, and one from Russia.

3.2 Results

In the following, we give for each team further details such as the number of solved instances, a brief algorithm description, the names of the participants, and a link to the code repository. All submissions apply a reduction rule that removes all vertices of degree at most two. The final ranking for track B is:

1st prize, 500 €: Yoichi Iwata (NII) and Kensuke Imanishi (University of Tokyo). This submission solved 84 out of 130 instances. The algorithm utilizes an LP-based branching [11] and an LP-based kernelization [10]. The program is written in Java. [`https://github.com/wata-orz/fvs`]

2nd prize, 300 €: Marcin Pilipczuk (University of Warsaw). This submission solved 66 out of 130 instances. The algorithm branches on a vertex of maximum degree. In addition, instances with small treewidth are solved by dynamic programming on tree decompositions and subcubic instances are solved by a polynomial-time algorithm that is based on a reduction to the graphic matroid parity problem. The program is written in C++. [`https://bitbucket.org/marcin_pilipczuk/fvs-pace-challenge`]

3rd prize, 200 €: Ruben Becker, Karl Bringmann, Dennis Gross, Erik Jan van Leeuwen, and Natalie Wirth (MPI Saarbrücken). This submission solved 50 out of 130 instances. The algorithm also branches on vertices of the highest degree. The search tree is pruned by computing upper and lower bounds. The program is written in C++. [`https://github.com/erikjanvl/FVS_MPI`]

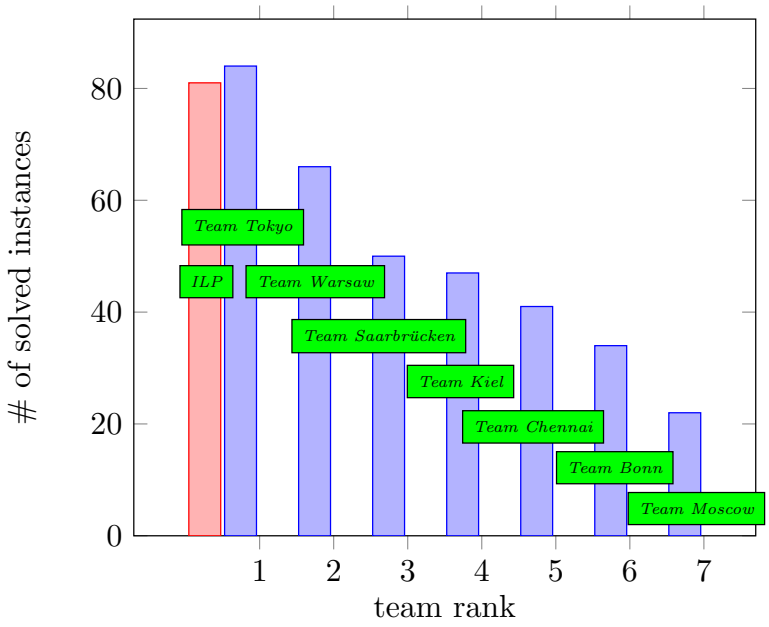


Figure 4 Results for track B. The figure shows, for each participating team, the number of instances for which an optimal solution was found within 30 minutes. The leftmost column corresponds to the Gurobi-based ILP used by the program committee.

4th place: Niklas Paulsen, Kevin Prohn, Malin Rau, and Lars Rohwedder (Kiel University). This submission solved 47 out of 130 instances. The algorithm is based on the combination of iterative compression with an improved branching strategy. Subcubic graphs are solved again by reduction graphic matroid parity. The program is written in C#. [`https://git.informatik.uni-kiel.de/npau/FFF`]

5th place: Shivam Garg (IIT Bombay), G. Philip, and Apoorva Tamaskar (Chennai Mathematical Institute). This submission solved 41 out of 130 instances. The algorithm branches on a shortest cycle. This program is written in Python. [`https://bitbucket.org/gphilip_bitbucket/pace-code`]

6th place: Fabian Brand, Simon Gehring, Florian Nelles, Kevin Wilkinghoff, and Xianghui Zhong (University of Bonn). This submission solved 34 out of 130 instances. The algorithm is based on iterative compression and also solves subcubic instances in polynomial time. The program is written in C++. Xianghui Zhong received a travel award of 780 € to be able to attend the award ceremony. [`https://github.com/s-gehring/feedback-vertex-set`]

7th place: Svyatoslav Feldsherov (Moscow State University). This submission solved 22 out of 130 instances. The algorithm uses the randomized approach with running time $4^k n^{\mathcal{O}(1)}$. An improvement is gained for the case where two vertices are connected by a multi-edge. In this case, the algorithm branches directly on these two vertices. The program is written in C++. [`https://github.com/feldsherov/pace2016`]

As a final remark, the ILP used by the program committee solved 81 out of 130 instances within the time limit. Thus, the best FPT approaches were competitive with this particular ILP formulation. Since alternative ILP formulations are possible, a more thorough comparison with further ILP-based approaches would be necessary to gain insight into the relative performance of FPT-based and ILP-based approaches for Feedback Vertex Set.

4 PACE organization

The PACE steering committee consists of the present authors, with Frances Rosamond as chair. The program committees for the two tracks in 2016 consisted of:

Track A:	Isolde Adler	University of Leeds
	Holger Dell (Chair)	Saarland University & Cluster of Excellence
	Thore Husfeldt	ITU Copenhagen & Lund University
	Lukas Larisch	University of Leeds
	Felix Salfelder	Goethe University Frankfurt
Track B:	Falk Hüffner	Industry
	Christian Komusiewicz	Friedrich-Schiller-University Jena

5 The future of PACE

As organizers, we consider the first iteration of PACE to be a huge success: we had great submissions building on existing and new theoretical ideas, which led to fast programs that performed well on the real-word inputs to which they were applied. The award ceremony at IPEC was very well attended, and many of the ALGO 2016 participants showed an interest in the competition.

To continue driving the transfer of algorithmic ideas from theory to practice, we intend PACE to become an annual event. The target problem(s) will change whenever relevant, but the same problem may be used for several consecutive years when there are indications that further developments are possible. Plans for the next iteration of PACE can be found on the challenge website [18].

We thank all the participants for their enthusiastic participation and look forward to many interesting iterations of the challenge in the future. We also thank all members of the community for their input in formulating the goals and setup of the challenge. We welcome anyone who is interested to add their name to the mailing list on the website [18] to receive PACE updates and join the discussion.

Acknowledgments. Prize money and travel grants were given through the generosity of Networks [17], an NWO Gravitation project of the University of Amsterdam, Eindhoven University of Technology, Leiden University, and the Center for Mathematics and Computer Science (CWI). The steering committee thanks the two program committees for their hard work in making PACE a success. In particular, we are grateful to Felix Salfelder for his help in preparing plots of the results.

References

1 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebra. Discr.*, 8:277–284, 1987. doi:10.1137/0608024.

2 Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized algorithms for the loop cutset problem. *J. Artif. Intell. Res. (JAIR)*, 12:219–234, 2000. doi:10.1613/jair.638.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

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

5 Krishnendu Chatterjee, Rasmus Ibsen-Jensen, and Andreas Pavlogiannis. Optimal reachability and a space-time tradeoff for distance queries in constant-treewidth graphs. In *Proc.*

24th ESA, volume 57 of *LIPIcs*, pages 28:1–28:17, 2016. doi:10.4230/LIPIcs.ESA.2016.28.

6 Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. *Theory Comput. Syst.*, 41(3):479–492, 2007. doi:10.1007/s00224-007-1345-z.

7 DIMACS graph coloring instances. URL: http://mat.gsia.cmu.edu/COLOR/instances.html.

8 Rodney G. Downey and Michael R. Fellows. Parameterized computational feasibility. In *Feasible Mathematics II*, pages 219–244. Birkhauser, 1994.

9 Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. Syst. Sci.*, 72(8):1386–1396, 2006. doi:10.1016/j.jcss.2006.02.001.

10 Yoichi Iwata. Linear-time kernelization for feedback vertex set. *CoRR*, abs/1608.01463, 2016. URL: http://arxiv.org/abs/1608.01463.

11 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.

12 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proc. of a Symp. on the Complexity of Computer Computations*, IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

13 Kalev Kask, Andrew Gelfand, Lars Otten, and Rina Dechter. Pushing the power of stochastic greedy ordering schemes for inference in graphical models. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011*. AAAI Press, 2011. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3771.

14 Petteri Kaski. Engineering motif search for large graphs, 2015. URL: https://simons.berkeley.edu/talks/petteri-kaski-2015-11-05.

15 Masashi Kiyomi, Yoshio Okamoto, and Yota Otachi. On the treewidth of toroidal grids. *Discrete Applied Mathematics*, 198:303–306, 2016. doi:10.1016/j.dam.2015.06.027.

16 List of all submissions for track A, 2016. URL: https://github.com/holgerdell/PACE-treewidth-testbed/blob/github/pace2016-submissions.yaml.

17 Networks project, 2016. URL: http://www.thenetworkcenter.nl.

18 Parameterized Algorithms and Computational Experiments website, 2015. URL: http://pacechallenge.wordpress.com.

19 Repository of all hidden and public inputs for track B on GitHub, 2016. URL: https://github.com/ckomus/PACE-fvs.

20 Repository of control flow graphs on GitHub, 2016. URL: https://github.com/freetdi/CFGs.git.

21 Repository of named graphs on GitHub, 2016. URL: https://github.com/freetdi/named-graphs.

22 Schulze rankings of heuristic treewidth implementations, 2016. URL: https://github.com/holgerdell/PACE-treewidth-testbed/blob/github/logs/2016-08-13.02-08-25/ranks-he-se.txt.

23 Markus Schulze. A new monotonic, clone-independent, reversal symmetric, and Condorcet-consistent single-winner election method. *Social Choice and Welfare*, 36(2):267–303, 2011. doi:10.1007/s00355-010-0475-4.

24 Mikkel Thorup. All structured programs have small tree-width and good register allocation. *Inf. Comput.*, 142(2):159–181, 1998. doi:10.1006/inco.1997.2697.

25 Treewidth testbed on GitHub, 2016. URL: https://github.com/holgerdell/PACE-treewidth-testbed.