PhD dissertation

University of Warsaw

Faculty of Mathematics, Informatics and Mechanics

Graph Width Parameters. Dependencies, Algorithms and Decompositions.

Author:
Wojciech Nadara

Supervisor:
Dr hab. Marcin Pilipczuk

February 13, 2023

Author's declaration:
I hereby declare that this dissertation is my own work.

February 13, 2023	____________________
Date	*Wojciech Nadara*

Supervisor's declaration:
This dissertation is ready to be reviewed.

February 13, 2023	____________________
Date	*dr hab. Marcin Pilipczuk*

Abstract

This dissertation will be based on results from structural graph theory, mainly revolving around graph width parameters, dependencies between them, their usages and finding certifying decompositions. Specific parameters that will be of most relevance to this dissertation are *treedepth, treewidth, pathwidth* and *mad* (short for *maximum average degree*).

In Chapter 3, based on the article "Improved bounds for the excluded-minor approximation of treedepth" [26], we show that there exists a constant C such that for every positive integers a, b and a graph G, if the treedepth of G is at least Cab, then the treewidth of G is at least a or G contains a subcubic (i.e., of maximum degree at most 3) tree of treedepth at least b as a subgraph. As a direct corollary, we obtain that every graph of treedepth $\Omega(k^3)$ is either of treewidth at least k, contains a subdivision of full binary tree of depth k, or contains a path of length 2^k. This improves the bound of $\Omega(k^5 \log^2 k)$ of Kawarabayashi and Rossman [SODA 2018]. We also show an application of our techniques for approximation algorithms of treedepth: given a graph G of treedepth k and treewidth t, one can in polynomial time compute a treedepth decomposition of G of width $\mathcal{O}(kt \log^{3/2} t)$. This improves upon a bound of $\mathcal{O}(kt^2 \log t)$ stemming from a tradeoff between known results. The main technical ingredient in our result is a proof that every tree of treedepth d contains a subcubic subtree of treedepth at least $d \cdot \log_3((1 + \sqrt{5})/2)$.

In Chapter 4, based on article "Efficient fully dynamic elimination forests with applications to detecting long paths and cycles" [18, 19], we show that every minimal graph of treedepth d has at most $d^{\mathcal{O}(d)}$ vertices, improving upon work of Dvorak et al. [31].

In Chapter 5, based on the article "Computing treedepth in polynomial space and linear fpt time" [62, 61], we propose an algorithm that given a graph G and an integer d, either finds a treedepth decompositon of G of depth at most d or concludes that no such decomposition exists; thus the algorithm decides whether the treedepth of G is at most d. The running time is $2^{\mathcal{O}(d^2)} \cdot n^{\mathcal{O}(1)}$ and the space usage is polynomial in n. Further, by allowing randomization, the time and space complexities can be improved to $2^{\mathcal{O}(d^2)} \cdot n$ and $d^{\mathcal{O}(1)} \cdot n$, respectively. This improves upon the algorithm of Reidl et al. [ICALP 2014], which also has time complexity $2^{\mathcal{O}(d^2)} \cdot n$, but uses space exponential in d.

In Chapter 6, based on the article "Approximating Pathwidth for Graphs of Small Treewidth" [41], we prove that every graph with large pathwidth has large treewidth or contains a subdivision of a large complete binary tree. Specifically, we show that every graph with pathwidth at least $th + 2$ has treewidth at least t or contains a subdivision of a complete binary tree of height $h + 1$. The bound $th + 2$ is best possible up to a multiplicative constant. This result was motivated by, and implies (with $c = 2$), the following conjecture of Kawarabayashi and Rossman (SODA'18): there exists a universal constant c such that every graph with pathwidth $\Omega(k^c)$ has treewidth at least k or contains a subdivision of a complete binary tree of height k. These structural insights allowed us to design a polynomial-time algorithm which, given a graph G with treewidth t, approximates the pathwidth of G to within a ratio of $O(t\sqrt{\log t})$, which is described in [41], but not in this thesis. This is the first algorithm to achieve an $f(t)$-approximation for some function f.

The maximum average degree $\mathrm{mad}(G)$ of a graph G is the maximum over all subgraphs of G, of the average degree of the subgraph. In Chapter 7, based on the article "Decreasing the maximum average degree by deleting an independent set or a d-degenerate subgraph" [63], we prove that for every G and positive integer k such that $\mathrm{mad}(G) \geq k$ there exists $S \subseteq V(G)$ such that $\mathrm{mad}(G - S) \leq \mathrm{mad}(G) - k$ and $G[S]$ is $(k - 1)$-degenerate. Moreover, such S can be computed in polynomial time. In particular, if G contains at least one edge then there exists an independent set I in G such that $\mathrm{mad}(G - I) \leq \mathrm{mad}(G) - 1$ and if G contains a cycle then there exists an induced forest F such that $\mathrm{mad}(G - F) \leq \mathrm{mad}(G) - 2$. As a side result, we also obtain a subexponential bound on the diameter of reconfiguration graphs of generalized colourings of graphs with bounded value of their mad.

2012 ACM Computing Classification System: Theory of computation → Design and analysis of algorithms → Graph algorithms analysis

Keywords: graph algorithms, graph width parameters, treedepth, treewidth, pathwidth, mad, maximum average degree, graph decompositions, approximation algorithms, parameterized algorithms

Streszczenie

Ta rozprawa jest bazowana na wynikach ze strukturalnej teorii grafów, dotyczących głównie parametrów szerokości grafów, zależności pomiędzy nimi, ich zastosowań oraz wyznaczania odpowiadających dekompozycji. Konkretne parametry najbardziej związane z tą pracą to *treedepth (głębokość drzewiasta), treewidth (szerokość drzewiasta), pathwidth (szerokość ścieżkowa)* i *mad* (skrót od *maximum average degree (największy średni stopień)*).

W rozdziale 3 bazowanym na pracy “Improved bounds for the excluded-minor approximation of treedepth” [26] dowodzimy istnienie stałej C takiej że dla każdych dodatnich liczb całkowitych a, b i grafu G, jeżeli treedepth grafu jest co najmniej Cab, wtedy albo treewidth grafu G jest co najmniej a lub G zawiera subkubiczne (tzn. o stopniach co najwyżej 3) drzewo o treedepthie co najmniej b jako podgraf. Jako bezpośredni wniosek, uzyskujemy że każdy graf o treedepthie $\Omega(k^3)$ albo ma treewidth co najmniej k, albo zawiera subdywizję pełnego drzewa binarnego wysokości k albo zawiera ścieżkę długości 2^k. To poprawia ograniczenie $\Omega(k^5 \log^2 k)$ udowodnione przez Kawarabayashi i Rossmana [SODA 2018]. Pokazujemy także zastosowanie naszych technik dla problemu aproksymacji treedepthu: dla danego grafu G o treedepthie k i treewidthie t, jesteśmy w stanie obliczyć w wielomianowym czasie dekompozycję treedepthową G o głębokości $\mathcal{O}(kt \log^{3/2} t)$. To polepsza najlepsze znane do tej pory ograniczenie $\mathcal{O}(kt^2 \log t)$ wynikające z kompromisu pomiędzy znanymi wynikami. Głównym technicznym składnikiem naszego wyniku jest dowód faktu, że każde drzewo o treedepthie d zawiera subkubiczne poddrzewo o treedepthie co najmniej $d \cdot \log_3((1 + \sqrt{5})/2)$.

W rozdziale 4 bazowanym na artykule “Efficient fully dynamic elimination forests with applications to detecting long paths and cycles” [18, 19] pokazujemy, że każdy minimalny graf o treedepthie d ma co najwyżej $d^{\mathcal{O}(d)}$ wierzchołków, polepszając wynik z pracy Dvorak et al. [31].

W rozdziale 5 bazowanym na artykule “Computing treedepth in polynomial space and linear fpt time” [62, 61] proponujemy algorytm, który dla danego grafu G i liczby całkowitej d albo znajduje dekompozycję treedepthową G o szerokości co najwyżej d albo stwierdza, że taka dekompozycja nie istnieje; zatem taki algorytm w szczególności stwierdza, czy treedepth G wynosi co najwyżej d. Czas działania tego algorytmu wynosi $2^{\mathcal{O}(d^2)} \cdot n^{\mathcal{O}(1)}$, a używana pamięć jest wielomianowa względem n. Co więcej, dzięki użyciu randomizacji, złożoności czasowe i pamięciowe mogą być polepszone do odpowiednio $2^{\mathcal{O}(d^2)} \cdot n$ i $d^{\mathcal{O}(1)} \cdot n$. To polepsza algorytm Reidl et al. [ICALP 2014], którego złożoność również wynosi $2^{\mathcal{O}(d^2)} \cdot n$, ale używa on pamięci wykładnicznej względem d.

W rozdziale 6 bazowanym na artykule “Approximating Pathwidth for Graphs of Small Treewidth” [41] dowodzimy, że każdy graf o dużym pathwidthie posiada duży treewidth albo subdywizję dużego kompletnego drzewa binarnego. Konkretniej, pokazujemy, że każdy graf o pathwidthie co najmniej $th + 2$ ma treewidth co najmniej t albo zawiera subdywizję kompletnego drzewa binarnego wysokości $h + 1$. Ograniczenie $th + 2$ jest najlepsze możliwe z dokładnością do multiplikatywnej stałej. Ten wynik był motywowany i implikuje (z $c = 2$) następującą hipotezę postawioną przez Kawarabayashi i Rossmana (SODA’18): istnieje uniwersalna stała c taka że każdy graf o pathwidthie $\Omega(k^c)$ ma treewidth co najmniej k albo zawiera subdywizję kompletnego drzewa binarnego o wysokości k. Wspomniane strukturalne spostrzeżenia pozwoliły nam zaprojektować algorytm działający w wielomianowym czasie, który dla grafu G o treewidthie t, aproksymuje jego pathwidth w stosunku $O(t\sqrt{\log t})$, który jest opisany w [41], ale nie w tej rozprawie. To pierwszy algorytm w wielomianowym czasie osiągający $f(t)$ aproksymację pathwidthu dla pewnej funkcji f.

Największy średni stopień $\mathrm{mad}(G)$ grafu G to największa wartość średniego stopnia po wszystkich podgrafach grafu G. W rozdziale 7 bazowanym na artykule “Decreasing the maximum average degree by deleting an independent set or a d-degenerate subgraph” [63] dowodzimy, że dla każdego grafu G i dodatniej liczby całkowitej k, istnieje zbiór $S \subseteq V(G)$ taki że $\mathrm{mad}(G - S) \leq \mathrm{mad}(G) - k$ i $G[S]$ jest $(k-1)$-zdegenerowany. Co więcej, taki S może być wyznaczony w wielomianowym czasie. W szczególności, jeżeli G zawiera co najmniej jedną krawędź, to istnieje taki zbiór niezależny I w G że $\mathrm{mad}(G - I) \leq \mathrm{mad}(G) - 1$ oraz jeżeli G zawiera jakiś cykl, to wtedy istnieje indukowany las F taki że $\mathrm{mad}(G - F) \leq \mathrm{mad}(G) - 2$. Jako poboczny rezultat otrzymujemy także podwykładnicze ograniczenie na średnicę grafów rekonfiguracji uogólnionych kolorowań grafów o ograniczonych wartościach mad.

Tytuł pracy w języku polskim: Parametry szerokości grafów. Zależności, algorytmy i dekompozycje.

Słowa kluczowe: algorytmy grafowe, parametry szerokości grafu, głębokość drzewiasta, szerokość drzewiasta, szerokość ścieżkowa, mad, największy średni stopień, dekompozycje grafowe, algorytmy aproksymacyjne, algorytmy parametryzowane

Acknowledgements

First of all, I would like to thank my supervisor — Marcin Pilipczuk. He offered me a lot of help and guidance both from the research side as well as the organizational and bureaucracy side while acting as my supervisor throughout 7 years of both my PhD and Master studies.

I would also like to thank his brother — Michał Pilipczuk — who despite not being my formal supervisor, also offered a lot of help in similar areas. They are excellent researchers and I definitely learned a lot from their courses, collaborating with them and from various informal talks. I do not imagine my PhD being half as fruitful, if not for them.

I am also grateful to whole University of Warsaw algorithmics group for creating a friendly and productive working environment, as well as to all my other co-authors who I had a pleasure collaborating with on various workshops.

I believe that being an active competitive programmer played a key role in my research development. It helped me direct my interests into algorithmics, motivated me to learn a lot and taught me many useful ideas and techniques that now could act as a solid basis in many various areas of research. I truly believe that despite research setting and competitive programming setting are vastly different, there is no better background for starting algorithmic research than being an experienced competitive programmer. That is why I would like to thank all people that made competitive programming possible and fun for me.

That includes, but is not limited to:

- Many friends who acted as both my teammates or closest rivals that I learned a lot from. Most importantly Marcin Smulewicz, Marek Sokołowski and Grzegorz Prusak that were my ICPC teammates. In particular, both Marcin and Marek were my PhD colleagues that I collaborated a lot with.
- A bit older friends who acted as our ICPC coaches and later on as more experienced PhD colleagues offering various advice — Tomasz Kociumaka and Adam Karczmarz.
- Senior professors responsible for creating Polish competitive programming community — Jan Madey and Krzysztof Diks.
- Various people building and being a part of online global competitive programming communities.

Funding My research was supported by funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programmes Grant Agreement No. 714704 — CUTACOMBS and No. 948057 — BOBR.

Contents

Chapter 1

Introduction

Graphs are fundamental and important objects in computer science. They allow us to model variety of networks, for example, road networks, social networks, eletrical networks or brain connections. Given such a general type of objects, one may pose a large number of problems regarding graphs. Many of these have real life applications, for example, navigation systems need to answer queries of how to get from point A to point B in the fastest possible way, which can be formulated as variants of classical shortest path problem. Other examples include problems like maximum flow, where we ask what is the biggest amount of water we can send through a pipes network or traveling salesman problem, which can be described as asking about the shortest route courier needs to have to deliver all packages.

Some of graph problems are solvable in polynomial time (for example, the shortest path problem or the maximum flow problem), while some of them are not (under standard complexity theory assumptions) and cannot be solved fast enough on large general graphs (for example, the traveling salesman problem). In order to broaden the horizon of feasibility one may ask whether some of these problems are solvable on graphs that are *large*, but *simple* in some sense. Hence, the structural graph theory has risen — an area which distinguishes a number of properties that graphs could have, that allows us to understand the structure of various classes of graphs, which further on could be utilized in a design of efficient algorithms.

One of the most famous notions in structural graph theory is the notion of *treewidth*. Intuitively speaking, it measures how much a graph is similar to a tree (the lower treewidth, the more similar the graph is to a tree). It is both very useful in a design of efficient algorithms, as well as plays a key role in the celebrated graph structure theorem by Robertson and Seymour [78]. The power of treewidth notion is also shown by the famous Courcelle's Theorem [23] stating that every graph property definable in the monadic second-order logic of graphs can be decided in linear time on graphs of bounded treewidth.

Treewidth notion is just an example of a graph width parameter and has a lot of siblings. Other known graph width parameters include notions like pathwidth, treedepth, rankwidth, branchwidth, cliquewidth, twinwidth, mimwidth. Knowing that a graph has small width of a certain type often turns out to be highly useful in designing efficient algorithm for various problems that are not efficiently solvable otherwise. Boundedness of width often enables designing a problem-specific dynamic programming algorithm or applying some meta-theorem to solve a certain problem. In majority of cases, the resulting algorithm has time complexity of type $f(w) \cdot n^{\mathcal{O}(1)}$, where w is the width of a graph and n is its number of vertices. Such algorithms are called FPT (fixed parameter tractable) algorithms, parameterized by the width.

In this thesis we will be the most interested in treewidth, treedepth and pathwidth parameters.

1.1 Treedepth results

For an undirected graph G, the *treedepth* of G is the minimum height of a rooted forest whose ancestor-descendant closure contains G as a subgraph. Together with more widely known related width notions, it plays a major role in structural graph theory, in particular in the study of general sparse graph classes [66, 68, 65]. Compared to *treewidth*, treedepth measures the depth of a tree-like decomposition of a graph, instead of width. The two parameters are related: if by $\mathrm{td}(G)$ and $\mathrm{tw}(G)$ we denote the treedepth and the treewidth of an n-vertex graph G, then it always holds that $\mathrm{tw}(G) \leq \mathrm{td}(G) \leq \mathrm{tw}(G) \cdot \log_2 n$. However, the two notions are qualitatively different: for instance, a path on t vertices has treewidth 1 and treedepth $\Theta(\log t)$.

Treedepth appears prominently in structural graph theory, especially in the theory of sparse graphs of Nešetřil and Ossona de Mendez. There, it serves as a basic building block for fundamental decompositions of sparse graphs — *low treedepth colorings* — which can be used for multiple algorithmic purposes, including designing algorithms for Subgraph Isomorphism and model-checking First-Order logic. See [68, Chapters 6 and 7] for an introduction and [39, 42, 67, 66, 69, 70, 72, 71] for examples of applications.

An important property of treedepth is that it admits a number of equivalent definitions. Following the definition of treedepth above, a *treedepth decomposition* of a graph G consists of a rooted forest F and an injective mapping $f : V(G) \to V(F)$ such that for every $uv \in E(G)$ the vertices $f(u)$ and $f(v)$ are in ancestor-descendant relation in F. The *width* (or the *depth*) of a treedepth decomposition (F, f) is the height of F (the number of vertices on the longest leaf-to-root path in F) and the treedepth of G is the minimum possible height of a treedepth decomposition of G. A *centered coloring* of a graph G is an assignment $\alpha : V(G) \to \mathbb{Z}$ such that for every connected subgraph H of G, α has a *center* in H: a vertex $v \in V(H)$ of unique color, i.e., $\alpha(v) \neq \alpha(w)$ for every $w \in V(H) \setminus \{v\}$. A *vertex ranking* of a graph G is an assignment $\alpha : V(G) \to \mathbb{Z}$ such that in every connected subgraph H of G there is a unique vertex of *maximum* rank (value $\alpha(v)$). Clearly, each vertex ranking is a centered coloring. It turns out that the minimum number of colors (minimum size of the image of α) needed for a centered coloring and for a vertex ranking are equal and equal to the treedepth of a graph [65].

In this thesis, among other topics, we are interested in using treedepth as a parameter for the design of fixed-parameter (FPT) algorithms. Clearly, every dynamic programming algorithm working on a tree decomposition of a graph can be adjusted to work also on an elimination forest, just because an elimination forest of depth d can be easily transformed into a tree decomposition of width $d - 1$. However, it has been observed in [38, 73, 47, 64, 71] that for multiple basic problems, one can design FPT algorithms working on elimination forests of bounded depth that have polynomial space complexity without sacrificing on the time complexity. These include the following: (In all results below, n is the vertex count and d is the depth of the given elimination forest.)

- A $3^d \cdot n^{\mathcal{O}(1)}$-time $\mathcal{O}(d + \log n)$-space algorithm for 3-Coloring [73].
- A $2^d \cdot n^{\mathcal{O}(1)}$-time $n^{\mathcal{O}(1)}$-space algorithm for counting perfect matchings [38].
- A $3^d \cdot n^{\mathcal{O}(1)}$-time $n^{\mathcal{O}(1)}$-space algorithm for Dominating Set [38, 73].
- A $d^{|V(H)|} \cdot n^{\mathcal{O}(1)}$-time $n^{\mathcal{O}(1)}$-space algorithm for Subgraph Isomorphism [71].
  (Here, H is the sought pattern graph.)
- A $3^d \cdot n^{\mathcal{O}(1)}$-time $n^{\mathcal{O}(1)}$-space algorithm for Connected Vertex Cover [47].
- A $5^d \cdot n^{\mathcal{O}(1)}$-time $n^{\mathcal{O}(1)}$-space algorithm for Hamiltonian Cycle [64].

We note that the approach used in [47, 64] to obtain the last two results applies also to several other problems with connectivity constraints. However, as these algorithms are based on the Cut&Count technique [25], they are randomized and no derandomization preserving the polynomial space complexity is known. An in-depth complexity-theoretical analysis of the time-space tradeoffs for algorithms working on different graph decompositions can be found in [73].

In the algorithms mentioned above one assumes that the input graph is supplied with an elimination depth of depth at most d. Therefore, it is imperative to design algorithms that given the graph alone, computes, possibly approximately, such an elimination forest. Compared to the setting of treewidth and tree decompositions, where multiple approaches have been proposed over the years (see e.g. [7, 55] for an overview), so far there was only a handful of algorithms to compute the treedepth exactly or approximately.

- It is well-known (see e.g. [68, Section 6.2]) that just running depth-first search and outputing the forest of recursive calls gives an elimination forest of depth at most $2^{\mathrm{td}(G)}$. So this gives a very simple linear-time approximation algorithm, but with the approximation factor exponential in the optimum.
- Reidl et al. [76] gave an exact FPT algorithm that in time $2^{\mathcal{O}(d \cdot \mathrm{tw}(G))} \cdot n$ either constructs an elimination forest of depth at most d, or concludes that the treedepth is larger than d. Recall here that $\mathrm{tw}(G) \leq$

td(G).

- Through a tradeoff trick, which we will describe below, one can get a polynomial time approximation algorithm computing a treedepth decomposition of G of width $\mathcal{O}(\mathrm{td}(G) \cdot \mathrm{tw}(G)^2 \log \mathrm{tw}(G))$.

In particular, obtaining a constant-factor approximation for treedepth running in time $2^{\mathcal{O}(\mathrm{td}(G))} \cdot n^{\mathcal{O}(1)}$ is a well-known open problem, see e.g. [26]. We note that implementation of practical FPT algorithms for computing treedepth was the topic of the 2020 Parameterized Algorithms and Computational Experiments (PACE) Challenge [56].

For approximation algorithms, the following folklore lemma (presented with full proof in [50]) is very useful.

Lemma 1.1.1. *Given a graph G and a tree decomposition (T, β) of G of maximum bag size w, one can in polynomial time compute a treedepth decomposition of G of width at most $w \cdot \mathrm{td}(T)$.*

Using Lemma 1.1.1, one can obtain an approximation algorithm for treedepth with a cheap tradeoff trick.[1]

Lemma 1.1.2. *Given a graph G, one can in polynomial time compute a treedepth decomposition of G of width $\mathcal{O}(\mathrm{td}(G) \cdot \mathrm{tw}(G)^2 \log \mathrm{tw}(G))$.*

Proof. Let $n = |V(G)|$. Using the polynomial-time approximation algorithm for treewidth [36], compute a tree decomposition (T, β) of G of width $t = \mathcal{O}(\mathrm{tw}(G)\sqrt{\log \mathrm{tw}(G)})$ and $\mathcal{O}(n)$ bags. For every integer $1 \leq k \leq (\log n)/t$, use the algorithm of [76] to check in polynomial time if the treedepth of G is at most k. Note that if this is the case, the algorithm finds an optimal treedepth decomposition and we can conclude. Otherwise, we have $\log n \leq \mathrm{td}(G) \cdot t$ and we can apply Lemma 1.1.1 to G and (T, β), obtaining a treedepth decomposition of G whose width based on folklore inequality $\mathrm{td}(T) \leq 1 + \log_2 |V(T)|$, which is proven in Lemma 3.3.3, can be estimated as

$$\mathcal{O}(t \cdot \mathrm{td}(T)) \leq \mathcal{O}(t \log n) \leq \mathcal{O}(\mathrm{td}(G) \cdot t^2) \leq \mathcal{O}(\mathrm{td}(G) \cdot \mathrm{tw}(G)^2 \log \mathrm{tw}(G)).$$

□

Lemma 1.1.2 is the only polynomial approximation algorithm for treedepth running in polynomial time we were aware of. In this thesis we improve state of the art algorithms in both polynomial time category (Theorem 1.1.7 in Chapter 3 based on [26]) as well as in the exact computation category (Theorem 1.1.10 in Chapter 5 based on [61]).

A related topic to exact and approximation algorithms computing minimum-width treedepth decomposition is the study of obstructions to small treedepth. For any integer d, the class of graphs with treedepth at most d is clearly minor-closed. As such, based on famous Robertson-Seymour theorem [78], it can be characterized by forbidding a finite set of minors. However, this general theorem does not provide useful insight on how these forbidden minors look like. Studying obstructions for small treedepth is important from a viewpoint of getting a fundamental understanding of what makes treedepth high and can possibly be used in algorithm design. Dvořák, Giannopoulou, and Thilikos [31] proved that every minimal graph of treedepth d has the number of vertices at most double-exponential in d and gave a construction of an obstruction with 2^d vertices. They also hypothesized that in fact, every minimal obstruction for treedepth d has at most 2^d vertices. In Chapter 4 based on [18, 19], we get closer to this conjecture by showing an improved upper bound of $d^{\mathcal{O}(d)}$ in Theorem 1.1.9.

As the set of minors that are obstructions for having treedepth at most d is not yet well understood, one may ask what happens if we forbid some simpler set of minors. Having a big treedepth is functionally equivalent to having a long path, or in other words, graphs with big treedepth have long paths and graphs with long paths have big treedepth. That is formalized within the following fact:

Fact 1. Let $p(G)$ be the length of the longest path in G (counting vertices). Then $2^{\mathrm{td}(G)-1} \leq p(G) \leq \mathrm{td}(G)$.

[1]This trick has been observed and communicated to us by Michał Pilipczuk. We thank Michał for allowing us to include it here.

This fact was actually the basis for the first approximation algorithm for treedepth that we have mentioned. However, the exponential gap in this statement is unavoidable, as evidenced by examples of paths (first inequality) and complete graphs (second inequality). That motivates the question whether we can get a polynomial gap by forbidding a richer set of minors.

Recently, Kawarabayashi and Rossman showed such polynomial excluded-minor theorem for treedepth.

Theorem 1.1.3 ([50])**.** *There exists a universal constant C such that for every integer k every graph of treedepth at least $Ck^5 \log^2 k$ is either of treewidth at least k, contains a subdivision of a full binary tree of depth k as a subgraph, or contains a path of length 2^k.*

Note that each of these three outcomes alone implies that the treedepth is at least k.

In Chapter 3, which is based on [26], in Theorem 1.1.4 we are able to improve the bound of $\Omega(k^5 \log^2 k)$ to $\Omega(k^3)$.

The results of [31, 18, 19, 50] and [26] are tightly linked with each other and we expect that a finer understanding of treedepth obstructions is necessary to provide more efficient algorithms computing or approximating the treedepth of a graph. In particular, the improved upper bound on the size of minimal obstructions plays a key role in the improved exact algorithm for treedepth that we present in Theorem 1.1.10.

Detailed Chapter 3 results — improved polynomial excluded-minor treedepth theorem and approximation algorithm. To recall, Chapter 3 is based on [26]. Our main graph-theoretical result is the following statement, improving upon the work of Kawarabayashi and Rossman [50].

Theorem 1.1.4. *Let G be a graph of treewidth $\mathrm{tw}(G)$ and treedepth $\mathrm{td}(G)$. Then there exists a subcubic tree H that is a subgraph of G and is of treedepth at least*

$$\frac{\mathrm{td}(G)}{\mathrm{tw}(G)+1} \cdot \frac{\log((1+\sqrt{5})/2)}{\log(3)}.$$

In other words, Theorem 1.1.4 states that there exists a constant $C = \frac{\log(3)}{\log((1+\sqrt{5})/2)}$ such that for every graph G and positive integers a, b, if the treedepth of G is at least Cab, then the treewidth of G is at least a or G contains a subcubic tree of treedepth b. Since every tree of treedepth d contains either a simple path of length $2^{\Omega(\sqrt{d})}$ or a subdivision of a full binary tree of depth $\Omega(\sqrt{d})$ [50], we have the following corollary.

Corollary 1.1.5. *Let G be a graph of treewidth $\mathrm{tw}(G)$ and treedepth $\mathrm{td}(G)$. Then for some*

$$h = \Omega\left(\sqrt{\mathrm{td}(G)/(\mathrm{tw}(G)+1)}\right)$$

G contains either a simple path of length 2^h or a subdivision of a full binary tree of depth h.

Consequently, there exists an absolute constant C such that for every integer $k \geq 1$ and a graph G of treedepth at least Ck^3, either

- *G has treewidth at least k,*
- *G contains a subdivision of a full binary tree of depth k as a subgraph, or*
- *G contains a path of length 2^k.*

If treedepth of a graph G is at least Ck^3 and its treewidth is less than k, then $\left(\sqrt{\mathrm{td}(G)/(\mathrm{tw}(G)+1)}\right) \geq \sqrt{C}k$ and that is why second part of that Corollary follows from the first part. In other words, Corollary 1.1.5 improves the bound $k^5 \log^2 k$ of Kawarabayashi and Rossman [50] to k^3. We remark here that there are subcubic trees of treedepth $\Omega(h^2)$ that contain neither a path of length 2^h nor a subdivision of a full binary tree of depth h,[2] and thus the quadratic loss between the statements of Theorem 1.1.4 and Corollary 1.1.5 is necessary.

Inside the proof of Theorem 1.1.4 we make use of the following lemma that may be of independent interest. This lemma is the main technical improvement upon the work of Kawarabayashi and Rossman [50].

Lemma 1.1.6. *Every tree of treedepth d contains a subcubic subtree of treedepth at least $\frac{\log((1+\sqrt{5})/2)}{\log(3)}d$. Furthermore, such a subtree can be found in polynomial time.*

Lemma 1.1.6, developed to prove Theorem 1.1.4, have some implications on the approximability of treedepth. We combine it with the machinery of Kawarabayashi and Rossman [50] to improve upon Lemma 1.1.2 as follows.

Theorem 1.1.7. *Given a graph G, one can in polynomial time compute a treedepth decomposition of G of width $\mathcal{O}(\mathrm{td}(G) \cdot \mathrm{tw}(G) \log^{3/2} \mathrm{tw}(G))$.*

The result of Kawarabayashi and Rossman [50] has been also an important ingredient in the study of *linear colorings* [57]. A coloring $\alpha : V(G) \to \mathbb{Z}$ of a graph G is a *linear coloring* if for every (not necessarily induced) path P in G there exists a vertex $v \in V(P)$ of unique color $\alpha(v)$ on P. Clearly, each centered coloring is a linear coloring, but the minimum number of colors needed for a linear coloring can be much smaller than the treedepth of a graph. Kun et al. [57] provided a polynomial relation between the treedepth and the minimum number of colors in a linear coloring; by replacing their usage of [50] by our result (and using an improved bound for the excluded grid theorem [21]) we obtain an improved bound.

Theorem 1.1.8. *There exists a polynomial p such that for every integer k and graph G, if the treedepth of G is at least $k^{19}p(\log k)$, then every linear coloring of G requires at least k colors.*

The previous bound of [57] is $k^{190}p(\log k)$.

Detailed Chapter 4 results — obstructions for small treedepth. Chapter 4 is based on [18, 19]. The main result of it is the following theorem.

Theorem 1.1.9. *If G is a minimal obstruction for treedepth d, then the vertex count of G is at most*

$$(d+1) \cdot \frac{\left((d+1)((d+1)^2+1)\right)^{d+1} - 1}{(d+1)((d+1)^2+1) - 1} \in d^{\mathcal{O}(d)}.$$

It improves upon the work of Dvorak et al. [31], where doubly exponential bound was proven. Let us point out that this improvement will turn out to be crucial in designing the algorithm from Chapter 5.

Detailed Chapter 5 results — exact decomposition in polynomial space. Chapter 5 is based on [61]. Coming back to the exact computation of treedepth, the exact algorithm of Reidl et al. [76] uses not only exponential time (in the treedepth), but also exponential space. This would make it a space bottleneck when applied in combination with any of the polynomial-space algorithms developed in [38, 73, 47, 64, 71]. One can make a tradeoff between this algorithm and the trivial $\mathcal{O}(n^{\mathrm{td}(G)})$ algorithm computing treedepth directly from the definition to get polynomial space. Precisely, if $\mathrm{td}(G) \leq \sqrt{\log(n)}$ the first algorithm uses polynomial space and if $\mathrm{td}(G) \geq \sqrt{\log(n)}$ the second runs in $2^{\mathcal{O}(\mathrm{td}(G)^3)}$ time. This tradeoff leads to significant deterioration of the running time. In this thesis we bridge those issues by proving the following result.

Theorem 1.1.10. *There is an algorithm that given an n-vertex graph G and an integer d, either constructs an elimination forest of G of depth at most d, or concludes that the treedepth of G is larger than d. The algorithm runs in $2^{\mathcal{O}(d^2)} \cdot n^{\mathcal{O}(1)}$ time and uses $n^{\mathcal{O}(1)}$ space.*

The space and time complexities can be improved to $d^{\mathcal{O}(1)} \cdot n$ and expected $2^{\mathcal{O}(d^2)} \cdot n$, respectively, at the cost of allowing randomization: the algorithm may return a false negative with probability at most $\frac{1}{c \cdot n^c}$, where c is any constant fixed a priori; there are no false positives.

[2]It is straightforward to deduce such an example from the proof of [50]. We provide such an example in Section 3.5.

Thus, the randomized variant of the algorithm of Theorem 1.1.10 has the same time complexity as the algorithm of Reidl et al. [76], but uses polynomial space. However, the algorithm of Reidl et al. [76] is deterministic, contrary to ours. Note that apart from possible false negatives, the bound on the running time is only in expectation and not worst-case (in other words, our algorithm is both Monte Carlo and Las Vegas). However, one can turn this into a worst-case bound at the cost of increasing the probability of false negatives to 1/2 by forcefully terminating the execution if the algorithm runs for twice as long as expected.

Simultaneously achieving time complexity linear in n and polynomial space complexity is a property that is desired from an algorithm for computing the treedepth of a graph. While many of the polynomial-space FPT algorithms working on elimination forests do not have time complexity linear in n due to the usage of various algebraic techniques, the simplest ones that exploit only recursion — like the ones for 3-Coloring or Independent Set considered in [73] — can be easily implemented to run in time $2^{\mathcal{O}(d)} \cdot n$ and space $d^{\mathcal{O}(1)} \cdot n$. Thus, the randomized variant of the algorithm of Theorem 1.1.10 would neither be a bottleneck from the point of view of space complexity nor from the point of view of the dependency of the running time on n. Admittedly, the parametric factor in the runtime of our algorithm is $2^{\mathcal{O}(d^2)}$, as compared to $2^{\mathcal{O}(d)}$ in most of the aforementioned polynomial-space FPT algorithms working on elimination forests; this brings us back to the open problem about constant-factor approximation for treedepth running in time $2^{\mathcal{O}(\mathrm{td}(G))} \cdot n^{\mathcal{O}(1)}$ raised in [26].

Let us briefly discuss the techniques behind the proof of Theorem 1.1.10. The algorithm of Reidl et al. [76] starts by approximating the treewidth of the graph (which is upper bounded by the treedepth) and tries to construct an elimination forest of depth at most d by bottom-up dynamic programming on the obtained tree decomposition. By applying the iterative compression technique, we may instead assume that we are supplied with an elimination forest of depth at most $d + 1$, and the task is to construct one of depth at most d.

Applying now the approach of Reidl et al. [76] directly (that is, after a suitable adjustment from the setting of tree decompositions to the setting of elimination forests) would not give an algorithm with polynomial space complexity. The reason is that their dynamic programming procedure is quite involved and in particular keeps track of certain disjointness conditions; this is a feature that is notoriously difficult to achieve using only polynomial space. Therefore, we resort to the technique of *inclusion-exclusion branching*, used in previous polynomial-space algorithms working on elimination forests; see [38, 73] for basic applications of this approach. In a nutshell, the idea is to count more general objects where the disjointness contraints are relaxed, and to use inclusion-exclusion at each step of the computation to make sure that objects not satisfying the constraints eventually cancel out. We note that while the application of inclusion-exclusion branching was rather simple in [38, 73], in our case it poses a considerable technical challenge. In particular, along the way we do not count single values, but rather polynomials with one formal variable that keeps track of how much the disjointness constraints are violated. In the exposition layer, our application of inclusion-exclusion branching mostly follows the algorithm for Dominating Set of Pilipczuk and Wrochna [73].

In this way, we can count the number of elimination forests[3] of depth at most d in time $2^{\mathcal{O}(d^2)} \cdot n^{\mathcal{O}(1)}$ and using polynomial space. So in particular, we can decide whether there exists at least one such elimination forest. Such a decision algorithm can be quite easily turned into a construction algorithm using self-reducibility of the problem. This establishes the first part of Theorem 1.1.10.

As for the second part — the randomized linear-time FPT algorithm using polynomial space — there are several obstacles that need to be overcome. First, there is a multiplicative factor n in the running time coming from the iterative compression scheme. We mitigate this issue by replacing iterative compression with the recursive contraction scheme used by Bodlaender in his linear-time FPT algorithm to compute the treewidth of a graph [6]. Second, when using self-reducibility, we may apply the decision procedure n times, each taking at least linear time. This is replaced by an approach based on color coding, whose correctness relies on the fact that in a connected graph of treedepth at most d there are at most $d^{\mathcal{O}(d)}$ different feasible candidates for the root of an optimum-depth elimination tree [18, 19]. Finally, in the counting procedure we may operate on numbers of bitsize as large as polynomial in n. This is resolved by hashing them modulo a random prime of magnitude $\Theta(\log n)$, so that we may assume that arithmetic operations take unit time.

[3]Formally, we count only elimination forests satisfying some basic connectivity property, which we call *sensibility*.

We remark that it is relatively rare that a polynomial-space algorithm based on algebraic techniques can be also implemented so that it runs in time linear in the input size. Therefore, we find it interesting and somewhat surprising that this can be achieved for the problem of computing the treedepth of a graph, which combinatorially is rather involved.

1.2 Pathwidth results

In their already mentioned paper [50], Kawarabayashi and Rossman, apart from asking whether their bound of $k^5 \log^2 k$ can be improved, also asked whether analogous result can be proven for pathwidth. More specifically, they stated the following conjecture:

Conjecture 1.2.1. *There is an absolute constant c such that every graph with pathwidth $\Omega(k^c)$ has treewidth at least k, or contains a subdivision of a complete binary tree of height k.*

We prove that conjecture for the best possible value of c, which equals 2, by proving the following theorem.

Theorem 1.2.2. *Every graph with treewidth $t-1$ has pathwidth at most $th+1$ or contains a subdivision of a complete binary tree of height $h+1$.*

The bound $th+1$ is best possible up to a multiplicative constant (see Section 6.3).

We remark that Wood [83] also conjectured a statement of this type, with a bound of the form $f(t) \cdot h$ on the pathwidth for some function f (see also [59, Lemma 6] and [48, Conjecture 6.4.1]). Both Theorem 1.2.2 and the treedepth results [51, 26] are a continuation of a line of research on excluded minor characterizations of graphs with small values of their corresponding width parameters (treewidth/pathwidth/treedepth), which was started by the seminal Grid Minor Theorem [77] and its improved polynomial versions [17, 22].

Since the complete binary tree of height h has pathwidth $\lceil h/2 \rceil$ [81], any subdivision of it (as a subgraph) can be used to certify that the pathwidth of a given graph is large. The following key concept provides a stronger certificate of large pathwidth, more suitable for our purposes. Let $(\mathcal{T}_h)_{h=0}^{\infty}$ be a sequence of classes of graphs defined inductively as follows: $\mathcal{T}_0$ is the class of all connected graphs, and $\mathcal{T}_{h+1}$ is the class of connected graphs G that contain three pairwise disjoint sets of vertices V_1, V_2, and V_3 such that $G[V_1], G[V_2], G[V_3] \in \mathcal{T}_h$ and any two of V_1, V_2, and V_3 can be connected in G by a path avoiding the third one. Every graph in $\mathcal{T}_h$ has the following properties:

- it has pathwidth at least h (see Lemma 6.1.1), and
- it contains a subdivision of a complete binary tree of height h (see Lemma 6.1.2).

Theorem 1.2.2 has a short and simple proof (see Section 6.2). It proceeds by showing that every connected graph with treewidth $t-1$ has pathwidth at most $th+1$ or belongs to $\mathcal{T}_{h+1}$. The stronger assertion allows us to apply induction on h. Unfortunately, this proof is not algorithmic.

However, Theorem 1.2.2 provides a key insight for the algorithm that approximates pathwidth in graphs of small treewidth in polynomial time.

Theorem 1.2.3. *For every connected graph G with treewidth at most $t-1$, there is an integer $h \geq 0$ such that $G \in \mathcal{T}_h$ and G has pathwidth at most $th+1$. Moreover, there is a polynomial-time algorithm to compute such an integer h, a path decomposition of G of width at most $th+1$, and a subdivision of a complete binary tree of height h in G given a tree decomposition of G of width at most $t-1$.*

Since every graph in $\mathcal{T}_h$ has pathwidth at least h, combining Theorem 1.2.3 (applied to every connected component of the input graph) with the aforementioned approximation algorithm for treewidth of Feige et al. [36], one can obtain the following approximation algorithm for pathwidth.

Corollary 1.2.4. *There is a polynomial-time algorithm which, given a graph G of treewidth t and pathwidth p, computes a path decomposition of G of width $O(t\sqrt{\log t} \cdot p)$. Moreover, if a tree decomposition of G of width t' is also given in the input, the resulting path decomposition has width at most $(t'+1)p+1$.*

However, Theorem 1.2.3 is significantly more complicated and was omitted from this dissertation.

1.3 Maximum average degree results

The maximum average degree (abbreviated as mad) of a graph is a heavily studied notion. It has to be admitted that it is less of a "width notion" as opposed to treedepth, treewidth and pathwidth that were discussed earlier and does not provide decompositions that could be used in FPT algorithms, yet it is still an interesting graph parameter to research that leads to variety of structural results about partitioning the vertex set of a graph into subsets fulfilling some specific conditions. Multiple results show that a lower or upper bound on mad implies the existence of such partitions, e.g., [10, 20, 24, 29, 54]. Another class of results considers edge partitions, including [3, 12, 33, 44, 46, 53]. In these directions, particular attention has been paid to planar graphs, where, due to the inequality $(\mathrm{mad}(G) - 2)(g(G) - 2) < 4$, an upper bound on mad can be inferred from a lower bound on the girth.

A graph parameter $f(G)$ is called *partitionable* [8, 58, 82] if for every undirected simple graph G and positive real numbers a and b such that $f(G) < a + b$, the vertex set $V(G)$ can be partitioned into A and B so that $f(G[A]) < a$ and $f(G[B]) < b$. It is quite simple to prove that degeneracy [60], maximum degree [58] and treewidth [28] are all partitionable parameters. Hendrey, Norin and Wood asked whether mad is also partitionable as part of the open problems for Barbados workshop [1, Problem #12]. Such a result would agree exactly with or even improve many existing results, for example, the ones mentioned in [29, 54]. We answer this question positively for cases $a = 1$ and $a = 2$. It is a consequence of a following theorem which is the main result of this paper.

Theorem 1.3.1. *For every undirected simple graph G and a positive integer k such that $\mathrm{mad}(G) \geq k$ there exists $S \subseteq V(G)$ such that $G[S]$ is $(k-1)$-degenerate and $\mathrm{mad}(G - S) \leq \mathrm{mad}(G) - k$. Moreover such S can be computed in polynomial time.*

Up to our knowledge this is the first theorem of a kind where we are given a graph with bounded value of its mad where we partition its vertex set into some parts so that their values of mad are smaller, however they need not be bounded by absolute constant. This is opposed to all results where every resulting part induces a forest or is an independent set or has maximum degree 1, etc.

Our results can be applied as a tool for directly deriving many results for some specific sparse graph classes, for example planar graphs with constraints on girth. It seems that our results do not show as much expressive power as it is possible to get on such restrictive graph classes (where arguments specifically adjusted to the researched restricted graph classes can be used), which is a price for deriving them from a more general theorem. However, our results can be seen as a nice way of unifying these results and there are cases where using our results improves the state of the art.

Our results imply a positive answer for the open problem presented in [32] (Problem 2 from the final remarks), which implies a subexponential bound on the diameter of reconfiguration graphs of $(k + 2)$-colourings for graphs G with maximum average degree strictly less than $k + 1$. However, this bound has already been improved in [34] to a polynomial bound depending on the value of $\mathrm{mad}(G)$ in a slightly less general setting. Nevertheless, we are able to get a novel analogous result for reconfiguration graphs of H-colourings, in particular for circular colourings. This line of research is motivated by Cereceda's conjecture [15] that states that reconfiguration graphs of $(k + 2)$-colourings for k-degenerate graphs have a quadratic diameter. Its polynomial version was proved in [13].

1.4 Covered papers and organization of the Thesis

Results included in this dissertation come from the following papers:

- *Improved Bounds for the Excluded-Minor Approximation of Treedepth* [26], a joint work with Wojciech Czerwiński and Marcin Pilipczuk, published in SIAM Journal on Discrete Mathematics, Volume 35. This paper covers excluded minor theorem for and polynomial time approximation algorithm for treedepth, which are presented in Chapter 3.

- *Efficient fully dynamic elimination forests with applications to detecting long paths and cycles.* [18, 19], a joint work with Jiehua Chen, Wojciech Czerwiński, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Michał Pilipczuk, Marcin Pilipczuk, Manuel Sorge, Bartłomiej Wróblewski and Anna Zych-Pawlewicz, presented at Symposium on Discrete Algorithms 2021. The result that is included in this dissertation from this paper is the improved bound on sizes of minimal obstructions for small treedepth. It is presented in Chapter 4.

- *Computing Treedepth in Polynomial Space and Linear FPT Time* [62, 61], a joint work with Michał Pilipczuk and Marcin Smulewicz, presented on European Symposia on Algorithms 2022. This papers shows the polynomial space exact algorithm for computing treedepth, which is presented in Chapter 5.

- *Approximating Pathwidth for Graphs of Small Treewidth* [41], a joint work with Carla Groenland, Gwenaël Joret and Bartosz Walczak, published in ACM Transactions on Algorithms. This paper proves the excluded minor theorem for pathwidth, which is presented in Chapter 6. It also shows the polynomial time approximation algorithm for pathwidth, which is not present in this dissertation.

- *Decreasing the Maximum Average Degree by Deleting an Independent Set or a d-Degenerate Subgraph* [63], a joint work with Marcin Smulewicz, published in The Electronic Journal of Combinatorics Volume 29, Issue 1 (2022). This paper covers all mentioned results related to maximum average degree, which are presented in Chapter 7.

Chapter 2

Preliminaries

2.1 Graph notions

All graphs in this paper are finite and simple (i.e. with no loops on vertices or multiple edges with the same endpoints). All proven theorems are about undirected graphs, however some directed graphs may show up in their proofs.

An undirected edge between vertices u and v will be denoted as uv. A directed edge from u to v will be denoted as $\overrightarrow{uv}$.

If G is a graph and A is a subset of its vertices, then by $G[A]$ we denote the subgraph of G induced on vertices of A. The length of the shortest cycle in a graph G is called girth and will be denoted as $g(G)$. If G is a forest we set that $g(G) = \infty$. The maximum degree of a vertex in a graph is denoted $\Delta(G)$. The set of neighbours of a vertex v is denoted by $N_G(v)$ (or $N(v)$ if clear from the context) and the closed neighbourhood of v, that is $N_G(v) \cup \{v\}$ is denoted by $N_G[v]$. For $Y \subseteq V(G)$ we additionally denote $N_G[Y] \coloneqq \bigcup_{v \in Y} N_G[v]$. By $\overline{G}$ we denote the complement of G, that is the graph on the same set of vertices, where for $u, v \in V(G)$, such that $u \neq v$, we have $uv \in E(G) \Leftrightarrow uv \notin E(\overline{G})$.

We need a few basic notions concerning tree decompositions and treewidth. Recall that a *tree decomposition* of a graph G is a pair (T, β) where T is a rooted tree and $\beta : V(T) \to 2^{V(G)}$ is such that for every $v \in V(G)$ the set $\{t \in V(T) \mid v \in \beta(t)\}$ induces a connected nonempty subtree of T and for every $uv \in E(G)$ there exists $t \in V(T)$ with $u, v \in \beta(t)$. The width of a tree decomposition (T, β) is $\max_{t \in V(T)} |\beta(t)| - 1$ and the treewidth of a graph is the minimum possible width of its tree decomposition.

A *bramble* in a graph G is a family $\mathcal{B}$ of connected subgraphs of G such that for every $B_1, B_2 \in \mathcal{B}$, either B_1 and B_2 share a vertex or there is an edge of G with one endpoint in B_1 and one endpoint in B_2. Standard arguments (see e.g. [27]) show the following:

Lemma 2.1.1. *Let G be a graph, (T, β) be a tree decomposition of G, and let $\mathcal{B}$ be a bramble in G. Then there exists $t \in V(T)$ such that for every $B \in \mathcal{B}$ it holds that $\beta(t) \cap V(B) \neq \emptyset$.*

Consider a rooted forest F. By Anc_F we denote the ancestor/descendant relation in F: for $u, v \in V(F)$, $\mathsf{Anc}_F(u, v)$ holds if and only if u is an ancestor of v or v is an ancestor of u in F. We assume that a vertex is an ancestor of itself, so in particular $\mathsf{Anc}_F(u, u)$ is always true. We also use the following notation. For $u \in V(F)$, by $\mathsf{tail}_F[u]$ we denote the set of all ancestors of u (including u) and by $\mathsf{tree}_F[u]$ we denote the set of all descendants of u, including u. Further, let $\mathsf{tail}_F(u) = \mathsf{tail}_F[u] \setminus \{u\}$, $\mathsf{tree}_F(u) = \mathsf{tree}_F[u] \setminus \{u\}$, and $\mathsf{comp}_F[u] = \mathsf{tail}_F[u] \cup \mathsf{tree}_F[u]$. Note that $v \in \mathsf{comp}_F[u]$ if and only if $\mathsf{Anc}_F(u, v)$ holds. By $\mathsf{chld}_F(u)$ we denote the set of children of u in F, and by $\mathsf{depth}_F(u)$ we denote the depth of u in F, that is, $\mathsf{depth}_F(u) = |\mathsf{tail}_F[u]|$ (in particular, roots have depth one). The *depth* (or a *height*) of a rooted forest F is the maximum depth_F among its vertices. For a set of vertices $A \subseteq V(F)$, by $\mathsf{cl}_F(A) = \bigcup_{u \in A} \mathsf{tail}_F[u]$ we denote the ancestor closure of A. A *prefix* of a rooted forest F is a rooted forest induced by some ancestor-closed set $A \subseteq V(F)$; that is, it is the forest on A with the parent-child relation inherited from F.

Definition 2.1.2. An *elimination forest* of a graph G is a rooted forest F on the same set of vertices as G such that for every edge $uv \in E(G)$, we have that $\mathsf{Anc}_F(u, v)$ holds. The *treedepth* of a graph G is the least possible depth of an elimination forest of G.

Note that an elimination forest of a connected graph must be connected as well, so in this case we may speak about an *elimination tree*. Sometimes, instead of identifying $V(G)$ and $V(F)$, we treat them as disjoint sets and additionally provide a bijective mapping $\phi\colon V(G) \to V(F)$ such that $uv \in E(G)$ entails $\mathsf{Anc}_F(\phi(u), \phi(v))$. In such case we consider the pair (F, ϕ) to be an elimination forest of G. This will be always clear from the context. More generally, for $B \subseteq V(G)$ and a rooted forest F, we shall say that a mapping $\phi\colon B \to V(F)$ *respects edges* if $uv \in E(G)$ entails $\mathsf{Anc}_F(u, v)$ for all $u, v \in B$. In this notation, (F, ϕ) is an elimination forest of G if and only if ϕ is a bijection from $V(G)$ to $V(F)$ that respects edges on $V(G)$.

Moreover, we mention a few basic properties of treedepth that we are going to use:

1. If H is a subgraph of G then $\mathrm{td}(H) \leq \mathrm{td}(G)$
2. $\mathrm{td}(P_k) = \lceil \log_2(k + 1)\rceil$, where P_k is a path on k vertices
3. $\mathrm{td}(B_k) = k$, where B_k is a complete binary tree where every path from root to leaf has k vertices
4. $\mathrm{td}(G) = \begin{cases} 0, & \text{if } |V(G)| = 0 \\ \min_{v \in V(G)} 1 + \mathrm{td}(G \setminus v)), & \text{if G is connected} \\ \max_{C \in cc(G)} \mathrm{td}(G) & \text{otherwise (cc(G) is a set of connected components of G)} \end{cases}$
5. If H_1 and H_2 are two subgraphs of G such that $\mathrm{td}(H_1) = \mathrm{td}(H_2)$, $V(H_1) \cap V(H_2) = \emptyset$ and both of them are contained in one connected component of G, then $\mathrm{td}(G) > \mathrm{td}(H_1)$. That follows from the recursive definition of a treedepth from previous property and the fact that for any $v \in V(G)$ either H_1 or H_2 is fully contained within some connected component of $G \setminus v$.

A *complete binary tree of height h* is a rooted tree in which every non-leaf node has two children and every path from the root to a leaf has h edges. Such a tree has $2^{h+1} - 1$ nodes. A *complete ternary tree of height h* is defined analogously but with the requirement that every non-leaf node has three children. A *subdivision* of a tree T is a tree obtained from T by replacing each edge uv with some path connecting u and v whose internal nodes are new nodes private to that path.

The maximum average degree of a given graph G is defined as follows:

$$\mathrm{mad}(G) := \max_{H \subseteq G, H \neq \emptyset} \frac{2|E(H)|}{|V(H)|},$$

where $E(H)$ and $V(H)$ are respectively the set of edges in H and the set of vertices of H. We assume that mad of a graph with an empty vertex set is $-\infty$.

We say that undirected graph G is k-degenerate if each of its subgraphs contains a vertex of degree at most k. Degeneracy of a graph is the smallest value of k such that this graph is k-degenerate.

Let us note that class of 0-degenerate graphs is exactly the same class of graphs as graphs with $\mathrm{mad}(G) < 1$, because both are just edgeless graphs. Moreover class of 1-degenerate graphs is exactly the same class of graphs as graphs with $\mathrm{mad}(G) < 2$, because both are just forests.

2.2 Other notation

The symbol $\log_p$ stands for base-p logarithm and log stands for $\log_2$. We denote $\varphi = \frac{1+\sqrt{5}}{2}$; note that φ is chosen in a way so that $\varphi^2 = \varphi + 1$ and $\varphi > 1$.

For a function $f\colon A \to B$ and a subset of the domain $X \subseteq A$, by $f(X)$ we denote the image of f on X. The image of f is denoted $\mathrm{im}(f) = f(A)$. For an element e outside of the domain and a value α, by $f[e \to \alpha]$ we denote the extension of f obtained by additionally mapping e to α.

We denote the set $\{1, 2, \ldots, k\}$ as $[k]$. We assume the standard word RAM model of computation with words of length $\log n$, where n is the vertex count of the input graph.

Chapter 3

Improved bounds for the excluded-minor approximation of treedepth

In this Chapter we are going to prove a variety of results. The main one will be Theorem 1.1.4.

Theorem 1.1.4. *Let G be a graph of treewidth $\mathrm{tw}(G)$ and treedepth $\mathrm{td}(G)$. Then there exists a subcubic tree H that is a subgraph of G and is of treedepth at least*

$$\frac{\mathrm{td}(G)}{\mathrm{tw}(G)+1} \cdot \frac{\log((1+\sqrt{5})/2)}{\log(3)}.$$

As a consequence, we will get the Corollary 1.1.5 that improves upon the result of Kawarabayashi and Rossman [50].

Corollary 1.1.5. *Let G be a graph of treewidth $\mathrm{tw}(G)$ and treedepth $\mathrm{td}(G)$. Then for some*

$$h = \Omega\left(\sqrt{\mathrm{td}(G)/(\mathrm{tw}(G)+1)}\right)$$

G contains either a simple path of length 2^h or a subdivision of a full binary tree of depth h.

Consequently, there exists an absolute constant C such that for every integer $k \geq 1$ and a graph G of treedepth at least Ck^3, either

- *G has treewidth at least k,*
- *G contains a subdivision of a full binary tree of depth k as a subgraph, or*
- *G contains a path of length 2^k.*

The insight behind these theorems will let us to improve the best known approximation ratio for treedepth achievable in polynomial time, what is precisely stated within Theorem 1.1.7.

Theorem 1.1.7. *Given a graph G, one can in polynomial time compute a treedepth decomposition of G of width $\mathcal{O}(\mathrm{td}(G) \cdot \mathrm{tw}(G) \log^{3/2} \mathrm{tw}(G))$.*

Along the way, we are also going to prove a lemma about treedepth of trees that could be of independent interest, namely Lemma 1.1.6.

Lemma 1.1.6. *Every tree of treedepth d contains a subcubic subtree of treedepth at least $\frac{\log((1+\sqrt{5})/2)}{\log(3)} d$. Furthermore, such a subtree can be found in polynomial time.*

We are also going to show a theorem relating treedepth to linear colorings, improving upon the best previous bound of $k^{190}p(\log k)$ [57].

Theorem 1.1.8. *There exists a polynomial p such that for every integer k and graph G, if the treedepth of G is at least $k^{19}p(\log k)$, then every linear coloring of G requires at least k colors.*

3.1 Subcubic subtrees of trees of large treedepth

This section focuses on proving Lemma 1.1.6.

Let us recall that vertex ranking is any function $\alpha : V(G) \to \mathbb{Z}$ such that in every connected subgraph of G there is a unique vertex of maximum rank (value $\alpha(v)$).

Proposition 3.1.1. *Function $\alpha : V(G) \to \mathbb{Z}$ is a valid vertex ranking if and only if there are no two distinct vertices u and v such that $\alpha(u) = \alpha(v)$ and for all internal vertices w of a path between u and v we have $\alpha(w) < \alpha(u)$.*

Proof. For "$\Rightarrow$" direction, such path would be a connected subgraph violating condition for vertex ranking. For "$\Leftarrow$" direction, if we have a connected subgraph H of G with more than one occurrence of maximum value of a rank, then we can take path between closest pair of them to get path with mentioned properties. $\square$

Schäffer [80] proved that there is a linear time algorithm for finding a vertex ranking with minimum number of colors of a tree T. We follow [57] for a good description of its properties.

In original Schäffer's algorithm ranks are starting from 1, however for the ease of exposition let us assume that ranks are starting from 0. That is, the algorithm constructs a vertex ranking $\alpha : V(T) \to \{0, 1, 2, \ldots\}$ trying to minimize the maximum value attained by α. Assume that T is rooted in an arbitrary vertex and for every $v \in V(T)$ let T_v be the subtree rooted at v.

Of central importance to Schäffer's algorithm are what we will refer to as *rank lists*. For a rooted tree T, the rank list $L(T)$ for vertex ranking α consists of these ranks i for which there exists a path P starting from the root and ending in a vertex v with $\alpha(v) = i$ such that for every $u \in V(P) \setminus \{v\}$ we have $\alpha(u) < \alpha(v)$, that is, v is the unique vertex of maximum rank on P. We will call such vertices *visible from v*, where this name stems from logic that vertices with bigger ranks overshadow vertices with smaller ranks that are behind them. More formally:

Definition 3.1.2. For a vertex ranking α of tree T, the rank list of T, denoted $L(T)$, can be defined recursively as $L(T) = L(T \setminus T_v) \cup \{\alpha(v)\}$ where v is the vertex of maximum rank in T.

Schäffer's algorithm arbitrarily roots T and builds the ranking from the leaves to the root of T, computing the rank of each vertex from the rank lists of each of its children. For brevity, we denote $L(v) = L(T_v)$ for every v in T.

Proposition 3.1.3 ([80], notation follows [57])**.** *Let α be a vertex ranking of T produced by Schäffer's algorithm and let $v \in T$ be a vertex with children $u_1, \ldots, u_l$. If x is the largest integer appearing on rank lists of at least two children of v (or -1 if all such rank lists are pairwise disjoint) then $\alpha(v)$ is the smallest integer satisfying $\alpha(v) > x$ and $\alpha(v) \notin \bigcup_{i=1}^{l} L(u_i)$.*

This proposition constitutes a core part of Schäffer's algorithm which decides values $\alpha(v)$ greedily in bottom-up fashion. Condition that $\alpha(v) \notin \bigcup_{i=1}^{l} L(u_i)$ must hold, because otherwise we would have a path from v to some vertex visible from v with the same rank, violating condition from Proposition 3.1.1. Condition that $\alpha(v) > x$ must hold, because otherwise we would have path between two vertices visible from v which have the same rank, violating condition from Proposition 3.1.1, and v is the only internal vertex of that path left that could prevent this. Such conditions must be fulfilled for any vertex ranking and Schäffer proves in [80] that if we always greedily assign to the current vertex the minimum value of rank fulfilling these conditions, we will get optimal vertex ranking. From now on, we regard to α as specific vertex ranking which is produced by Schäffer's algorithm.

For a node $v \in V(T)$, and vertex ranking α, the following potential is pivotal to the analysis of Schäffer's algorithm. Let $l_0 > l_1 > \ldots > l_{|L(v)|-1}$ be the elements of $L(v)$ sorted in decreasing order.

$$\zeta(v) = \sum_{r \in L(v)} 3^r = \sum_{i=0}^{|L(v)|-1} 3^{l_i}.$$

When we write $\zeta(T)$ for some tree T we refer to $\zeta(s)$ where s is a root of T. For our purposes, we will also use a skewed version of potential function with a different base

$$\sigma(v) = \sum_{i=0}^{|L(v)|-1} \varphi^{l_i - i},$$

where again $l_0 > l_1 > \ldots > l_{|L(v)|-1}$ are elements of $L(v)$ sorted in decreasing order. Throughout this section, when focusing on one node $v \in V(T)$, we use notation that l_i is $i-$th element of set $L(v)$ when sorted in decreasing order and when $0-$based indexed.

Let us start with proving following two bounds that estimate $\operatorname{td}(T)$ in terms of $\zeta(T)$ and $\sigma(T)$.

Claim 3.1.4. $\log_\varphi(\sigma(T)) \geq \operatorname{td}(T) - 1$.

Proof. We know that $L(T)$ is nonempty and its biggest element is equal to $\operatorname{td}(T) - 1$ (we need to subtract one because we use nonnegative numbers as ranks, not positive). Therefore we have

$$\sigma(T) = \sum_{i=0}^{|L(T)|-1} \varphi^{l_i - i} \geq \varphi^{l_0} = \varphi^{\operatorname{td}(T)-1}.$$

Hence, $\log_\varphi(\sigma(T)) \geq \operatorname{td}(T) - 1$, as desired. ⌟

Claim 3.1.5. $\log_3(\zeta(T)) + \log_3(2) < \operatorname{td}(T)$.

Proof. We have that

$$\begin{aligned} \zeta(T) = \sum_{r \in L(v)} 3^r \leq \sum_{r=0}^{\operatorname{td}(T)-1} 3^r = \frac{3^{\operatorname{td}(T)} - 1}{2}, \\ 2\zeta(T) \leq 3^{\operatorname{td}(T)} - 1 < 3^{\operatorname{td}(T)} \\ \log_3(2) + \log_3(\zeta(T)) < \operatorname{td}(T). \end{aligned}$$

⌟

We are ready to prove Lemma 1.1.6. Given tree T we want to produce a subcubic (i.e., maximum degree at most 3) tree S which is a subtree of T and that fulfills $\operatorname{td}(S) > \operatorname{td}(T) \log_3(\varphi)$.

Let us start our algorithm by arbitrary rooting T and computing rank lists using Schäffer's algorithm. Then for every vertex $v \in T$ we define $C(v)$ as a set of two children of v that have the biggest value of ζ in case v has at least two children, or all children otherwise. Let us now define forest F whose vertex set is the same as vertex set of T where for every v we put edges between v and all elements of $C(v)$. Clearly this is a forest consisting of subcubic trees which are subtrees of T (where *subtree* is understood as subgraph, not necessarily as some vertex t along with all its descendants in a rooted tree). Let S be a tree of this forest containing root of T. We claim that S is that subcubic subtree of T we are looking for. Note that computing F and thus S can be trivially done in polynomial time. Hence, we are left with proving that $\operatorname{td}(S) > \operatorname{td}(T) \log_3(\varphi)$.

Let us root every tree of F in a vertex that was closest to root of T in T. Then compute rank lists for these trees using Schäffer's algorithm. So now, for every vertex we have two rank lists, one for T and one for F. Let us now denote these second ranklists as $\widetilde{L}(v)$ for $v \in V(T)$ and let us define function $\widetilde{\zeta}$ which will be similar potential function as ζ, but operating on rank lists $\widetilde{L}(v)$ instead of $L(v)$. Following claim will be crucial.

Claim 3.1.6. *For every $v \in V(T)$ it holds that $\widetilde{\zeta}(v) \geq \sigma(v)$.*

We first verify that Claim 3.1.6 implies Lemma 1.1.6.

Proof of Lemma 1.1.6. Using also Claims 3.1.5 and 3.1.4 we infer that

$$\begin{aligned}
\mathrm{td}(S) &> \log_3(\widetilde{\zeta}(S)) + \log_3(2) && \text{by Claim 3.1.5 for } S\\
&\geq \log_3(\sigma(T)) + \log_3(2) && \text{by Claim 3.1.6}\\
&= \log_\varphi(\sigma(T)) \cdot \log_3(\varphi) + \log_3(2) && \text{logarithm base change}\\
&\geq (\mathrm{td}(T) - 1) \cdot \log_3(\varphi) + \log_3(2) && \text{by Claim 3.1.4 for } T\\
&= \mathrm{td}(T) \cdot \log_3(\varphi) - \log_3(\varphi) + \log_3(2) > \mathrm{td}(T) \cdot \log_3(\varphi).
\end{aligned}$$

□

Thus it remains to prove Claim 3.1.6. To this end, we prove two auxiliary inequalities.

Claim 3.1.7. *For every $v \in V(T)$ it holds that $\widetilde{\zeta}(v) \geq 1 + \sum_{s \in C(v)} \widetilde{\zeta}(s)$*

Proof. We express every $\widetilde{\zeta}(x)$ for $x \in \{v\} \cup C(v)$ as a sum of powers of 3 and count how many times each power occurs on both sides of this claimed inequality. Consider a summand 3^c. If $c > \alpha(v)$ then, by the choice of $\alpha(v)$, 3^c appears at most once on the right side and if it appears there, then it appears on the left side as well, so contributions of summands of form 3^c for $c > \alpha(v)$ to both sides are equal. The summand $3^{\alpha(v)}$ appears once on the left side and does not appear on the right side. For $c < \alpha(v)$, the summands of form 3^c appear at most twice in $\sum_{s \in C(v)} \widetilde{\zeta}(s)$, so their contribution to right side can be bounded from above by $\sum_{c=0}^{\alpha(v)-1} 2 \cdot 3^c = 3^{\alpha(v)} - 1$, so in fact $3^{\alpha(v)}$ from the left side contributes at least as much as remaining summands from the right side. This finishes the proof of the claim. ⌟

Claim 3.1.8. *For every $v \in V(T)$ it holds that $\sigma(v) \leq 1 + \sum_{s \in C(v)} \sigma(s)$*

Proof. Recall that by the definition $C(v)$ is a set of two children of v in T with the biggest values of ζ or a set of all children of v in case it has less than two of them. Observe that having bigger value of $\zeta(v)$ is another way of expressing having the set $L(v)$ bigger lexicographically when sorted in decreasing order.

If v is a leaf then $C(v)$ is empty and $\sigma(v) = 1$, so the inequality is obvious. Henceforth we focus on a vertex v that is not a leaf. In our proof following equation will come handy:

$$\varphi = \sum_{i=0}^{\infty} \varphi^{-2i}$$

It holds since $\sum_{i=0}^{\infty} \varphi^{-2i} = \frac{1}{1-\varphi^{-2}} = \frac{\varphi^2}{\varphi^2-1} = \frac{\varphi^2}{\varphi} = \varphi$.

Let us now analyze $L(v)$. It consists of some prefix P of values that appeared exactly once in children of v, then $\alpha(v)$ and then nothing (when enumerated from the biggest to the smallest). Let us now denote by A_i intersection of $L(u_i)$ and P, where u_i is i−th child of v when sorted in nonincreasing order by their values $\zeta(u_i)$ (1-based). We distinguish two cases:

Case 1: A_2 is nonempty. If A_2 is nonempty then in particular it means that v has at least two children. Let us denote the biggest element of $L(u_2)$ by d. We have that $d \in P$, but d is not the biggest element of P. Its contribution to $\sigma(u_2)$ is φ^d, however its contribution to $\sigma(v)$ is at most φ^{d-1} (because of the skew and since d is not the biggest element of P). Contribution to $\sigma(v)$ of elements smaller than d can be bounded from above by $\varphi^{d-3} + \varphi^{d-5} + \ldots$. We know that $d = l_j$ for some j, where $j \geq 1$ and $L(v)$ consists of elements $l_0 > l_1 > \ldots > l_{|L(v)-1|}$. We have that $l_k \in L(u_1)$ for $k < j$ and that $l_j \geq l_i + (i - j)$ for $i \geq j$, so $l_i - i \leq l_j - j - 2(i - j) = d - j - 2(i - j)$.

We can deduce that

$$\sigma(v) = \sum_{i=0}^{|L(v)|-1} \varphi^{l_i - i} = \sum_{i=0}^{j-1} \varphi^{l_i - i} + \sum_{i=j}^{|L(v)|-1} \varphi^{l_i - i} \leq \sigma(u_1) + \sum_{i=j}^{|L(v)|-1} \varphi^{d-j-2(i-j)}$$

$$\leq \sigma(u_1) + \varphi^{d-j} \sum_{i=j}^{\infty} \varphi^{-2(i-j)} = \sigma(u_1) + \varphi^{d-j} \sum_{i=0}^{\infty} \varphi^{-2i} = \sigma(u_1) + \varphi^{d-j+1}$$

$$\leq \sigma(u_1) + \varphi^d \leq \sigma(u_1) + \sigma(u_2) < 1 + \sigma(u_1) + \sigma(u_2),$$

which is what we wanted to prove.

Case 2: A_2 is empty. Let us now introduce a few variables:

- d - the biggest integer number smaller than $\alpha(v)$ that is not an element of $L(u_1)$. We know that elements from $d+1$ to $\alpha(v)-1$ belong to $L(u_1)$.
- k - shorthand for number of these elements (which is equal to $\alpha(v) - 1 - d$). k can be zero, but cannot be negative.
- g - the number of elements of $L(v)$ that are bigger than $\alpha(v)$.

Then from the definition of $\alpha(v)$ either

- $d = -1$; or
- v has at least two children and $L(u_2)$ contains a number that is at least d.

Let us now argue that $1 + \sum_{s \in C(v)} \sigma(s) \geq \sigma(u_1) + \varphi^d$. We know that $\sum_{s \in C(v)}$ is either $\sigma(u_1)$ or $\sigma(u_1) + \sigma(u_2)$, depending on whether v has only one child or more. If $d = -1$ then $1 \geq \varphi^d$ and stated inequality holds. If $d \neq -1$ then u_2 exists and $\sigma(u_2) \geq \varphi^d$.

Let us now argue that $k > 0$ or $g > 0$ and therefore $k + g \geq 1$. If $k = g = 0$ then $d = \alpha(v) - 1$ and $L(u_1)$ cannot contain elements bigger then $\alpha(v)$ (because $g = 0$), cannot contain $\alpha(v)$ (from the definition of $\alpha(v)$) and cannot contain $\alpha(v) - 1$ (since $d = \alpha(v) - 1$), so its biggest element is at most $d - 1$. If $d = -1$ then it means that v is a leaf, but we already assumed it is not one. However, if v has at least two children and $L(u_2)$ contains a number that is at least d, then it contradicts the assumption that $\zeta(u_1) \geq \zeta(u_2)$. So indeed it holds that $k > 0$ or $g > 0$ and therefore $k + g \geq 1$.

We have that

$$\sigma(v) - \sigma(u_1) \leq \varphi^{\alpha(v)-g} - (\varphi^{\alpha(v)-g-1} + \varphi^{\alpha(v)-g-3} + \ldots + \varphi^{\alpha(v)-g-2k+1}),$$

which is because summands coming from numbers bigger than $\alpha(v)$ in $L(v)$ and $L(u_1)$ cancel out (A_2 is empty, so all elements of $L(v)$ different than $\alpha(v)$ come from $L(u_1)$) and new rank $\alpha(v)$ contributes $\varphi^{\alpha(v)-g}$ to $\sigma(v)$ whereas $L(u_1)$ contains numbers from $d+1$ up to $\alpha(v)-1$ and their contribution to $\sigma(u_1)$ is $\varphi^{\alpha(v)-g-1} + \varphi^{\alpha(v)-g-3} + \ldots + \varphi^{\alpha(v)-g-2k+1}$.
We conclude that $\sigma(v) - \sigma(u_1) \leq \varphi^{-g}(\varphi^{\alpha(v)} - (\varphi^{\alpha(v)-1} + \varphi^{\alpha(v)-3} + \ldots + \varphi^{\alpha(v)-2k+1}))$.
On the other hand since $\varphi^2 = \varphi + 1$ we have that

$$\varphi^{\alpha(v)} = \varphi^{\alpha(v)-1} + \varphi^{\alpha(v)-2} = \varphi^{\alpha(v)-1} + \varphi^{\alpha(v)-3} + \varphi^{\alpha(v)-4} = \ldots =$$
$$= (\varphi^{\alpha(v)-1} + \varphi^{\alpha(v)-3} + \ldots + \varphi^{\alpha(v)-2k+1}) + \varphi^{\alpha(v)-2k}.$$

Because of that we have

$$\sigma(v) - \sigma(u_1) \leq \varphi^{-g} \cdot \varphi^{\alpha(v)-2k} = \varphi^{\alpha(v)-2k-g} = \varphi^{\alpha(v)-(\alpha(v)-1-d)-k-g} = \varphi^{d+1-(k+g)} \leq \varphi^d.$$

From that we conclude that $\sigma(v) \leq \sigma(u_1) + \varphi^d \leq 1 + \sum_{s \in C(v)} \sigma(s)$, what concludes proof of this claim. ⌟

Now, having claims 3.1.8 and 3.1.7 proven, we can wrap our reasoning up. If v is a leaf then $\sigma(v) = \widetilde{\zeta}(v) = 1$. If v is not a leaf then we know that $\sigma(v) \leq 1 + \sum_{s \in C(v)} \sigma(s)$ and $\widetilde{\zeta}(v) \geq 1 + \sum_{s \in C(v)} \widetilde{\zeta}(s)$, so by straightforward induction we get that $\sigma(v) \leq \widetilde{\zeta}(v)$ for every $v \in V(T)$, as desired by Claim 3.1.6.

3.2 Proof of Theorem 1.1.4

Theorem 1.1.4 is a direct corollary of Lemma 1.1.6 and the following statement.

Theorem 3.2.1. *Let G be a graph and a, b be positive integers. If the treewidth of G is less than a and G does not contain any tree of treedepth more than b as a subgraph, then the treedepth of G is at most ab.*

Proof. We prove the fact by induction on b. For $b = 1$, if G contains no tree of treedepth 2 as a subgraph, then G is edgeless, and its treedepth is at most $1 \leq ab$, as desired.

Assume now $b > 1$ and that the statement is true for all $b' < b$. Without loss of generality assume that G is connected, as otherwise we prove the treedepth bound for each connected component of G separatedly.

Let $\mathcal{B}$ be the family of all subgraphs of G that are trees of treedepth exactly b. The crucial observation is the following.

Claim 3.2.2. *For every two $B_1, B_2 \in \mathcal{B}$, $V(B_1) \cap V(B_2) \neq \emptyset$.*

Proof. Assume the contrary and let $B_1, B_2 \in \mathcal{B}$ be two offending trees in $\mathcal{B}$. Since $b > 1$, B_1 and B_2 are nonempty. Let P be a shortest path from $V(B_1)$ and $V(B_2)$; it exists as G is connected. Define a subgraph B of G being the union of B_1, P, and B_2. Since P is a shortest path from B_1 to B_2, B is a tree. However, since the treedepth of B_1 and B_2 equals b, the treedepth of B is more than b, a contradiction. ⌟

Claim 3.2.2 implies that $\mathcal{B}$ is a bramble in G.

Let (T, β) be a tree decomposition of G of minimum width. By Lemma 2.1.1, there exists a bag $\beta(t)$ for some $t \in V(T)$ that intersects $V(B)$ for every $B \in \mathcal{B}$. By the definition of $\mathcal{B}$, every tree that is a subgraph of $G' := G - \beta(t)$ has treedepth less than b. By the inductive hypothesis, the treedepth of G' is at most $a \cdot (b-1)$. Thus, the treedepth of G is at most $\mathrm{td}(G') + |\beta(t)| \leq a \cdot (b-1) + a = ab$, as desired. □

3.3 Proof of Theorem 1.1.7

We consider a greedy tree decomposition of a connected graph G, as defined in [50]. A greedy tree decomposition is a tree decomposition that can be also interpreted as a treedepth decomposition. More formally, a tree decomposition (T, β) of a graph G is *greedy* if

1. $V(T) = V(G)$,
2. for every $uv \in E(G)$, the nodes u and v in T are in ancestor-descendant relation in T, and
3. for every vertex $u \in V(T)$ and its child v there is some descendant w of v in T such that $uw \in E(G)$.

For the proof of Theorem 1.1.7, without loss of generality we can assume that the input graph G is connected. As in the proof of Lemma 1.1.2, we apply the polynomial-time approximation algorithm for treewidth [36], to compute a tree decomposition (T_0, β_0) of G with $\mathcal{O}(n)$ nodes of T_0 and $|\beta(t)| \leq \tau$ for every $t \in V(T_0)$ and some $\tau = \mathcal{O}(\mathrm{tw}(G)\sqrt{\log \mathrm{tw}(G)})$. As discussed in [50], one can in polynomial time turn (T_0, β_0) into a greedy tree decomposition (T, β) of G without increasing the maximum size of a bag, that is, still $|\beta(t)| \leq \tau$ for every $t \in V(T)$. We apply Lemma 1.1.1 to (T, β), returning a treedepth decomposition of G of width at most $\tau \cdot \mathrm{td}(T) = \mathcal{O}(\mathrm{td}(T)\mathrm{tw}(G)\sqrt{\log \mathrm{tw}(G)})$. To conclude the proof of Theorem 1.1.7, it remains to bound the approximation ratio by proving that $\mathrm{td}(T) = \mathcal{O}(\mathrm{td}(G) \log \mathrm{tw}(G))$.

To this end, we prove the following lemma that combines Lemma 1.1.6 with the machinery of Kawarabayashi and Rossmann [50].

Lemma 3.3.1. *Let G be a connected graph, (T, β) be a greedy tree decomposition of G, and let $\tau \geq 2$ be such that $|\beta(t)| \leq \tau$ for every $t \in V(T)$. Then G contains a subcubic tree of treedepth $\Omega(\mathrm{td}(T)/\log \tau)$.*

To prove Lemma 3.3.1, we first apply Lemma 1.1.6 to tree T and obtain a subcubic tree S such that

$$\mathrm{td}(S) \geq \mathrm{td}(T) \cdot \log_3(\varphi). \tag{3.1}$$

Second, we apply the core part of the reasoning of Kawarabayashi and Rossman [50]. The construction of Section 5 of [50] can be encapsulated in the following lemma.

Lemma 3.3.2 (Section 5 of [50])**.** *Let (T, β) be a greedy tree decomposition of graph G and let $\tau = \max_{t\in V(T)} |\beta(t)|$. Then for every subcubic subtree S of T there exists a subtree F of G such that $V(S) \subseteq V(F)$ and the maximum degree of F is bounded by $\tau + 2$.*

By application of Lemma 3.3.2 to our decomposition (T, β) and subtree S we get a tree F in G, which has large treedepth, as we show in a moment. To this end, we need the following simple bound on treedepth of trees.

Lemma 3.3.3. *For every tree H with maximum degree bounded by $d \geq 2$ it holds that*

$$\log_d |V(H)| \leq \mathrm{td}(H) \leq 1 + \log_2 |V(H)|.$$

Proof. We use the following equivalent recursive definition of treedepth: Treedepth of an empty graph is 0, treedepth of a disconnected graph equals the maximum of treedepth over its connected components, while for nonempty connected graphs G we have $\mathrm{td}(G) = 1 + \min_{v\in V(G)} \mathrm{td}(G - v)$.

For the lower bound, for $k \geq 1$ let $f_d(k)$ be the maximum possible number of vertices of a tree of maximum degree at most d and treedepth at most k. Clearly, $f_d(1) = 1$. Since removing a single vertex from a tree of maximum degree at most d results in at most d connected components, we have that

$$f_d(k + 1) \leq 1 + d \cdot f_d(k).$$

Consequently, we obtain by induction that

$$f_d(k) \leq d^k - 1.$$

This proves the lower bound. For the upper bound, note that in every tree T there exists a vertex $v \in V(T)$ such that every connected component of $T - \{v\}$ has at most $|V(T)|/2$ vertices. Consequently, if we define $g(n)$ to be the maximum possible treedepth of an n-vertex tree, then $g(1) = 1$ and we have that

$$g(n) \leq 1 + g(\lfloor n/2 \rfloor).$$

This proves the upper bound. □

By (3.1) and Lemma 3.3.3 we get that $|V(S)| \geq 2^{\mathrm{td}(T)\cdot\log_3(\varphi)-1}$. This implies that also

$$|V(F)| \geq 2^{\mathrm{td}(T)\cdot\log_3(\varphi)-1}. \tag{3.2}$$

As S is subcubic, by Lemma 3.3.2 we know that the maximum degree of F is bounded by $\tau+2$. Therefore Lemma 3.3.3 and (3.2) jointly imply that

$$\mathrm{td}(F) \geq \log_{(\tau+2)} 2^{\mathrm{td}(T)\cdot\log_3(\varphi)-1} \geq \frac{\mathrm{td}(T) \cdot \log_3(\varphi) - 1}{\log(\tau + 2)} = \Omega(\mathrm{td}(T)/\log \tau). \tag{3.3}$$

Here, the last inequality follows from the assumption $\tau \geq 2$.

As tree F is not necessarily subcubic, we apply one more time Lemma 1.1.6 and get a subcubic subtree H of F such that

$$\mathrm{td}(H) \geq \mathrm{td}(F) \cdot \log_3(\varphi) = \Omega(\mathrm{td}(T)/\log \tau). \tag{3.4}$$

which finishes the proof of Lemma 3.3.1.

With Lemma 3.3.1 in hand, we are ready to conlude the proof of Theorem 1.1.7. Recall that it suffices to prove $\mathrm{td}(T) = \mathcal{O}(\mathrm{td}(G) \log \mathrm{tw}(G))$.

Lemma 3.3.1 asserts that G contains a subcubic tree H of treedepth $\Omega(\mathrm{td}(T)/\log \tau)$. Therefore $\mathrm{td}(T) = \mathcal{O}(\mathrm{td}(H) \log \tau) = \mathcal{O}(\mathrm{td}(G) \log \mathrm{tw}(G))$ and thus the width of the computed treedepth decomposition is $\mathcal{O}(\mathrm{td}(G)\mathrm{tw}(G) \log^{3/2} \mathrm{tw}(G))$. This finishes the proof of Theorem 1.1.7.

3.4 Proof of Theorem 1.1.8

Here we show how to assemble the proof of Theorem 1.1.8 from Theorem 1.1.4, a number of intermediate results of [57], and an improved excluded grid theorem due to Chuzhoy and Tan [21]:

Theorem 3.4.1 ([21])**.** *There exists a polynomial p_{GMT} such that for every integer k if a graph G has treewidth at least $k^9 p_{\text{GMT}}(\log k)$ then G contains a $k \times k$ grid as a minor.*

The following two results were proven in [57].

Lemma 3.4.2 ([57])**.** *There exists a constant $c_{\text{grid}} > 0$ such that if a graph G contains a $k \times k$ grid as a minor, then every linear coloring of G requires $c_{\text{grid}} \cdot \sqrt{k}$ colors.*

Lemma 3.4.3 ([57])**.** *If G is a tree of treedepth d and maximum degree Δ, then every linear coloring of G requires at least $d/\log_2(\Delta)$ colors.*

Recall that Theorem 1.1.4 asserts that there exists a constant C such that for every graph G and integers $a, b \geq 2$, if the treedepth of G is at least Cab, then either the treewidth of G is at least a or G contains a subcubic tree of treedepth at least b. Applying this theorem to $a = \lceil k/c_{\text{grid}} \rceil^2$ and $b = \lceil k \log_2(3) \rceil$, one obtains that if the treedepth of G is $\Omega(k^{19} p_{\text{GMT}}(\log k))$, then G contains either a $(\lceil k/c_{\text{grid}} \rceil^2) \times (\lceil k/c_{\text{grid}} \rceil^2)$ grid minor or a subcubic tree of treedepth at least $k \log_2(3)$. In the first outcome, Lemma 3.4.2 gives the desired number of colors of a linear coloring, while in the second outcome the same result is obtained from Lemma 3.4.3. This concludes the proof of Theorem 1.1.8.

3.5 An example of a tree with treedepth quadratic in the height of the binary tree or logarithm of a length of a path

In this section we provide a construction of a family of trees $(G_n)_{n \geq 1}$ such that

1. The tree G_n does not contain a path with 2^{n+2} vertices.
2. The tree G_n does not contain a subdivision of a full binary tree of depth $n+2$.
3. The treedepth of G_n is at least $\binom{n+1}{2}$.

We will consider each tree G_n as a rooted tree. The tree G_1 consists of a single vertex. For $n \geq 2$, the tree G_n is defined recursively as follows. We take a path P_n with 2^n vertices and for each $v \in V(P_n)$ we create a copy C_n^v of G_{n-1} and attach its root to v. We root G_n in one of the endpoints of P_n; see Figure 3.1.

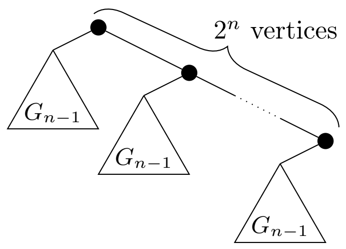

Figure 3.1: Construction of G_n.

We now proceed with the proof of the properties of G_n. Since every path in G_n is contained in a union of at most two root-to-leaf paths (not necessarily edge-disjoint), to show Property (1) it suffices to show the following.

Lemma 3.5.1. *Every root-to-leaf path in G_n contains less than 2^{n+1} vertices.*

Proof. We prove the statement by induction on n. For $n = 1$ the statement is straightforward. For the inductive step, observe that every root-to-leaf path in G_n consists of a subpath of P_n (which has 2^n vertices) and a root-to-leaf path in one of the copies C_n^v of G_{n-1} (which has less than 2^n vertices by the inductive assumption). □

We say that a subtree H of G_n that is a subdivision of a full binary tree of height $h \geq 1$ is *aligned* if $h = 1$ or $h \geq 2$ and the closest to the root vertex of H is of degree 2 in H and its deletion breaks H into two subtrees containing a subdivision of a full binary tree of height $h - 1$. In other words, an aligned subtree has the same ancestor-descendant relation as the tree G_n. Observe that any subtree H_0 of G_n that is a subdivision of a full binary tree of height $h \geq 2$ contains a subtree that is an aligned subdivision of a full binary tree of height $h - 1$. Therefore, to prove Property (2), it suffices to show the following.

Lemma 3.5.2. *G_n does not contain an aligned subdivision of a full binary tree of height $n + 1$.*

Proof. We prove the claim by induction on n. It is straightforward for $n = 1$. For $n \geq 2$, assume for contradiction that H is such an aligned subtree of G_n and let w be the closest to the root of G_n vertex of H. If $w \in V(C_n^v)$ for some $v \in V(P_n)$, then H is completely contained in C_n^v, which is a copy of G_{n-1}. Otherwise, $w \in V(P_n)$ and thus one of the components of $H - \{w\}$ lies in C_n^w. However, this component contains an aligned subdivision of a full binary tree of height n. In both cases, we obtain a contradiction with the inductive assumption. □

We are left with the treedepth lower bound of Property (3). To this end, we consider the following families of trees. For integers $a \geq 0, b \geq 1$, the family $\mathcal{G}_{a,b}$ contains all trees H that are constructed from a path P_H with at least 2^a vertices by attaching, for every $v \in V(P_H)$, a tree T_v of treedepth at least b by an edge to v. We show the following.

Lemma 3.5.3. *For every $H \in \mathcal{G}_{a,b}$ we have $\mathrm{td}(H) \geq a + b$.*

Proof. We prove the lemma by induction on a. For $a = 0$ it is trivial. Assume then $a \geq 1$ and $H \in \mathcal{G}_{a,b}$. Then for every $v \in V(H)$, $H - v$ contains a connected component that contains a subtree belonging to $\mathcal{G}_{a-1,b}$. This finishes the proof. □

We show Property (3) by induction on n. Clearly, $\mathrm{td}(G_1) = 1 = \binom{1+1}{2}$. Consider $n \geq 2$. Since the treedepth of G_{n-1} is at least $\binom{n}{2}$, we have that $G_n \in \mathcal{G}_{n,\binom{n}{2}}$. By Lemma 3.5.3, we have that

$$\mathrm{td}(G_n) \geq n + \binom{n}{2} = \binom{n+1}{2}.$$

This finishes the proof of Property (3).

Chapter 4

Obstructions for small treedepth

To recall, the main goal of this Chapter is to improve the upper bound on the size of minimal obstructions for having a small treedepth. More specifically, we are going to prove the already mentioned Theorem 1.1.9.
Theorem 1.1.9. *If G is a minimal obstruction for treedepth d, then the vertex count of G is at most*

$$(d+1) \cdot \frac{\big((d+1)((d+1)^2+1)\big)^{d+1} - 1}{(d+1)((d+1)^2+1) - 1} \in d^{\mathcal{O}(d)}.$$

In order to do that, first we are going to recall a few basic lemmas, then we are going to introduce a key notion of a *treedepth core* and then, having done all the necessary preparations, we will conclude by proving Theorem 1.1.9.

4.1 Basic terminology and treedepth tools

A *rooted forest* F is a directed acyclic graph in which each vertex u has at most one out-neighbor, called the *parent* of u and denoted by $\mathsf{parent}_F(u)$. A vertex u is a *root* of F if it has no parent, which we denote by $\mathsf{parent}_F(u) = \bot$. The set of roots of a forest F is denoted by roots_F. Two vertices of F that either are both roots or have the same parent are called *siblings*.

By F_u we denote the subtree of F induced by the descendants of u.

A subset of vertices $X \subseteq V(F)$ is *straight* in F if for all $u, v \in X$, either u is an ancestor of v in F or v is an ancestor of u in F. Equivalently, vertices of a straight set lie on one leaf-to-root path in F. Here, by a root-to-leaf path in F we mean a path connecting a leaf with the root of some tree in F.

A *prefix* of a forest F is an ancestor-closed subset of vertices, that is, $A \subseteq V(F)$ is a prefix if $u \in A$ implies $\mathsf{tail}_F[u] \subseteq A$. The set of *appendices* of a prefix A, denoted $\mathsf{App}_F(A)$, comprises all ancestor-minimal elements of $V(F) \setminus A$, that is, vertices $u \notin A$ such that either $u \in \mathsf{roots}_F$ or $\mathsf{parent}_F(u) \in A$. Note that for all $u \in \mathsf{App}_F(A)$, we have $\mathsf{tail}_F(u) \subseteq A$. If A is a prefix of F, then by $F - A$ we denote the forest obtained from F by removing all the vertices of A and keeping the parent/child relation on the remaining vertices intact.

If F is an elimination forest of a graph G and $u \in V(G)$, then we define the *strong reachability set* of u:

$$\mathsf{SReach}_{F,G}(u) \coloneqq N_G(\mathsf{tree}_F[u]).$$

We remark that the name *strong reachability set* comes from the theory of structural sparsity, where this concept is present and is an analogue of the definition above; see e.g. [52, 43, 85]. Note that for every vertex u we have $\mathsf{SReach}_{F,G}(u) \subseteq \mathsf{tail}_F(u)$.

An elimination forest of G is *optimal* if its height is equal to the treedepth of G. We will need a more refined notion of "local" optimality, as expressed next.

Definition 4.1.1. An elimination forest F of G is *recursively optimal* if for every $u \in V(G)$, we have that:

- the graph $G[\mathsf{tree}_F[u]]$ is connected; and
- the tree F_u is an optimal elimination forest of $G[\mathsf{tree}_F[u]]$.

We remark that Dvořák et al. [30] also use this definition; they call such elimination forests just "optimal". We will also widely use a weakened version of this definition given below.

Definition 4.1.2. An elimination forest F of G is *recursively connected* if for every $u \in V(G)$, we have that the graph $G[\mathsf{tree}_F[u]]$ is connected.

Let us note here that if F is an elimination forest of a graph G and A is a prefix of F, then $F - A$ is an elimination forest of $G - A$. If F is moreover recursively connected or recursively optimal, then the same can be also said about $F - A$.

We now point out a simple, yet important property of recursively connected elimination forests.

Lemma 4.1.3. *Suppose F is a recursively connected elimination forest of a graph G. Let u be a vertex of G and v be a child of u in F. Then $u \in \mathsf{SReach}_{F,G}(v)$.*

Proof. Otherwise, no vertex of $\mathsf{tree}_F[v]$ would have a neighbor in $\mathsf{tree}_F[u] \setminus \mathsf{tree}_F[v]$, so the graph $G[\mathsf{tree}_F[u]]$ would not be connected. ☐

Clearly, every recursively optimal elimination forest is recursively connected, as we require that explicitly in the definition. Note also that every recursively optimal elimination forest is in particular optimal, as it optimally decomposes each connected component of the graph.

Finally, we will use some basic properties of elimination forests related to the connectivity in graphs. First, the following fact is well-known.

Lemma 4.1.4. *Suppose that F is an elimination forest of a graph G and $A \subseteq V(G)$ is such that $G[A]$ is connected. Then there exists $x \in A$ such that $A \subseteq \mathsf{tree}_F[x]$.*

Proof. Let x be any vertex of A that minimizes $\mathsf{depth}_F(x)$. Let $B \coloneqq A \cap \mathsf{tree}_F[x]$ and suppose, for contradiction, that $A \setminus B$ is non-empty. Observe that every vertex $y \in A \setminus B$ can be neither an ancestor of x — by the minimality of x — nor a descendant of x — for it would be included in B. Hence, for each $y \in A \setminus B$ and $z \in B$, the set $\{y, z\}$ is not straight in F. As F is an elimination forest of G, this implies that there is no edge between B and $A \setminus B$. Since B is non-empty due to containing x, this is a contradiction to the assumption that $G[A]$ is connected. ☐

Next, vertices x and y of a graph G are *k-vertex-connected* in G if there exists k internally vertex-disjoint paths with endpoints x and y. We will use the following simple claim.

Lemma 4.1.5. *Suppose that x and y are d-vertex-connected in a graph G. Then for every elimination forest F of G of height at most d, the set $\{x, y\}$ is straight in F.*

Proof. If $\{x, y\}$ was not straight in F, then every path connecting x and y would have to contain an internal vertex that belongs to $\mathsf{anc}_F(x) \cap \mathsf{anc}_F(y)$. Since $|\mathsf{anc}_F(x) \cap \mathsf{anc}_F(y)| < \mathsf{height}(F) \leq d$, this implies that there cannot be d internally vertex-disjoint paths connecting x and y in G, a contradiction. ☐

4.2 Treedepth cores

We now introduce the most important definition in this chapter: the *core* of an elimination forest of a graph. Intuitively, this is a relatively small subset of vertices that retains all the relevant connectivity information that is essential for, say, treedepth computation.

Definition 4.2.1. Suppose that F is an elimination forest of a graph G. For $q \in \mathbb{N}$, a non-empty prefix K of F is called a *q-core* of (G, F) if the following condition holds for every vertex $u \in \mathsf{App}_F(K)$: for each $X \subseteq \mathsf{SReach}_{F,G}(u)$ with $|X| \leq 2$, there exist at least q siblings w of u such that $w \in K$, $X \subseteq \mathsf{SReach}_{F,G}(w)$, and $\mathsf{height}(F_w) \geq \mathsf{height}(F_u)$.

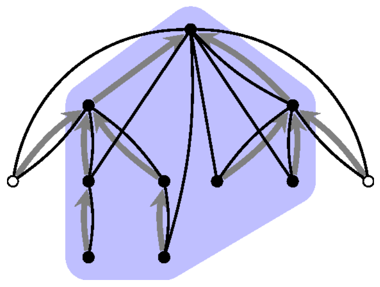
Figure 4.1: A graph G (black edges), an elimination tree F of G (thick gray arcs, directed from children to parents), and a 2-core of F (vertices on blue background). White vertices are in $\mathsf{App}_F(K)$.

See Figure 4.1 for an illustration.

Before we proceed further, let us observe that in recursively optimal elimination forests we can always find q-cores of size bounded by a function of q and the height. In the following, for a set A, by $\binom{A}{\leq 2}$ we denote the set of all subsets of X of size at most 2.

Lemma 4.2.2. *Let F be a recursively optimal elimination forest of a graph G and let d be the height of F. Then for every $q \in \mathbb{N}$, there is a q-core K of (G, F) such that*

$$|K| \leq q \cdot \frac{\big(q(d^2+1)\big)^d - 1}{q(d^2+1) - 1}.$$

Proof. Consider the following recursive marking procedure $\mathtt{recCore}(q, u)$, which can be applied to any vertex $u \in V(G)$. For each $X \in \binom{\mathsf{tail}_F[u]}{\leq 2}$, among vertices $w \in \mathsf{chld}_F(u)$ that satisfy $X \subseteq \mathsf{SReach}_F(w)$ mark q with the highest value of $\mathsf{height}(F_w)$, or all of them if their number is smaller than q. Note that the total number of vertices marked in this way is bounded by $q \cdot |\binom{\mathsf{tail}_F[u]}{\leq 2}| \leq q(d^2+1)$. Finally, apply the procedure $\mathtt{recCore}(q, w)$ recursively for every marked child w of u.

Now, let $R \subseteq \mathsf{roots}_F$ comprise q roots r of F with the highest value of $\mathsf{height}(F_r)$, or all of them if their number is smaller than q. We apply procedure $\mathtt{recCore}(q, r)$ to all $r \in R$, and let K be the set comprising all the vertices marked this way. Clearly, for every $i \in \{1, \ldots, d\}$, K contains at most $q \cdot \big(q(d^2+1)\big)^{i-1}$ vertices at depth i in F, hence

$$|K| \leq q \cdot \sum_{i=1}^{d} \big(q(d^2+1)\big)^{i-1} = q \cdot \frac{\big(q(d^2+1)\big)^d - 1}{q(d^2+1) - 1},$$

as claimed. That K is indeed a q-core of (G, F) follows directly from the construction. $\square$

In subsequent lemmas we will establish several important properties of cores. The following notation will be convenient: if K is a prefix in an elimination forest F of G, then for each $u \in \mathsf{App}_F(K)$ and $X \in \binom{\mathsf{SReach}_{F,G}(u)}{\leq 2}$, we define $W_K(u, X)$ to be the set of all siblings w of u in F such that:

- $w \in K$;
- $X \subseteq \mathsf{SReach}_{F,G}(w)$; and
- $\mathsf{height}(F_w) \geq \mathsf{height}(F_u)$.

Then, provided that K is a q-core of (G, F), we have $|W_K(u, X)| \geq q$ for all $u \in \mathsf{App}(K)$ and $X \in \binom{\mathsf{SReach}_{F,G}(u)}{\leq 2}$. Note that the definition of $W_K(\cdot, \cdot)$ depends on F and G; these will always be clear from the context, as K will be a core with respect to some pair (G, F).

As mentioned, the intuition is that a q-core for a sufficiently large q stores all the essential information about the graph needed for the purpose of computing its treedepth. We now formalize this intuition in a series of lemmas. First, we observe that cores retain the essential connectivity property from Definition 4.1.2.

Lemma 4.2.3. *Suppose that F is a recursively connected elimination forest of a graph G and K is a 1-core of (G, F). Then for every $u \in K$, we have that*
(i) $\mathsf{SReach}_{F,G}(u) = N_{G[K]}(K \cap \mathsf{tree}_F[u])$ *and*
(ii) the graph $G[K \cap \mathsf{tree}_F[u]]$ *is connected.*

Proof. We first prove (i) by induction on $\mathsf{height}(F_u)$. The base case when $\mathsf{height}(F_u) = 1$ is trivial: then $K \cap \mathsf{tree}_F[u] = \mathsf{tree}_F[u] = \{u\}$ and $\mathsf{tail}_F[u] \subseteq K$, hence $\mathsf{SReach}_{F,G}(u) = N_G(u) = N_{G[K]}(u) = N_{G[K]}(K \cap \mathsf{tree}_F[u])$.

Suppose now that $\mathsf{height}(F_u) > 1$. The inclusion $\mathsf{SReach}_{F,G}(u) \supseteq N_{G[K]}(K \cap \mathsf{tree}_F[u])$ is obvious, so it remains to show the following: if some $v \in \mathsf{tail}_F[u]$, $v \neq u$, has a neighbor in $\mathsf{tree}_F[u]$, then v has also a neighbor in $K \cap \mathsf{tree}_F[u]$. For this, let a be a neighbor of v in $\mathsf{tree}_F[u]$. If $a = u$ then we are done, because $u \in K$. Otherwise $a \in \mathsf{tree}_F[w]$ for some $w \in \mathsf{chld}_F(u)$, which implies that $v \in \mathsf{SReach}_{F,G}(w)$. As K is a 1-core in F, there exists $w' \in \mathsf{chld}_F(u) \cap K$ such that $v \in \mathsf{SReach}_{F,G}(w')$. By applying the induction hypothesis to w' we infer that $\mathsf{SReach}_{F,G}(w') = N_{G[K]}(K \cap \mathsf{tree}_F[w'])$, hence v has a neighbor a' in $K \cap \mathsf{tree}_F[w']$. Then also $a' \in K \cap \mathsf{tree}_F[u]$; this completes the proof of (i).

We now move to the proof of (ii), which we again perform by induction on $\mathsf{height}(F_u)$. As before, the base case when $\mathsf{height}(F_u) = 1$ is trivial, as then $G[K \cap \mathsf{tree}_F[u]]$ consists of one vertex u. So suppose that $\mathsf{height}(F_u) > 1$. Since F is recursively connected, by Lemma 4.1.3 u has a neighbor in each of the sets $\mathsf{tree}_F[w]$ for $w \in \mathsf{chld}_F(u)$. Consider any $w \in \mathsf{chld}_F(u)$ such that $K \cap \mathsf{tree}_F[w] \neq \emptyset$. Since K is a prefix of F, we have $w \in K$. By applying (i) and the induction hypothesis to w, we infer that u has a neighbor in $K \cap \mathsf{tree}_F[w]$ and $G[K \cap \mathsf{tree}_F[w]]$ is connected. Thus, $G[K \cap \mathsf{tree}_F[u]]$ consists of a disjoint union of connected graphs, plus there is vertex u which has neighbors in each of these connected graphs. Hence $G[K \cap \mathsf{tree}_F[u]]$ is connected as well. This proves (ii). $\square$

We now present the following lemma, which intuitively provides good "re-attachment points" for trees obtained by removing a core from an elimination forest.

Lemma 4.2.4. *Let F be a recursively connected elimination forest of a graph G such that F has height at most d. Let K be a d-core of (G, F). Let F^K be any elimination forest of $G[K]$ of height at most d. Then for every $u \in \mathsf{App}_F(K)$, the set $\mathsf{SReach}_{F,G}(u)$ is straight in F^K.*

Proof. Consider any pair of distinct vertices $x, y \in \mathsf{SReach}_{F,G}(u)$. Denote $W := W_K(u, \{x, y\})$; then $|W| \geq d$. Consider any $w \in W$; recall that $w \in K$ and $x, y \in \mathsf{SReach}_{F,G}(w)$. By Lemma 4.2.3, the graph $G[K \cap \mathsf{tree}_F[w]]$ is connected and both x and y have neighbors in $K \cap \mathsf{tree}_F[w]$ in $G[K]$. This implies that in $G[K]$ there is a path P^{xy}_w with endpoints x and y such that every internal vertex of P^{xy}_w belongs to $\mathsf{tree}_F[w]$. Since the sets in $\{\mathsf{tree}_F[w] \colon w \in W\}$ are pairwise disjoint, this shows that x and y are d-vertex-connected in $G[K]$. By Lemma 4.1.5 we infer that $\{x, y\}$ is straight in the elimination forest F^K. Since x and y were chosen arbitrarily from $\mathsf{SReach}_{F,G}(u)$, we conclude that $\mathsf{SReach}_{F,G}(u)$ is straight in F^K, as claimed. $\square$

From Lemma 4.2.4 we can derive the following claim: restricting an elimination forest to a d-core preserves the treedepth of each subgraph induced by a subtree.

Lemma 4.2.5. *Let F be a recursively optimal elimination forest of a graph G such that F is of height at most d, and let K be a d-core of (G, F). Then for every $v \in K$, we have* $\mathrm{td}(G[K \cap \mathsf{tree}_F[v]]) = \mathrm{td}(G[\mathsf{tree}_F[v]])$.

Proof. Observe that for every $v \in K$, we have that F_v is a recursively optimal elimination tree of $G[\mathsf{tree}_F[v]]$ and $K \cap V(F_v)$ is a d-core for $(G[\mathsf{tree}_F[v]], F_v)$. Hence, it suffices to prove the lemma for the case when F is a tree and v is the root of F. Indeed, if we succeed in this, then applying the statement for this case to each $v \in K$ yields the general statement of the lemma.

We proceed by induction on $\mathsf{height}(F)$. The base case where $\mathsf{height}(F) = 1$ is trivial. For the inductive step, we may assume that

$$\mathrm{td}(G[K \cap \mathsf{tree}_F[z]]) = \mathrm{td}(G[\mathsf{tree}_F[z]]) \qquad \text{for each } z \in K \setminus \{v\}. \tag{4.1}$$

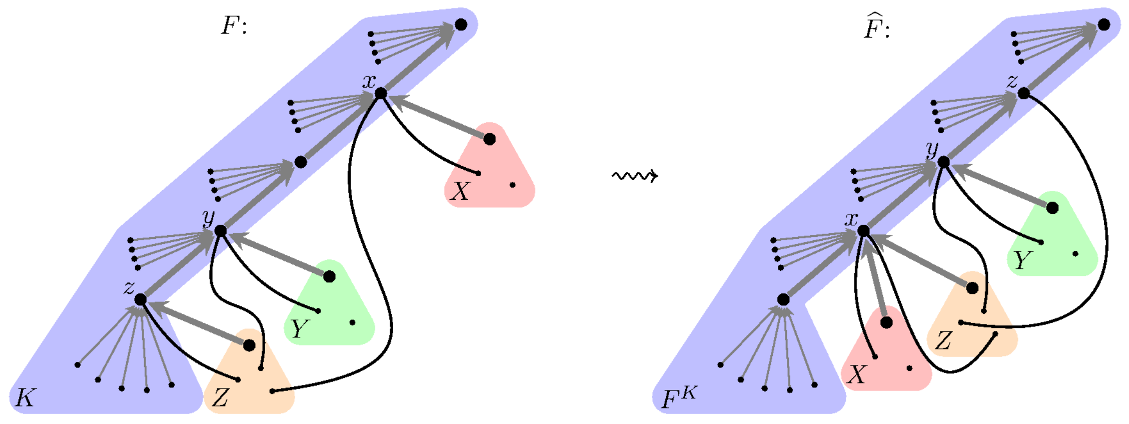

Figure 4.2: The reattachment procedure carried out in Lemma 4.2.5. Left: A recursively optimal elimination tree F of the graph G and a core K of F. Edges of F are gray and edges of G are black. X, Y and Z indicate the subtrees of F that are disjoint from K. The edges of G drawn between K and X, Y and Z indicate the sets $\mathsf{SReach}_{F,G}$ of the roots of X, Y, and Z. Right: An elimination tree $\widehat{F}$ constructed from G, F and K according to Lemma 4.2.5. We take a recursively optimal elimination tree F^K for $G[K]$ and reattach each subtree X, Y, and Z at the deepest vertex of the set $\mathsf{SReach}_{F,G}$ of the corresponding root of X, Y or Z.

We need to prove that $\mathrm{td}(G[K]) = \mathrm{td}(G)$. Clearly $\mathrm{td}(G[K]) \le \mathrm{td}(G)$, so for contradiction suppose that $\mathrm{td}(G[K]) < \mathrm{td}(G)$. Let F^K be an elimination forest of $G[K]$ of height strictly smaller than $\mathrm{td}(G)$; in particular, $\mathsf{height}(F^K) < d$. Observe that, thus, $V(G) \setminus K \neq \emptyset$. Our goal is to construct an elimination forest $\widehat{F}$ of G of height equal to the height of F^K, which will be a contradiction.

Consider any $u \in \mathsf{App}_F(K)$. By Lemma 4.2.4, the set $M \coloneqq \mathsf{SReach}_{F,G}(u)$ is straight in F^K. Note here that since $u \neq v$ (due to $u \notin K$ and $v \in K$) and v is the only root of F, u has a parent in F, which by Lemma 4.1.3 belongs to M. In particular $M \neq \emptyset$. Let then m be the vertex of M that is the deepest in F^K; this is well-defined because M is straight in F^K. Further, let $\widehat{M} \coloneqq \mathsf{tail}_{F^K}[m]$; as $\mathsf{height}(F^K) < d$ and M is straight in F^K, we have $M \subseteq \widehat{M}$ and $|\widehat{M}| < d$.

Let $W \coloneqq W_K(u, \{m\})$; then $|W| \ge d$ as K is a d-core. Since $|W| \ge d$, there exists $w \in W$ such that $\mathsf{tree}_F[w] \cap \widehat{M} = \emptyset$. Recall that $m \in \mathsf{SReach}_{F,G}(w)$ and $\mathsf{height}(F_w) \ge \mathsf{height}(F_u)$ by the definition of W.

By Lemma 4.2.3, graph $G[K \cap \mathsf{tree}_F[w]]$ is connected and $N_{G[K]}(K \cap \mathsf{tree}_F[w]) = \mathsf{SReach}_{F,G}(w)$. As $G[K \cap \mathsf{tree}_F[w]]$ is connected and F^K is an elimination forest of $G[K]$, by Lemma 4.1.4 there exists $x \in K \cap \mathsf{tree}_F[w]$ such that $K \cap \mathsf{tree}_F[w] \subseteq \mathsf{tree}_{F^K}[x]$. Note that $x \notin \widehat{M}$ because $\mathsf{tree}_F[w] \cap \widehat{M} = \emptyset$. On the other hand, since $m \in \mathsf{SReach}_{F,G}(w) = N_{G[K]}(K \cap \mathsf{tree}_F[w])$, m has a neighbor in the set $K \cap \mathsf{tree}_F[w] \subseteq \mathsf{tree}_{F^K}[x]$. As F^K is an elimination forest of $G[K]$, this implies that $x \in \mathsf{tree}_{F^K}(m)$. Since $K \cap \mathsf{tree}_F[w] \subseteq \mathsf{tree}_{F^K}[x]$ and $M \subseteq \mathsf{tail}_{F^K}[m]$, we conclude that in F^K, all the vertices of $K \cap \mathsf{tree}_F[w]$ are descendants of all the vertices of M, hence also of all the vertices of $\widehat{M}$. In particular,

$$|\widehat{M}| + \mathrm{td}(G[K \cap \mathsf{tree}_F[w]]) \le \mathsf{height}(F^K).$$

Since $w \neq v$, applying (4.1) with $z = w$ yields $\mathrm{td}(G[K \cap \mathsf{tree}_F[w]]) = \mathrm{td}(G[\mathsf{tree}_F[w]])$. Moreover, observe that we have $\mathrm{td}(G[\mathsf{tree}_F[w]]) = \mathsf{height}(F_w) \ge \mathsf{height}(F_u) = \mathrm{td}(G[\mathsf{tree}_F[u]])$, where the equalities follow from the recursive optimality of F. By combining these, we find that

$$|\widehat{M}| + \mathsf{height}(F_u) \le \mathsf{height}(F^K). \tag{4.2}$$

We now construct a new elimination forest $\widehat{F}$ of G as follows. See Figure 4.2 for an illustration. Begin with $\widehat{F} = F^K$ and, for every $u \in \mathsf{App}_F(K)$, insert the tree F_u into $\widehat{F}$ by making u a child of m, defined as the vertex of $\mathsf{SReach}_{F,G}(u)$ that is deepest in F^K (this vertex is well-defined since $\mathsf{SReach}_{F,G}(u)$ is straight in F^K). It is straightforward to see that $\widehat{F}$ constructed this way is an elimination forest of G. Moreover, by (4.2) we conclude that $\mathsf{height}(\widehat{F}) = \mathsf{height}(F^K)$ (recall that $M \subseteq \widehat{M}$). As $\mathsf{height}(F^K) < \mathrm{td}(G)$, this is a contradiction, and the inductive step is proved. □

4.3 Obstructions for treedepth

In this section we consider bounds on the sizes of induced subgraphs that are obstructions for having low treedepth, as explained formally through the following notion.

Definition 4.3.1. A graph G is a *minimal obstruction for treedepth d* if $\mathrm{td}(G) > d$, but $\mathrm{td}(G - v) \leq d$ for each $v \in V(G)$.

As mentioned earlier, Dvořák et al. [31] proved the following result.

Theorem 4.3.2 ([31])**.** *Let $d \in \mathbb{N}$. Then every minimal obstruction for treedepth d has at most $2^{2^{d-1}}$ vertices. Furthermore, there exists a minimal obstruction for treedepth d that has 2^d vertices.*

In fact, the lower bound of 2^d provided by Theorem 4.3.2 is obtained by showing that every acyclic minimal obstruction for treedepth d has exactly 2^d vertices, and such obstructions can be precisely characterized by means of an inductive construction. This led Dvořák et al. [31] to conjecture that in fact every minimal obstruction for treedepth d has at most 2^d vertices. We now show that from the combinatorial analysis presented in the previous section we can derive an upper bound with asymptotic growth $d^{\mathcal{O}(d)}$. While this still leaves a gap to the conjectured value of 2^d, the new estimate is dramatically lower than the doubly-exponential upper bound provided in [31].

Let us recall Theorem 1.1.9.

Theorem 1.1.9. If G is a minimal obstruction for treedepth d, then the vertex count of G is at most

$$(d+1) \cdot \frac{\big((d+1)((d+1)^2+1)\big)^{d+1} - 1}{(d+1)((d+1)^2+1) - 1} \in d^{\mathcal{O}(d)}.$$

Proof. Since G is a minimal obstruction for treedepth d, G is connected and $\mathrm{td}(G) = d+1$. Let F be a recursively optimal elimination tree of G; then $\mathsf{height}(F) = d+1$. Let r be the root of F. By Lemma 4.2.2, we can find a $(d+1)$-core K of (G, F) of size at most $M(d)$, where $M(d)$ is the bound provided in the theorem statement. Clearly, $r \in K$. Applying Lemma 4.2.5 to $v = r$, we find that $\mathrm{td}(G[K]) = \mathrm{td}(G)$. Since G is a minimal obstruction for treedepth d, this means that $K = V(G)$, implying $|V(G)| \leq M(d)$. □

We remark that a more careful analysis of the bounds used in Lemma 4.2.2 yields a slightly better upper bound than the one claimed in Theorem 1.1.9, however with the same asymptotic growth of $d^{\mathcal{O}(d)}$. It remains open whether this can be improved to an upper bound of the form $2^{\mathcal{O}(d)}$.

Chapter 5

Computing treedepth in polynomial space and linear FPT time

In this Chapter we will present algorithms computing optimal treedepth decomposition of a graph in polynomial space. More precisely, we are going to prove Theorem 1.1.10.

Theorem 1.1.10. *There is an algorithm that given an n-vertex graph G and an integer d, either constructs an elimination forest of G of depth at most d, or concludes that the treedepth of G is larger than d. The algorithm runs in $2^{\mathcal{O}(d^2)} \cdot n^{\mathcal{O}(1)}$ time and uses $n^{\mathcal{O}(1)}$ space.*

The space and time complexities can be improved to $d^{\mathcal{O}(1)} \cdot n$ and expected $2^{\mathcal{O}(d^2)} \cdot n$, respectively, at the cost of allowing randomization: the algorithm may return a false negative with probability at most $\frac{1}{c \cdot n^c}$, where c is any constant fixed a priori; there are no false positives.

As mentioned in the statement, this algorithm comes in two versions — deterministic and randomized ones. First, we are going to describe the deterministic version and then we are going to show how to improve it through the usage of randomization.

5.1 Deterministic FPT algorithm

In this section we prove the first part of Theorem 1.1.10: we give a deterministic polynomial-space algorithm with running time $2^{\mathcal{O}(d^2)} \cdot n^{\mathcal{O}(1)}$ that for a given n-vertex graph G, either outputs an elimination forest of G of depth at most d or concludes that no such forest exists. The most complex part of the algorithm will be procedure `CountElimTrees`, which, roughly speaking, counts the number of different elimination trees of a connected graph G of depth at most d. We describe `CountElimTrees` first, and then we utilize it to achieve the main result.

5.1.1 Description of `CountElimTrees`

As mentioned above, procedure `CountElimTrees` counts the number of different elimination trees of G of depth at most d. However, we will not count all of them, but only such that are in some sense minimal; a precise formulation will follow later. We remark that this part is inspired by the $3^d \cdot n^{\mathcal{O}(1)}$-time polynomial space algorithm of Pilipczuk and Wrochna [73] for counting dominating sets in a graph of bounded treedepth. This algorithm exploits the same underlying trick — sometimes dubbed "inclusion-exclusion branching" — but the application here is technically more involved than in [73].

Before describing `CountElimTrees`, let us carefully define objects that we are going to count. We start by recalling the following standard fact about the existence of elimination forests with basic connectivity properties.

Lemma 5.1.1. *Let H be a graph and let R be an elimination forest of H. Then there exists an elimination forest R' of H such that*

- *for every vertex u of H, we have $\mathsf{depth}_{R'}(u) \le \mathsf{depth}_R(u)$; and*
- *whenever vertices $u, v \in V(H)$ belong to the same connected component of R', they also belong to the same connected component of H.*

Proof. For every connected component C of H, let R_C be the rooted tree on vertex set $V(C)$ with the ancestor relation inherited from R: for $u, v \in V(C)$, u is an ancestor of v in R_C if and only if u is an ancestor of v in R. Let R' be the disjoint union of trees R_C over all connected components C of H. It is straightforward to check that R' constructed in this way is an elimination forest of H that satisfies both the asserted properties. □

We remark that computing R' can be easily done in linear time by using depth-first search from the root of each elimination tree in R. This procedure will be used many times throughout the algorithm when justifying the usual assumption that our current graph is connected. (Disconnected graphs will often naturally appear when recursing after performing some deletions in the original graph.)

The following lemma can be proved using a very similar, though a bit more involved reasoning. Recall that we work with a fixed connected graph G and its elimination tree T.

Lemma 5.1.2. *Let G be a connected graph of treedepth at most d and T be an elimination tree of G (possibly of depth larger than d). Then there exists an elimination tree R of G of depth at most d that satisfies the following property: for every $u \in V(G)$ and $v_1, v_2 \in \mathsf{chld}_T(u)$, $v_1 \neq v_2$, we have*

$$\mathsf{cl}_R(\mathsf{comp}_T[v_1]) \cap \mathsf{cl}_R(\mathsf{comp}_T[v_2]) = \mathsf{cl}_R(\mathsf{tail}_T[u]). \tag{5.1}$$

Proof. Let R be an elimination tree of G of depth at most d that minimizes $\sum_{u \in V(G)} \mathsf{depth}_R(u)$. We claim that R satisfies the required property. Assume otherwise: there are $u \in V(G)$ and distinct $v_1, v_2 \in \mathsf{chld}_T(u)$ such that (5.1) does not hold. Since the $\mathsf{cl}_R(\cdot)$ operator is monotone under taking subsets, we have

$$\mathsf{cl}_R(\mathsf{comp}_T[v_1]) \cap \mathsf{cl}_R(\mathsf{comp}_T[v_2]) \supseteq \mathsf{cl}_R(\mathsf{tail}_T[u]) \cap \mathsf{cl}_R(\mathsf{tail}_T[u]) = \mathsf{cl}_R(\mathsf{tail}_T[u]).$$

So there is $r \in V(G)$ such that $r \notin \mathsf{cl}_R(\mathsf{tail}_T[u])$, but $r \in \mathsf{cl}_R(\mathsf{comp}_T[v_1])$ and $r \in \mathsf{cl}_R(\mathsf{comp}_T[v_2])$.

Note that since $r \in \mathsf{cl}_R(\mathsf{comp}_T[v_1]) \setminus \mathsf{cl}_R(\mathsf{tail}_T[u])$, we have $r \in \mathsf{cl}_R(\mathsf{tree}_T[v_1])$. So there exists $u_1 \in \mathsf{tree}_T[v_1]$ such that $u_1 \in \mathsf{tree}_R[r]$. Similarly, there exists $u_2 \in \mathsf{tree}_T[v_2]$ such that $u_2 \in \mathsf{tree}_R[r]$. As $r \notin \mathsf{cl}_R(\mathsf{tail}_T[u])$, we have $\mathsf{tail}_T[u] \cap \mathsf{tree}_R[r] = \emptyset$.

Observe that since T is an elimination tree of G, we have $N_G[\mathsf{tree}_T[v_1]] \subseteq \mathsf{comp}_T[v_1]$. This implies that $N_{G \setminus \mathsf{tail}_T[u]}[\mathsf{tree}_T[v_1]] \subseteq \mathsf{tree}_T[v_1]$; analogously $N_{G \setminus \mathsf{tail}_T[u]}[\mathsf{tree}_T[v_2]] \subseteq \mathsf{tree}_T[v_2]$. So in $G \setminus \mathsf{tail}_T[u]$, each of the sets $\mathsf{tree}_T[v_1]$ and $\mathsf{tree}_T[v_2]$ is the union of vertex sets of a collection of connected components. Note that $\mathsf{tree}_T[v_1] \cap \mathsf{tree}_T[v_2] = \emptyset$.

Consider now the graph $H = G[\mathsf{tree}_R[r]]$. As $\mathsf{tail}_T[u] \cap \mathsf{tree}_R[r] = \emptyset$, H is an induced subgraph of $G \setminus \mathsf{tail}_T[u]$. Further, H intersects both $\mathsf{tree}_T[v_1]$ and $\mathsf{tree}_T[v_2]$, namely $u_1 \in V(H) \cap \mathsf{tree}_T[v_1]$ and $u_2 \in V(H) \cap \mathsf{tree}_T[v_2]$. Then the conclusion of the previous paragraph implies that H is disconnected.

Let R_H be an elimination forest of H obtained by applying Lemma 5.1.1 to H and its elimination tree inherited from R and rooted at r. Let R' be the elimination tree of G obtained from R by first removing all vertices of $V(H) = \mathsf{tree}_R[r]$, and then reintroducing them again by adding forest R_H and making all roots of R_H into children of the parent of r in R. Note that r is not the root of R, since G is connected. It is straightforward to check that R' is still an elimination tree of G, and from Lemma 5.1.1 it follows that $\mathsf{depth}_{R'}(w) \le \mathsf{depth}_R(w)$ for each $w \in V(G)$. However, since $R[\mathsf{tree}_R[r]]$ has only one root — r — while R_H has at least two roots — due to H being disconnected — it follows that $\mathsf{depth}_{R'}(w) < \mathsf{depth}_R(w)$ for at least one $w \in V(H)$. So R' is an elimination tree of G of depth at most d in which the sum of depths of vertices is strictly smaller than in R. This is a contradiction with the choice of R. □

An elimination tree R of a graph G satisfying the conclusion of Lemma 5.1.2 (that is, the depth of R is at most d and for all $u \in V(G)$ and distinct $v_1, v_2 \in \mathsf{chld}_T(u)$ we have (5.1)) will be called *sensible* with respect to T. In our search for elimination trees of low depth, we will restrict attention only to trees that are sensible with respect to some fixed elimination tree T. Then Lemma 5.1.2 justifies that we may do this without losing all solutions.

With all ingredients introduced, we may finally precisely state the goal of this section.

Lemma 5.1.3. *There exists an algorithm* `CountElimTrees`(G, T, d) *that, given a connected graph G on n vertices, an elimination tree T of depth k, and an integer d, runs in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$, uses $n^{\mathcal{O}(1)}$ space, and outputs the number of different elimination trees of G of depth at most d that are sensible with respect to T.*

Note here that the input to `CountElimTrees` consists not only of G and d, but also of an auxiliary elimination tree T of G. The depth k of T may be, and typically will be, larger than d. Also, we assume that an elimination tree is represented solely by its vertex set and the ancestor relation. In particular, permuting children of a vertex yields the *same* elimination tree, which should be counted as the same object by procedure `CountElimTrees`.

The remainder of this section is devoted to the proof of Lemma 5.1.3. We first need to introduce some definition.

Let us arbitrarily enumerate the vertices of G as $v_1, v_2, \ldots, v_n$ in a top-down manner in T. That is, whenever v_i is an ancestor of v_j, we have $i \leq j$. Consider another rooted tree R and a mapping $\phi \colon V(T) \to V(R)$. For a vertex u of T, we call a vertex $v_i \in \mathsf{tree}_T(u)$ a *proper surplus image* (for u and (R, ϕ)) if at least one of the following conditions holds:

- $\phi(v_i) \in \mathsf{cl}_R(\phi(\mathsf{tail}_T[u]))$, or
- there exists j such that $j < i$, $v_j \in \mathsf{tree}_T(u)$, and $\phi(v_j) = \phi(v_i)$.

We define *non-proper surplus images* analogously, but using sets $\mathsf{tail}_T(u)$ and $\mathsf{tree}_T[u]$ instead of $\mathsf{tail}_T[u]$ and $\mathsf{tree}_T(u)$, respectively.

We will work in the ring of polynomials $\mathbb{Z}[x]$, where x is a formal variable. By an abuse of notation, we equip this ring with an operation of division by x defined through equations:

$$\frac{x^i}{x} = \begin{cases} x^{i-1} & \text{if } i \geq 1, \\ 0 & \text{if } i = 0 \end{cases}$$

$$\frac{\alpha A + \beta B}{x} = \alpha \cdot \frac{A}{x} + \beta \cdot \frac{B}{x} \qquad \text{for all} \qquad A, B \in \mathbb{Z}[x] \text{ and } \alpha, \beta \in \mathbb{Z}.$$

Formally speaking, division by x is just the unique function from $\mathbb{Z}[x]$ to $\mathbb{Z}[x]$ satisfying the two properties above.

Even though our final goal is to count the number of elimination trees, along the way we are going to count more general objects, called *generalized elimination trees*. A generalized elimination tree of a graph H is a rooted tree R along with a mapping $\phi \colon V(H) \to V(R)$ such that ϕ respects edges. Note that in particular, it may be the case that $\mathrm{im}(\phi) \subsetneq V(R)$ or that $\phi(u) = \phi(v)$ for some $u, v \in V(H)$. Clearly, a generalized elimination tree is an elimination tree in the usual sense if and only if ϕ is a bijection between $V(H)$ and $V(R)$. We shall call two generalized elimination trees (R, ϕ) and (R', ϕ') *isomorphic* if there is an isomorphism of rooted trees ψ mapping R to R' such that $\phi' = \psi \circ \phi$.

A generalized elimination tree (R, ϕ) of an induced subgraph H of G is *sensible* for T if for every $u \in V(H)$ and distinct $v_1, v_2 \in \mathsf{chld}_T(u) \cap V(H)$, we have $\mathsf{cl}_R(\phi(\mathsf{comp}_T[v_1])) \cap \mathsf{cl}_R(\phi(\mathsf{comp}_T[v_2])) = \mathsf{cl}_R(\phi(\mathsf{tail}_T[u]))$. Thus, this notion projects to sensibility of (standard) elimination trees when $H = G$ and (R, ϕ) is an elimination tree of G. Generalized elimination trees of induced subgraphs of G that are sensible for T shall be called *monsters*.

For a rooted tree K, a mapping ϕ with co-domain $V(K)$ is called a *cover* of K if $\mathsf{cl}_K(\mathrm{im}(\phi)) = V(K)$, or equivalently, every leaf of K is in the image of ϕ. For a vertex $u \in V(G)$, rooted tree K of depth at most d, a subset of vertices $A \subseteq V(K)$ that contains all leaves of K, and a mapping $\phi \colon \mathsf{tail}_T(u) \to A$ that is a cover of K, we define

$$f(u,K,\phi,A) = \sum_{i=0}^{n} a_i x^i \in \mathbb{Z}[x],$$

where a_i is the number of non-isomorphic monsters $(R,\overline{\phi})$ such that:

- $(R,\overline{\phi})$ is a generalized elimination tree of $G[\mathsf{comp}_T[u]]$ of depth at most d;
- K is a prefix of R;
- $\overline{\phi}$ is an extension of ϕ satisfying

$$V(R)\setminus V(K) \subseteq \mathrm{im}(\overline{\phi}) \subseteq (V(R)\setminus V(K)) \cup A; \qquad \text{and}$$

- in $\mathsf{tree}_T[u]$ there are exactly i non-proper surplus images for u and $(R,\overline{\phi})$.

Note that since ϕ is assumed to be a cover of K, and by the second and third condition, the last condition can be rephrased as follows:

$$i = |\mathsf{tree}_T[u]| - |V(R)\setminus V(K)|.$$

We define polynomial $g(u,K,\phi,L)$ analogously, but using $\mathsf{tail}_T[u]$, $\mathsf{tree}_T(u)$, and proper surplus images, instead of $\mathsf{tail}_T(u)$, $\mathsf{tree}_T[u]$ and non-proper surplus images. That in $\mathsf{tree}_T(u)$ there are i proper surplus images is then equivalent to $i = |\mathsf{tree}_T(u)| - |V(R)\setminus V(K)|$.

Our goal now is to compute the polynomials $f(\cdot,\cdot,\cdot,\cdot)$ and $g(\cdot,\cdot,\cdot,\cdot)$ recursively over the elimination tree T. It can be easily seen that if $\mathsf{chld}_T(u) = \emptyset$ then

$$g(u,K,\phi,A) = \begin{cases} 1 & \text{if } \phi \text{ respects edges,} \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

Indeed, $(R,\overline{\phi}) = (K,\phi)$ is the only possible pair that can satisfy the last three conditions, and it is a sensible generalized elimination tree of $G[\mathsf{comp}_T[u]]$ if and only if ϕ respects edges.

First, we show how to compute polynomials $g(u,\cdot,\cdot,\cdot)$ based on the knowledge of polynomials $f(v,\cdot,\cdot,\cdot)$ for children v of u.

Lemma 5.1.4. *If $\mathsf{chld}_T(u) \neq \emptyset$, then for all relevant u,K,ϕ,A we have*

$$g(u,K,\phi,A) = \prod_{v\in\mathsf{chld}_T(u)} f(v,K,\phi,A)$$

Proof. Let $\mathsf{chld}_T(u) = \{v_1,\ldots,v_c\}$ and let $(R_1,\overline{\phi}_1),\ldots,(R_c,\overline{\phi}_c)$ be any monsters such that $(R_i,\overline{\phi}_i)$ is a monster counted in the definition of $f(v_i,K,\phi,A)$. Note that K is a prefix of each R_i, and each $\overline{\phi}_i$ is an extension of ϕ. Therefore, we can construct a monster $(R,\overline{\phi})$ as follows:

- R is the union of $R_1,\ldots,R_c$ with the vertices of K identified naturally;
- $\overline{\phi}$ is the union of $\overline{\phi}_1,\ldots,\overline{\phi}_c$ (note that values on K match).

That $(R,\overline{\phi})$ constructed in this manner is sensible for T is easy to verify. Moreover, observe that every distinct tuple of monsters $(R_1,\overline{\phi}_1),\ldots,(R_c,\overline{\phi}_c)$ gives rise to a different (non-isomorphic) monster $(R,\overline{\phi})$.

On the other hand, we argue that every monster $(R,\overline{\phi})$ counted in the definition of $g(u,K,\phi,A)$ can be obtained from some monsters $(R_1,\overline{\phi}_1),\ldots,(R_c,\overline{\phi}_c)$ in the way described above. Indeed, $(R,\overline{\phi})$ is sensible for T, hence every subtree of $R-K$ accommodates images under $\overline{\phi}$ of vertices from only one subtree $\mathsf{tree}_T[v_i]$, for some $i \in \{1,\ldots,c\}$. Distributing the subtrees of $R-K$ according to the index i as above naturally gives rise to monsters $(R_1,\overline{\phi}_1),\ldots,(R_c,\overline{\phi}_c)$ that are counted in the definitions of $f(v_1,K,\phi,A),\ldots,f(v_c,K,\phi,A)$, respectively.

Altogether, we have shown that distinct tuples of monsters $(R_1,\overline{\phi}_1),\ldots,(R_c,\overline{\phi}_c)$ contributing to the definitions of $f(v_1,K,\phi,A),\ldots,f(v_c,K,\phi,A)$ are in one-to-one correspondence with monsters $(R,\overline{\phi})$ contributing to the definition of $g(u,K,\phi,A)$. This correspondence preserves the number of surplus vertices in the following sense: if for each $i \in \{1,\ldots,c\}$, $\mathsf{tree}_T[v_i]$ has j_i non-proper surplus images for v_i and $(R_i,\overline{\phi}_i)$, then $\mathsf{tree}_T(u)$ has $j_1+\ldots+j_c$ proper surplus images for u and $(R,\overline{\phi})$. This directly implies the postulated equality of polynomials. $\square$

Let us elaborate on the intuition on what happened in Lemma 5.1.2. Intuitively, we aggregated information about the children of u to the information about u itself. Since in the definitions of monsters we do not insist on the mappings being injective, this aggregation could have been performed by a simple product of polynomials (though, the assumption of sensibility was crucial for arguing the correctness). In a natural dynamic programming, such as the one in [76], one would need to ensure injectivity when aggregating information from the children of u, which would result in a dynamic programming procedure that would need to keep track of all subsets of K (and thus use exponential space). Thus, relaxing injectivity here allows us to use simple multiplication of polynomials, but obviously we will eventually need to enforce injectivity. The idea is that we enforce surjectivity instead, and make sure that the size of the co-domain matches the size of the domain. In turn, surjectivity is enforced using inclusion-exclusion in the computation of polynomials $f(u,\cdot,\cdot,\cdot)$ based on polynomials $g(u,\cdot,\cdot,\cdot)$, which is the subject of the next lemma.

Lemma 5.1.5. *For all relevant u, K, ϕ, A, we have:*

$$\begin{aligned} f(u,K,\phi,A) = & \sum_{v\in A} x\cdot g(u,K,\phi[u\to v],A) + \\ & \sum_{w\in K} \sum_{p=1}^{d-\mathsf{depth}(w)} \frac{1}{x^{p-1}} \\ & \sum_{B\subseteq\{w_1,\ldots,w_{p-1}\}} (-1)^{p-1-|B|} g(u,K[w,w_1,\ldots,w_p],\phi[u\to w_p],A\cup B\cup\{w_p\}), \end{aligned}$$

where $K[w,w_1,\ldots,w_p]$ denotes the rooted tree obtained from K by adding a path $[w,w_1,\ldots,w_p]$ so that w is the parent of w_1 and each w_i is the parent of w_{i+1}, for $i\in\{1,\ldots,p-1\}$.

Proof. Let $(R,\overline{\phi})$ be a monster counted in the definition of $f(u,K,\phi,A)$. Observe that u is in the domain of $\overline{\phi}$, but not in the domain of ϕ. The intuition is that extending ϕ by mapping u to $\overline{\phi}(u)$ yields an object that is indirectly taken into account in the polynomials $g(u,\cdot,\cdot,\cdot)$, but we need to be careful that we express the contribution of $(R,\overline{\phi})$ to $f(u,K,\phi,A)$ as a combination of contributions of different monsters to different polynomials $g(u,\cdot,\cdot,\cdot)$. Let $v=\overline{\phi}(u)$.

Consider first the case when $v\in V(K)$. Note that then we necessarily have $v\in A$. Then $(R,\overline{\phi})$ is a monster that is counted in the definition of $g(u,K,\phi[u\to v],A)$. Observe that u is a non-proper surplus image for u and $(R,\overline{\phi})$, but it is not a proper surplus image for u and $(R,\overline{\phi})$, hence the number of proper surplus images for u and $(R,\overline{\phi})$ is exactly one larger than the number of non-proper surplus images for u and (R,ϕ). Also, every monster counted in the definition of $g(u,K,\phi[u\to v],A)$ contributes to $f(u,K,\phi,A)$ as above. This justifies the summand $\sum_{v\in A} x\cdot g(u,K,\phi[u\to v],A)$ in the formula.

Consider now the case when $v\in V(R)\setminus V(K)$. We need to consider various cases on how $\mathsf{cl}_R(\overline{\phi}(\mathsf{tail}[u]))$ differs from $\mathsf{cl}_R(\overline{\phi}(\mathsf{tail}(u)))$. The former can be described as the latter with a path attached, connecting v with the least ancestor w of v that belongs to $\mathsf{cl}_R(\overline{\phi}(\mathsf{tail}(u)))$. Let this path be $P=[w,w_1,\ldots,w_p]$, where $w_p=v$, and observe that the length of P, call it p, satisfies $p+\mathsf{depth}(w)\le d$. Therefore, if we denote $K'=K[w,w_1,\ldots,w_p]$, then it the case that $(R,\overline{\phi})$ is a monster that is counted in the definition of $g(u,K',\phi[u\to w_p],A\cup\{w_1,\ldots,w_p\})$. The problem is that not every monster counted in the definition of $g(u,K',\phi[u\to w_p],A\cup\{w_1,\ldots,w_p\})$ contributes to $f(u,K,\phi,A)$, because in the definition of the latter we require that $\overline{\phi}$ is surjective onto $V(R)\setminus V(K)$.

This issue is mitigated using the inclusion-exclusion principle. We iterate over all subsets $B\subseteq\{w_1,\ldots,w_{p-1}\}$ and take into account the contribution from $g(u,K',\phi[u\to w_p],A\cup B\cup\{w_p\})$ with sign $(-1)^{p-1-|B|}$. In this way, the only monsters that survive in the summation are those corresponding to monsters that are surjective onto $\{w_1,\ldots,w_{p-1}\}$.

Finally, we need to be careful about properly counting surplus images through the degrees of the formal variable x. As argued, the only summands that survive inclusion-exclusion summation are those corresponding to monsters $(R,\overline{\phi})$ where $\{w_1,\ldots,w_{p-1}\}\subseteq\mathrm{im}(\overline{\phi})$; so fix such a monster. For each $j\in\{1,\ldots,p-1\}$ there is the smallest index $s(j)$ such that $v_{s(j)}\in\mathsf{tree}(u)$ and $\overline{\phi}(v_{s(j)})=w_j$. Then $v_{s(j)}$ is a proper surplus image for u and $(R, \overline{\phi})$, but is not a non-proper surplus image for u and $(R, \overline{\phi})$. It is straightforward to check that all vertices of $\mathsf{tree}[u]$ except for $v_{s(1)}, \ldots, v_{s(p)}$ retain their status: they are a proper surplus image for u and $(R, \overline{\phi})$ if and only if they are a non-proper surplus image for $(R, \overline{\phi})$. Hence, there are exactly $p-1$ more proper surplus images for u and $(R, \overline{\phi})$ than there are non-proper surplus images for u and $(R, \overline{\phi})$. This justifies dividing the result of the inclusion-exclusion summation by x^{p-1} and concludes the proof. □

We need to take an additional care of how to deduce the overall number of elimination trees based on the polynomial $f(\cdot,\cdot,\cdot,\cdot)$ and $g(\cdot,\cdot,\cdot,\cdot)$. Define polynomial

$$h = \sum_{p=1}^{d} \frac{1}{x^{p-1}} \sum_{B \subseteq \{w_1, \ldots, w_{p-1}\}} (-1)^{p-1-|B|} g(r, [w_1, \ldots, w_p], [r \to w_p], B \cup \{w_p\}) \in \mathbb{Z}[x],$$

where r is the root of T, $[w_1, \ldots, w_p]$ is a path on p vertices rooted at w_1, and $[r \to w_p]$ denotes the function with domain $\{r\}$ that maps r to w_p.

Lemma 5.1.6. *The number of elimination trees of G that are sensible with respect to T and have depth at most d is the term in h standing by x^0.*

Proof. By Lemma 5.1.5, the formula can be seen as the formula for $f(r, K, \phi, A)$ for empty K, ϕ, and A. Therefore, h can be written as $h = \sum_{i=0}^{n} a_i x^i$, where a_i is the number of non-isomorphic sensible generalized elimination trees $(R, \overline{\phi})$ such that R has depth at most d, $\overline{\phi}\colon V(G) \to V(R)$ is surjective, and in G there are i non-proper surplus images for r and $(R, \overline{\phi})$. However, since K is empty, the number of surplus images is exactly the number of vertices $v_j \in V(G)$ that are mapped by $\overline{\phi}$ to the same vertex of R as some other vertex of G with a smaller index. Then the assertion that $\overline{\phi}$ is injective is equivalent to the assertion that the number of such surplus images is 0. It follows that the number of non-isomorphic sensible elimination trees of G of depth at most d is equal to the term in h that stands by x^0. □

Having established Lemmas 5.1.4, 5.1.5 and 5.1.6, we can conclude the description of procedure `CountElimTrees`. By 5.1.6, the goal is to compute polynomial h and return the coefficient standing by x^0. We initiate the computation using the formula for h, and then we use two mutually-recursive procedures to compute polynomials $f(\cdot,\cdot,\cdot,\cdot)$ and $g(\cdot,\cdot,\cdot,\cdot)$ using formulas provided by Lemmas 5.1.4 and 5.1.5. The base case of recursion is for a leaf of T, where we use formula (5.2).

The correctness of the procedure is established by Lemmas 5.1.4, 5.1.5 and 5.1.6. So it remains to bound its time complexity and memory usage. It is clear that polynomials that we compute will always have degrees at most n. Trees K relevant in the computation will never have more than dk vertices, for at every recursive call the tree K can grow by at most d new vertices.

As the next step, we bound the numbers that can be present in the computations.

Lemma 5.1.7. *Every coefficient of $f(u, K, \phi, A)$ is an integer from the range $[0, (dk \cdot 2^d)^{|\mathsf{tree}_T[u]|}]$ and every coefficient of $g(u, K, \phi, A)$ is an integer from the range $[0, (dk \cdot 2^d)^{|\mathsf{tree}_T(u)|}]$. Hence, all integers present in the computations are at most $(dk2^d)^n$.*

Proof. We prove this by induction on the recursion tree. The base of the induction (that is, calls of g on leaves) is clear. Induction step for g called on a vertex u that is not a leaf is clearly following from the bounds on f called on children of u as $|\mathsf{tree}_T(u)| = \sum_{v \in \mathsf{chld}_T(u)} |\mathsf{tree}_t[v]|$. Induction step for f called on a vertex u follows from the fact that it is a sum of at most $|A| + |K| \cdot (2^0 + 2^1 + \ldots + 2^{d-1}) \leq dk + dk \cdot (2^d - 1) = dk \cdot 2^d$ calls of g with coefficients from the set $\{-1, 1\}$ on the same vertex and the fact that $|\mathsf{tree}_T[u]| = |\mathsf{tree}_T(u)| + 1$. □

It follows that all integers present in the computation have bitsize bounded polynomially in n.

As for the memory usage, the run of the algorithm is a recursion of depth bounded by $2k$. The memory used is a stack of at most $2k$ frames for recursive calls of procedures computing polynomials $f(\cdot,\cdot,\cdot,\cdot)$ and $g(\cdot,\cdot,\cdot,\cdot)$ for relevant arguments. Each of these frames requires space polynomial in n, hence the total space complexity is polynomial in n.

As for the time complexity, each call to a procedure computing a polynomial of the form $f(u,\cdot,\cdot,\cdot)$ makes at most $dk \cdot 2^d$ recursive calls to procedures computing polynomials of the form $g(u,\cdot,\cdot,\cdot)$. In turn, each of these calls makes one call to a procedure computing a polynomial of the form $f(v,\cdot,\cdot,\cdot)$ for each child v of u. It follows that the total number of calls to procedures computing polynomials of the form $f(u,\cdot,\cdot,\cdot)$ and $g(u,\cdot,\cdot,\cdot)$ is bounded by $2 \cdot (dk \cdot 2^d)^k = 2^{\mathcal{O}(dk)}$. The internal work needed in each recursive call is bounded by $2^{\mathcal{O}(d)} \cdot n^{\mathcal{O}(1)}$. As T has n vertices, the total time complexity is $2^{\mathcal{O}(dk)} \cdot 2^{\mathcal{O}(d)} \cdot n^{\mathcal{O}(1)} \cdot n = 2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$, as claimed. This concludes the proof of Lemma 5.1.3.

We note that having designed `CountElimTrees`(G,T,d), it is easy to design a similar function `CountElimForest`(G,T,d) that does not need an assumption of G being connected and where T is some elimination forest instead of an elimination tree (by using the procedure described after Lemma 5.1.1).

5.1.2 Utilizing `CountElimTrees`

With the description of `CountElimTrees` completed, we can describe how we can utilize it in order to construct a bounded-depth elimination tree of a graph. That is, we prove the first part of Theorem 1.1.10.

First, we lift `CountElimTrees` to a constructive procedure that still requires to be provided an auxiliary elimination tree of the graph.

Lemma 5.1.8. *There is an algorithm* `ConstructElimForest`(G,T,d) *that, given an n-vertex graph G, an elimination forest T of G of depth at most k, and an integer d, runs in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$, uses $n^{\mathcal{O}(1)}$ space, and either correctly concludes that $\mathrm{td}(G) > d$ or returns an elimination forest of G of depth at most d.*

Proof. By treating every connected component separately, we may assume that G is connected (see the remark after Lemma 5.1.1). Thus T is an elimination tree of G.

The first step of `ConstructElimForest`(G,T,d) is calling `CountElimTrees`(G,T,d). If this call returns 0, we terminate `ConstructElimForest` and report that $\mathrm{td}(G) > d$; this is correct by Lemma 5.1.2. Otherwise we are sure that $\mathrm{td}(G) \leq d$, and we need to construct any elimination tree of depth at most d. In order to do so, we check, for every vertex $v \in V(G)$, whether v is a feasible candidate for the root of desired elimination tree. Note that a vertex v can be the root of an elimination tree of G of depth at most d if and only if $\mathrm{td}(G-v) < d$, or equivalently, if an only if the procedure `CountElimForest`$(G\backslash v, T-v, d-1)$ returns a positive value. (Here, by $T-v$ we mean the forest T with v removed and all former children of v made into children of the parent of v, or to roots in case v was a root.) As $\mathrm{td}(G) \leq d$, we know that for at least one vertex v, this check will return a positive outcome. Then we recursively call `ConstructElimForest`$(G-v, T-v, d-1)$, thus obtaining an elimination forest F' of $G-v$ of depth at most $d-1$, and we turn it into an elimination tree F of G by adding v as the new root and making it the parent of all the roots of F'. As F has depth at most d, it can be returned as the result of the procedure.

That the procedure is correct is clear. As for the time and space complexity, it is easy to see that there will be at most dn calls to the procedure `CountElimTrees` in total, because at each level of the recursion there will be at most one invocation of `CountElimTrees` per vertex of the original graph. As each of these calls uses $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ time and $n^{\mathcal{O}(1)}$ space, the same complexity bounds also follow for `ConstructElimForest`. ☐

It remains to show how to lift the assumption of being provided an auxiliary elimination forest of bounded depth. For this we use the iterative compression technique.

Proof of the first part of Theorem 1.1.10. Arbitrarily enumerate the vertices of G as $v_1, v_2, \ldots, v_n$. For $i \in \{1,\ldots,n\}$, let $G_i = G[\{v_1,\ldots,v_i\}]$ be the graph induced by the first i vertices. For each $i = 1,2,\ldots,n$ we will compute F_i, an elimination forest of G_i of depth at most d. For $i=1$ this is trivial. Assume now that we have already computed F_i and want to compute F_{i+1}. We first construct T_{i+1}, an elimination tree of G_{i+1}, by taking F_i, adding v_{i+1}, and making v_{i+1} the parent of all the roots of F_i. Note that T_{i+1} has depth at most $d+1$. We now call `ConstructElimForest`(G_{i+1}, T_{i+1}, d). If this procedure concludes that $\mathrm{td}(G_{i+1}) > d$, then this implies that $\mathrm{td}(G) > d$ as well, and we can terminate the algorithm and provide a negative answer. Otherwise, the procedure returns an elimination forest F_{i+1} of G_{i+1} of depth at most d, with which we can proceed. Eventually, the algorithm constructs an elimination forest $F = F_n$ of $G = G_n$ of depth at most d.

The algorithm is clearly correct. Since every call to `ConstructElimForest` is supplied with an elimination forest of depth at most $d + 1$, and there are at most n calls, the total time complexity is $2^{\mathcal{O}(d^2)} \cdot n^{\mathcal{O}(1)}$ and the space complexity is $n^{\mathcal{O}(1)}$, as desired. □

5.2 Randomized linear FPT algorithm

In this section we prove the second part of Theorem 1.1.10: we reduce the time and space complexities to linear in n at the cost of relying on randomization. There are three main reasons why the algorithm presented in the previous section does not run in time linear in n.

- First, in procedure `ConstructElimForest`, we applied `CountElimTrees` $\mathcal{O}(dn)$ times. Even if `CountElimTrees` runs in time linear in n, this gives at least a quadratic time complexity for `ConstructElimForest`.
- Second, in the iterative compression scheme we add vertices one by one and apply procedure `ConstructElimForest` n times. Again, even if `ConstructElimForest` runs in linear time, this gives at least a quadratic time complexity.
- Third, in procedure `CountElimTrees` we handle polynomials of degree at most n and with coefficients of bitsize bounded only polynomially in n. Algebraic operations on those need time polynomial in n.

In short, these obstacles are mitigated as follows:

- We give another implementation of `ConstructElimForest` that applies a modified variant of `CountElimTrees` only $d^{\mathcal{O}(d)}$ times. In essence, we sample a random coloring of the graph with $d^{\mathcal{O}(d)}$ colors, and for every color we apply a modification of `CountElimTrees` that is able to pinpoint a candidate for the root of an optimum-depth elimination forest in this color, provided there is exactly one. Since the total number of candidates in a connected graph of treedepth at most d is at most $d^{\mathcal{O}(d)}$ [18, 19], this procedure finds a candidate root with high probability.
- Iterative compression is replaced by a contraction scheme of Bodlaender [6] that allows us to replace iteration with recursion, where every recursive step reduces the total number of vertices by a constant fraction, rather than peels off just one vertex.
- We observe that in `CountElimTrees`, we may care only about monomials with degrees bounded by dk, so the degrees are not a problem. As for coefficients, we hash them modulo a sufficiently large prime. This is another source of randomization.

We proceed to formal details.

5.2.1 Optimizing the running time of `CountElimTrees`

We deal with monomials of high degree first.

Lemma 5.2.1. *The output of* `CountElimTrees`(G, T, d) *does not change if we use the quotient ring* $\mathbb{Z}[x]/(x^{dk})$ *instead of* $\mathbb{Z}[x]$.

Proof. Recall that the final output of `CountElimTrees` is the *free term* of the polynomial h, that is, the coefficient standing by x^0. The only division by x in the whole algorithm happens in the formula provided by Lemma 5.1.5, where we divide by x^{p-1}, where $p \leq d$. On any path of recursive calls in our algorithm, there are at most k calls of this type, hence the summands of form x^i for $i > k(d-1)$ will never have any contribution to the free term in the polynomial returned at the root of the recursion. Therefore, ignoring those summands does not affect the final result of the computation. □

Now, we optimize the cost of arithmetic operations. To this end, we use the standard technique of performing arithmetic operations modulo a random prime.

We start by recalling the following fact [73, Fact 29], which is based on [79, Theorem 4].

Fact 2. There is a positive integer L such that for all integers $\ell \geq L$ it holds that the product of primes strictly between ℓ and 2ℓ is larger than 2^ℓ.

Let $C \geq 1$ be a sufficiently large constant, to be specified later. Let

$$A = \max(L, n^5 2^{5Cd^2}).$$

Recall that by Lemma 5.1.7, the coefficients that appear during the computation of `CountElimTrees` are upper bounded $(dk2^d)^n$. However, due to modifications that will be explained later when we will speak about the weighted variant of procedure `CountElimTrees`, we will actually need to perform arithmetics on numbers as large as $(ndk2^d)^n$. As k in our applications is never larger than $2d$ and as $2d \leq 2^d$ for positive integers d, we have $(ndk2^d)^n \leq (n2^{3d})^n$. Positive integers that are at most that large cannot have more than n distinct prime factors in the interval $(A, 2A)$, as $A^n > (n2^{3d})^n$. However, by Fact 2, we know that there are at least $\frac{\log_2 2^A}{\log_2 2A} = \frac{A}{\log_2 A+1} = \Omega(n^4 2^{4Cd^2})$ primes in this interval. Since each non-zero number x that appears in the computation has no more than n distinct prime factors in the interval $(A, 2A)$, it means that the probability that a (uniformly sampled) random prime from this interval divides x is at most $n \cdot \mathcal{O}\left(\frac{1}{n^4 2^{4Cd^2}}\right) = \mathcal{O}\left(\frac{1}{n^3 2^{4Cd^2}}\right)$.

Consider the procedure `CountElimTrees` modified as follows: at the beginning we sample uniformly at random a prime $p \in (A, 2A)$ and instead of computing every number explicitly, we work in the ring $\mathbb{Z}_p = \mathbb{Z}/(p)$ and thus only compute the remainders modulo p. If the number of elimination trees of G of depth at most d is 0, then we are sure that this algorithm eventually obtains 0 as well. However, if this number is nonzero, then this algorithm will obtain 0 modulo p with probability is at most $\mathcal{O}\left(\frac{1}{n^3 2^{4Cd^2}}\right)$. Note that the bitsize of p is $d^{\mathcal{O}(1)} + \mathcal{O}(\log n)$, hence all arithmetic operations in $\mathbb{Z}_p$ can be performed in $d^{\mathcal{O}(1)}$ time in the RAM model. Hence, by working in the ring $\mathbb{Z}_p$ for a random prime $p \in (A, 2A)$, we significantly improve the cost of arithmetic operations while sacrificing only a little in terms of the correctness. That is, testing whether the number of elimination trees of G of depth at most d is nonzero may result in a false negative with probability $\mathcal{O}\left(\frac{1}{n^3 2^{4Cd^2}}\right)$, so we have a Monte Carlo algorithm with one-sided error. Throughout the remaining part of this article, we are sometimes going to refer to numbers that are present in the computation of `CountElimTrees` working in $\mathbb{Z}$ as *true* numbers, as opposed to their corresponding remainders that appear in the computation where `CountElimTrees` works in $\mathbb{Z}_m$ for some number m.

Let us briefly describe how we sample a random prime from the interval $(A, 2A)$. We repeat the following procedure until we find the first prime: We first uniformly sample a random integer from this interval and then we check whether it is prime using the AKS primality test [2]. As argued before, there are at least $\frac{A}{\log_2 A+1}$ primes in this interval, hence the probability of finding a prime when sampling a random number from this interval is at least $\frac{1}{\log_2 A+1}$. Therefore, the expected number of trials needed to sample a prime will be at most $\log_2 A + 1$. The AKS primality test works in time $(\log A)^{\mathcal{O}(1)}$, hence the expected work spent till discovering a prime is $(\log A)^{\mathcal{O}(1)} = (d \log n)^{\mathcal{O}(1)} \subseteq d^{\mathcal{O}(1)} n$. So this is a Las Vegas algorithm (which obviously can be turned into a Monte Carlo algorithm by stopping it after a certain number of failed trials). Note that we draw only one random prime p at the very beginning of our algorithm, and whenever we want to use a prime, we use this one.

After improving both the degrees of involved polynomials and the cost of arithmetic operations, single call of `CountElimTrees` in its current version takes $2^{\mathcal{O}(dk)} n$ time.

5.2.2 Faster root recovery

Having improved the running time of `CountElimTrees` to linear, now we are going to improve the running time of `ConstructElimForest` to linear. Recall that `ConstructElimForest` in its current version iterates over all vertices $v \in V(G)$ and checks whether $\operatorname{td}(G-v) \leq d-1$ (by calling `CountElimTrees` with appropriate parameters) — such vertices v could be placed as roots of an elimination tree of G of depth at most d. Finding any feasible root is the crucial part that needs to be optimized in order to achieve a linear running time for `ConstructElimForest`. The key fact we are going to use is that the number of possible roots of optimum-depth elimination forests of a connected graph is bounded in terms of the treedepth [18, 19, 31].

We need a definition.

Definition 5.2.2. We say that a graph G is a *minimal obstruction* for treedepth d if $\mathrm{td}(G) > d$, but $\mathrm{td}(G - v) \le d$ for each $v \in V(G)$.

Dvořák et al. [31] proved that every minimal obstruction for treedepth d satisfies $|V(G)| \le 2^{2^{d-1}}$. This bound was later on improved by Chen et al. [18, 19] to $d^{\mathcal{O}(d)}$, as we showed in Chapter 4. An easy consequence of these facts is the following:

Lemma 5.2.3. *Suppose G is a graph whose treedepth is equal to d. Then there are at most $d^{\mathcal{O}(d)}$ vertices $v \in V(G)$ such that* $\mathrm{td}(G - v) < d$.

Proof. Let G' be an inclusion-wise minimal induced subgraph of G satisfying $\mathrm{td}(G') = d$. By minimality, G' is a minimal obstruction for treedepth $d - 1$. So by the result of Chen et al. [18, 19], $|V(G')| \in d^{\mathcal{O}(d)}$. Note that for every $v \in V(G) \setminus V(G')$ we have $\mathrm{td}(G - v) = d$, for in such case $G - v$ contains G' as an induced subgraph and $\mathrm{td}(G') = d$. So, the number of vertices v satisfying $\mathrm{td}(G - v) < d$ is bounded by $|V(G')|$, which in turn is bounded by $d^{\mathcal{O}(d)}$. □

Note that any improvement in the upper bound on the sizes of obstructions entails an analogous improvement in the bound of Lemma 5.2.3. Let us point out that improvement of this bound from Chapter 4 was crucial, as otherwise we would not achieve desired complexity. Also observe that supposing G is connected, vertices v satisfying $\mathrm{td}(G - v) < \mathrm{td}(G)$ are exactly those that can be placed as roots of an optimum-depth elimination tree.

As the next step, we are going to modify the procedure `CountElimTrees`(G, T, d) by introducing weights. Let G be a connected graph. Enumerate vertices of G as $V(G) = \{v_1, \ldots, v_n\}$ and let t_i be the number of elimination trees of G that are sensible with respect to T and in which v_i is the root. Then the result of `CountElimTrees`(G, T, d) can be expressed as $t_1 + t_2 + \ldots + t_n$. However, with a slight modification, we are able to compute $t_1\mu_1 + t_2\mu_2 + \ldots + t_n\mu_n$ for any sequence $\mu_1, \mu_2, \ldots, \mu_n \in \mathbb{Z}$. In order to do so, we change the formula from Lemma 5.1.5 to the following:

$$\begin{aligned} f(u, K, \phi, A) = & \sum_{v \in A} x \cdot g(u, K, \phi[u \to v], A) \cdot \mu(u, v) + \\ & \sum_{w \in K} \sum_{p=1}^{d-\mathsf{depth}(w)} \frac{1}{x^{p-1}} \\ & \sum_{B \subseteq \{w_1, \ldots, w_{p-1}\}} (-1)^{p-1-|B|} g(u, K[w, w_1, \ldots, w_p], \phi[u \to w_p], A \cup B \cup \{w_p\}), \end{aligned}$$

where

$$\mu(v_i, u) = \begin{cases} \mu_i & \text{if } u \text{ is the root of } K, \\ 1 & \text{otherwise.} \end{cases}$$

Similarly, we adjust the formula for the polynomial h:

$$h = \sum_{p=1}^{d} \frac{1}{x^{p-1}} \sum_{B \subseteq \{w_1, \ldots, w_{p-1}\}} (-1)^{p-1-|B|} g(r, [w_1, \ldots, w_p], [r \to w_p], B \cup \{w_p\}) \cdot \mu(r, w_p)$$

(w_p is the root of the path $[w_1, \ldots, w_p]$ if and only if $p = 1$).

Naturally, the definition of $f(\cdot, \cdot, \cdot, \cdot)$ and $g(\cdot, \cdot, \cdot, \cdot)$ change as well. Instead of simply counting monsters in a weighted fashion so that the contribution of every monster to the sum is the product of numbers μ_i over all v_i-s that were mapped onto the root in the monster (the empty product is assumed to be equal to 1).

However, we already know that the contribution of each monster that is not a valid elimination tree cancels out, so only valid elimination trees remain in the final result. For these, exactly one vertex was mapped to the root of the generalized elimination tree, hence the contribution of each such elimination tree is μ_i instead of 1, where v_i is the vertex that is mapped to the root. All in all, the final result is indeed equal to $\sum_{i=1}^{n} t_i \mu_i$, as claimed.

Assume wishfully that there is exactly one vertex $v_i \in V(G)$ that could serve as the root of an elimination tree of G of depth d; equivalently, v_i is the only vertex such that $\mathrm{td}(G - v_i) < d$. In other words, t_j is nonzero if and only if $i = j$. Note that in such case we have $i = \frac{\sum_{j=1}^{n} j \cdot t_j}{\sum_{j=1}^{n} t_j}$. The denominator of this expression is simply the number of all elimination trees of G of depth at most d that are sensible with respect to T, while the numerator is the result of the modified version of `CountElimTrees` where we set $\mu_j = j$ for all $j \in [n]$. Hence, we can find i (that is: pinpoint the unique root) by dividing the outcomes of two calls to weighted `CountElimTrees`, instead of calling `CountElimTrees` n times, as we did previously. Note that such division can be performed both in $\mathbb{Z}$ and in $\mathbb{Z}_p$ for any prime p, unless the denominator is zero. In case of $\mathbb{Z}_p$, it takes $\mathcal{O}(\log p)$ arithmetic operations to compute modular inverse, which unfortunately poses a technical challenge in the time complexity analysis: if applied without care, it would lead to the increase of time complexity to $\mathcal{O}(n \log n)$ time, because we would perform a linear number of divisions in $\mathbb{Z}_p$. This issue will be resolved in the final time complexity analysis, so let us ignore it for now.

Next, we lift the assumption about the uniqueness of the candidate for the root of an elimination tree. There are two key ingredients here. The first one is Lemma 5.2.3, which bounds the number of possible candidate roots for elimination trees of optimum depth. The second one is the color coding technique.

Suppose $\mathrm{td}(G) = d$. We can do so, as we enter that part of the algorithm only if $\mathrm{td}(G) \leq d$ and we can determine $\mathrm{td}(G)$ by calling `CountElimTrees`(G, T, d') for $d' = 1, 2, \ldots, d$ and set d as the smallest value of d' where it returns a nonzero value, which will be equal to $\mathrm{td}(G)$ (assuming we did not encounter a false negative). We can also assume that G is connected as otherwise we can make a separate call on each connected component. Let R be the set of vertices that are potential roots of optimum-depth elimination trees of G; that is, $v \in R$ if and only if $\mathrm{td}(G - v) < d$. Then, Lemma 5.2.3 implies that $|R| \in d^{\mathcal{O}(d)}$, and obviously, we have $|R| \geq 1$. Let $B \in d^{\mathcal{O}(d)}$ be the specific bound stemming from Lemma 5.2.3. Consider a random coloring of $V(G)$ with B colors, that is, a function $C \colon V(G) \to [B]$ where each vertex is independently and uniformly mapped to a random number from $[B]$. We note the following: (here, e is the Euler's number)

Lemma 5.2.4. *With probability at least $\frac{1}{e}$ there is a color $c \in [B]$ such that $|R \cap C^{-1}(c)| = 1$.*

Proof. Let v be any vertex from R (recall that R is nonempty). If all other vertices from R have colors different from that of v, then $C(v)$ is a color fulfilling the desired property. This happens with probability

$$\left(1 - \frac{1}{B}\right)^{|R|-1} \geq \left(1 - \frac{1}{B}\right)^{B-1} = \frac{1}{\left(1 + \frac{1}{B-1}\right)^{B-1}} \geq \frac{1}{e}. \qquad \square$$

For each $c \in [B]$ we do the following. Create a sequence $X = (x_1, \ldots, x_n)$, where $x_i = 1$ if $C(v_i) = c$ and $x_i = 0$ otherwise, and a sequence $Y = (y_1, \ldots, y_n)$, where $y_i = i$ if $C(v_i) = c$ and $y_i = 0$ otherwise. Then, we call the modified version of `CountElimTrees`, where X is supplied as the sequence $\mu_1, \ldots, \mu_n$, and then call it again with Y instead of X. Similarly as in the case of unique candidates for a root from the previous paragraph, the number $i := \frac{\sum_{j=1}^{n} t_j \cdot y_j}{\sum_{j=1}^{n} t_j \cdot x_j}$ will be the index of a possible root, provided that there exists exactly one possible root with that color. If the denominator of that expression is nonzero, $i \in C^{-1}(c)$, and $\mathrm{td}(G - v_i) = d - 1$, then we are sure that $v_i \in R$. If we do not succeed in finding any member of R for any color c in this way, we repeat the procedure with a different coloring until we find one. As we execute this part of the algorithm only if R is nonempty, by Lemma 5.2.4, the expected number of colorings we need to try until we discover a member of R is at most e.

As checking each coloring takes at most $3B \in d^{\mathcal{O}(d)}$ executions of the modified version of `CountElimTrees`, identifying any possible root of an optimum-depth elimination tree takes expected $2^{\mathcal{O}(d^2)} \cdot n$ time. After identifying one, we remove it from the graph, partition the remaining part into connected components (and appropriately distribute the elimination tree T into elimination trees of connected components). and recurse for each connected component. After that, we connect roots of elimination trees returned from recursive calls as children of the root found on this level, obtaining an elimination tree for the whole G. There will be at most d recursion levels and the total size of graphs on each level is at most n, hence the expected total work that `CountElimTrees` calls will perform will be $2^{\mathcal{O}(d^2)} \cdot n$ as well. However, as mentioned before, this does not include the time needed for divisions in $\mathbb{Z}_p$ and we defer this analysis to a later part.

5.2.3 Replacing iterative compression

Finally, we replace the iterative compression scheme with a technique proposed by Bodlaender in his linear-time FPT algorithm to compute the treewidth of a graph [6]. The main part of this technique was succinctly encapsulated in [7, Lemma 2.7]. We need a few definitions.

Definition 5.2.5. For a graph G and an integer d, the *d-improved graph* of G, denoted $G^{\langle d \rangle}$, is the graph obtained from G by adding an edge between every pair of vertices that are non-adjacent, but have at least $d+1$ common neighbours of degree at most d in G.

We note the following.

Lemma 5.2.6. *For every graph G and integer d, we have $\mathrm{td}(G) \leq d$ if and only if $\mathrm{td}(G^{\langle d \rangle}) \leq d$.*

Proof. The right-to-left implication is obvious, so we need to prove that if $\mathrm{td}(G) \leq d$, then $\mathrm{td}(G^{\langle d \rangle}) \leq d$. Let F be an elimination forest of G of depth at most d. We claim that F is also an elimination forest of $\mathrm{td}(G^{\langle d \rangle})$. Suppose otherwise. Then there are vertices $u, v \in V(G)$ such that $\mathsf{Anc}_F(u, v)$ does not hold, while uv is an edge in $G^{\langle d \rangle}$. Since F is an elimination forest of G, u and v are non-adjacent in G but have at least $d+1$ common neighbors. However, as $\mathsf{Anc}_F(u, v)$ does not hold, every common neighbor of u and v belongs to $\mathsf{tail}_F(u) \cap \mathsf{tail}_F(v)$, which is a set of cardinality smaller than d. This is a contradiction. $\square$

Recall that we are given an n-vertex graph G and we would like to construct an elimination forest of G of depth at most d, or conclude that $\mathrm{td}(G) > d$. It is well-known that an n-vertex graph of treedepth at most d has at most dn edges, hence we may assume that $|E(G)| \leq dn$; otherwise we immediately provide a negative answer. In that case, as proved by Bodlaender [6], the d-improved graph $G^{\langle d \rangle}$ can be computed in time $d^{\mathcal{O}(1)} \cdot n$ using radix sort. We call a vertex v of G *d-improved-simplicial* if the neighbourhood $N_{G^{\langle d \rangle}}[v]$ is a clique in $G^{\langle d \rangle}$. Note that if in $G^{\langle d \rangle}$ there is a clique of size at least $d+1$, then $\mathrm{td}(G^{\langle d \rangle}) > d$, which in turn implies that $\mathrm{td}(G) > d$ due to Lemma 5.2.6.

We now recall the aforementioned statement from [7].

Lemma 5.2.7 (Lemma 2.7 of [7])**.** *There is an algorithm working in time $d^{\mathcal{O}(1)} \cdot n$ time that, given an n-vertex graph G and an integer d, either*

1. *returns a maximal matching in G of cardinality at least $\frac{n}{\mathcal{O}(d^6)}$, or,*
2. *returns a set of at least $\frac{n}{\mathcal{O}(d^6)}$ d-improved-simplicial vertices, or*
3. *correctly concludes that the treewidth of G is larger than d.*

With the lemma stated, we are ready to optimize the way we use `ConstructElimForest` in order to construct an elimination forest of G.

We define a procedure $\texttt{Solve}(G, d)$ that for a graph G and an integer d, either reports that $\mathrm{td}(G) > d$ or provides an elimination forest of G of depth at most d. If G consists of a single vertex, we return it as a valid elimination forest of depth 1, so we assume that $|V(G)| > 1$ from now on. As the very first step, we check if $|E(G)| \leq dn$. As argued, if this is not the case, then we report that $\mathrm{td}(G) > d$ and terminate. Otherwise, we apply the algorithm of Lemma 5.2.7 with G and d as an input. If it reports that $\mathrm{tw}(G) > d$, then this implies that also $\mathrm{td}(G) > d$, so this conclusion can be reported and the procedure terminated.

Next, suppose the procedure returns a matching M of size at least $\frac{n}{\mathcal{O}(d^6)}$. Contract all edges of M, thus obtaining a new graph G_M as a result. Call $\texttt{Solve}(G_M, d)$. Note that if this procedure returned that $\mathrm{td}(G_M) > d$, then we also have $\mathrm{td}(G) > d$, because G_M is a minor of G and treedepth is monotone under taking minors. Therefore, we may assume that we have obtained an elimination forest F' of G_M of depth at most d. We can now easily transform F' into an elimination forest F'' of G of depth at most $2d$, by replacing every vertex obtained from the contraction of an edge of M by the two endpoints of this edge (these two vertices are put in place of the contracted as a parent and a child). Then, we may call $\texttt{ConstructElimForest}(G, F'', d)$ to either conclude that whether $\operatorname{td}(G) > d$, to construct an elimination forest of G of depth at most d.

Finally, suppose the procedure of Lemma 5.2.7 returned a set A consisting of at least $\frac{n}{\mathcal{O}(d^6)}$ d-improved-simplicial vertices. We compute $G^{\langle d \rangle}$ and call $\texttt{Solve}(G^{\langle d \rangle} \setminus A, d)$. If this call reports that $\operatorname{td}(G^{\langle d \rangle} \setminus A) > d$, then by Lemma 5.2.6 we also have $\operatorname{td}(G) > d$, hence we can return this conclusion and terminate the algorithm. Otherwise, we have an elimination forest F' of $G^{\langle d \rangle} \setminus A$ of depth at most d. We order A arbitrarily as $v_1, \ldots, v_a$ and process these vertices one by one. We shall iteratively construct $F_0, F_1, \ldots, F_a$, where each F_i is an elimination forest of $G \setminus \{v_{i+1}, \ldots, v_a\}$. We set F_0 to be F. Now, we argue how F_i can be constructed from F_{i-1}, for $i = 1, 2, \ldots, a$. Since v_i is d-improved-simplicial in G, the neighbourhood in $G^{\langle d \rangle} \setminus \{v_{i+1}, \ldots, v_a\}$ is a clique (we may assume that this clique has size smaller than d, for otherwise it is safe to conclude that $\operatorname{td}(G^{\langle d \rangle}) > d$, implying $\operatorname{td}(G) > d$). This implies that all the neighbors of v_i in this graph lie on some root-to-leaf path in F_{i-1}. We can easily see that if we take the neighbor that is the lowest in F_{i-1} and attach v_i as its child, what we get as a result is a valid elimination forest of $G^{\langle d \rangle} \setminus \{v_{i+1}, \ldots, v_a\}$ and we may call it F_i. This way, we can compute F_a from F in time $d^{\mathcal{O}(1)} \cdot n$, and such F_a is a valid elimination forest of $G^{\langle d \rangle}$. We claim that if the depth of F_a is larger than $2d$ then $\operatorname{td}(G) > d$. Suppose so and take any u such that $\texttt{depth}_{F_a}(u) = 2d + 1$. Let $u_1, u_2, \ldots, u_{2d+1}$ be the path from the root to u in F_a, where $u_{2d+1} = u$. Note that since the depth of F is at most d, the vertices $u_{d+1}, u_{d+2}, \ldots, u_{2d+1}$ were all added in the process of obtaining F_a from F_0, meaning that they are all d-improved-simplicial in G and pairwise adjacent. In particular, $u_{d+1}, u_{d+2}, \ldots, u_{2d+1}$ is a clique of size $d + 1$ in $G^{\langle d \rangle}$, implying $\operatorname{td}(G^{\langle d \rangle}) > d$, which in turn implies that $\operatorname{td}(G) > d$; so it is safe to return this conclusion then. Otherwise, we have obtained an elimination forest F_a of G of depth at most $2d$. It now remains to call $\texttt{ConstructElimForest}(G, F_a, d)$ to either conclude that $\operatorname{td}(G) \leq d$, or construct an elimination forest of G of depth at most d.

In short, the size of our graph shrinks by a constant factor with each recursive call, hence we improve the running time by a factor of n. We perform more detailed analysis in the next section.

5.2.4 Detailed specification and the analysis of the time and space complexity

Throughout previous subsections we introduced a series of modifications to the deterministic algorithm from Theorem 1.1.10 in order to improve the $n^{\mathcal{O}(1)}$ factor to n. However, as there are nontrivial dependencies between these improvements and interplays between various sources of randomness, some details were omitted. Only now that we have an overall view of modifications, we may fully specify and analyze the algorithm. In this section we assume that n always denotes the number of vertices of the original input graph, while r denotes the number of vertices of a graph that was passed to either `CountElimTrees` or `ConstructElimForest` in some recursive call.

Each call of the modified version of `CountElimTrees` is computed in a ring $\mathbb{Z}_m$ for some number m. If m is prime, then $\mathbb{Z}_m$ can be equipped with a division operation so that it becomes the field $\mathbb{F}_m$. We promise that it will always hold that $m \in n^{\mathcal{O}(d)}$, hence the bitsizes of all numbers present in the computation will never be larger than $\mathcal{O}(d \log n)$. Hence, additions, subtractions and multiplications on such numbers take $d^{\mathcal{O}(1)}$ time and space in the RAM model.

For the unweighted version of `CountElimTrees`, Lemma 5.1.7 shows the bound of $(dk \cdot 2^d)^r$ for all numbers present in the computation when performed on a graph with r vertices. However, with the introduction of weights, this bound grows into $(Wdk \cdot 2^d)^r$, where W is the maximum supplied weight. After the appropriate renumeration of vertices in each recursive call, we can assume that $W \leq r$, which gives a bound of $(rdk \cdot 2^d)^r$ on the numbers present in the computation. Interestingly enough, even though intermediate numbers present in the computation of weighted `CountElimTrees` can be as large as r^r, the final result $\sum_{i=1}^{r} t_i \mu_i$ can be bounded more efficiently. Namely, we have $\sum_{i=1}^{r} t_i \mu_i \leq W \sum_{i=1}^{r} t_i$ and we already know from Lemma 5.1.7 that $\sum_{i=1}^{r} t_i \leq (dk \cdot 2^d)^r$. Hence the outcomes returned by the weighted version of `CountElimTrees` are bounded by $r(dk \cdot 2^d)^r$.

We need to specify what numbers m we use as moduli in `CountElimTrees`. On one hand, we want to use large numbers, so that probabilities of errors are small. On the other hand, we need to deal with the issue of modular division cost potentially worsening our complexity to $\mathcal{O}(n \log n)$. The idea to deal with it is to distinguish two cases based on whether r is large or small. If r is large, the division cost will not be larger than the cost of `CountElimTrees`. If r is small, then the bound on the result is sufficiently small so that performing the whole computation without hashing modulo a large prime (almost) fits into the RAM model and provides a true outcome at the end.

More specifically, we distinguish two cases:

1. $r \geq \log_2 n$

   In that case, we use as m the random prime p that we drew at the beginning from the interval $(A, 2A)$, where $A = \max(L, n^5 2^{5Cd^2})$. We have $\log m \in d^{\mathcal{O}(1)} + \mathcal{O}(\log n)$ and the bound we use for the running time of the call to `CountElimTrees` is $2^{\mathcal{O}(d^2)} r$. As a consequence, calling a modular inverse taking $\log m \cdot d^{\mathcal{O}(1)}$ time does not worsen the time complexity, as $\log m \cdot d^{\mathcal{O}(1)} \subseteq 2^{\mathcal{O}(d^2)} r$.

2. $r < \log_2 n$

   In that case we use $r(dk \cdot 2^d)^r + 1$ as m. In all our calls $k = \mathcal{O}(d)$, hence numbers of this magnitude will have bitsize $\mathcal{O}(d \log n)$, so again, arithmetic operations on them can be performed in $d^{\mathcal{O}(1)}$ time in the RAM model. As explained before, even though true numbers that would be present in the computations could hypothetically exceed the value of m, the final result will not, hence the result modulo m is equal to the true result. In other words $(\sum_{i=1}^{r} t_i \mu_i) \mod m = \sum_{i=1}^{r} t_i \mu_i$. Because of that, the division $\frac{\sum_{i=1}^{r} t_i \cdot i}{\sum_{i=1}^{r} t_i}$ can be performed on ordinary integers instead of on their moduli, and it takes $d^{\mathcal{O}(1)}$ time instead of $\mathcal{O}(\log n)$ time. We note that if this division does not result in an integer number, we already know that the we did not succeed in finding a candidate for a root in this color and we may continue to search within other colors. We also note that there is no randomness in this case, the output of this case is always correct.

The expected total cost of divisions in the first case is not larger than the work that `CountElimTrees` performs, hence it can be bounded by $2^{\mathcal{O}(d^2)} n$. Because the expected number of `CountElimTrees` calls is $d^{\mathcal{O}(d)} n$, and the expected total cost of divisions in the second case is $d^{\mathcal{O}(d)} n$ as well. As such, we conclude that the expected total cost of divisions is $2^{\mathcal{O}(d^2)} n$ too. Therefore, the expected time that one `ConstructElimForest` call takes on the graph on n vertices is $2^{\mathcal{O}(d^2)} n$.

In the next step, we come back to the time and space complexity analysis of the recursive scheme that replaced iterative compression technique. As for the time complexity, in both non-trivial cases we make a single recursive call on a graph with $n(1 - \frac{1}{\mathcal{O}(d^6)})$ vertices, and perform additional work taking expected $2^{\mathcal{O}(d^2)} \cdot n$ time. Hence the expected time complexity $T(n, d)$ can be bounded using recurrence

$$T(n, d) \leq T\left(n\left(1 - \frac{1}{\mathcal{O}(d^6)}\right), d\right) + 2^{\mathcal{O}(d^2)} \cdot n.$$

As in [6], this recurrence solves to $T(n, d) = 2^{\mathcal{O}(d^2)} \cdot n$, because unraveling the recursion results in a geometric series. As for the space complexity, we have argued that both `ConstructElimForest` and internal computation of `Solve`(G, d) use $d^{\mathcal{O}(1)} \cdot n$ space. Therefore, the space complexity $S(n, d)$ can be bounded using recurrence

$$S(n, d) \leq S\left(n\left(1 - \frac{1}{\mathcal{O}(d^6)}\right), d\right) + d^{\mathcal{O}(1)} \cdot n,$$

which again solves to $S(n, d) = d^{\mathcal{O}(1)} \cdot n$.

In order to conclude, we need to bound the error probability. We recall that the randomness stemming from color coding and drawing a random prime is of type Las Vegas, that is, there is a possibility that the algorithm runs indefinitely long, but there are no errors that this randomness introduces. By using

Markov's inequality we know that there is at most $\frac{1}{n}$ chance that our algorithm takes time that is at least n times longer than its expected execution time, hence with at least $\frac{n-1}{n}$ probability there will be at most $2^{\mathcal{O}(d^2)}n^2$ calls to `CountElimTrees`. As argued before, the errors stem only from cases where the true result of `CountElimTrees` should be nonzero, but becomes zero as a result of unluckily chosen modulo m. The probability of that happening for a particular call is at most $\frac{1}{2^{Cd^2}n^3}$ for any constant C of our choice. By using the union bound, we conclude that the probability that we never encounter any error of this type is at least $\frac{n-1}{n} - \frac{2^{\mathcal{O}(d^2)}n^2}{2^{Cd^2}n^3} \geq \frac{n-2}{n}$, for any sufficiently large C. We remark that the errors are of the false negative type, that is, if an elimination forest is returned, it is guaranteed to a be a valid elimination forest of depth at most d. This concludes the description of the procedure `Solve`(G, d) and the analysis of its time complexity, space complexity, and the probability of correctness.

Chapter 6

Excluded-minor theorem for pathwidth

In this Chapter we are going to discuss polynomial excluded-minor theorem for pathwidth. More specifically, we are going to prove Theorem 1.2.2.

Theorem 1.2.2. *Every graph with treewidth $t-1$ has pathwidth at most $th+1$ or contains a subdivision of a complete binary tree of height $h+1$.*

First, we will make necessary preparations for that, then we are going to prove this Theorem, show its tightness and then discuss related work.

6.1 Basic tools

6.1.1 Witnesses for Large Pathwidth

Recall that $(\mathcal{T}_h)_{h=0}^{\infty}$ is the sequence of classes of graphs defined inductively as follows: $\mathcal{T}_0$ is the class of all connected graphs, and $\mathcal{T}_{h+1}$ is the class of connected graphs G that contain three pairwise disjoint sets of vertices V_1, V_2, and V_3 such that $G[V_1], G[V_2], G[V_3] \in \mathcal{T}_h$ and any two of V_1, V_2, and V_3 can be connected in G by a path avoiding the third one.

A $\mathcal{T}_h$*-witness* for a graph $G \in \mathcal{T}_h$ is a complete ternary tree of height h of subsets of $V(G)$ defined inductively following the definition of $\mathcal{T}_h$. The $\mathcal{T}_0$-witness for a connected graph G is the tree with the single node $V(G)$, denoted by $\langle V(G)\rangle$. A $\mathcal{T}_{h+1}$-witness for a graph $G \in \mathcal{T}_{h+1}$ is a tree with root $V(G)$ and with three subtrees W_1, W_2, W_3 of the root that are $\mathcal{T}_h$-witnesses of $G[V_1], G[V_2], G[V_3]$ for some sets V_1, V_2, V_3 as in the definition of $\mathcal{T}_{h+1}$; it is denoted by $\langle V(G); W_1, W_2, W_3\rangle$.

It clearly follows from these definitions that every graph in $\mathcal{T}_h$ has at least 3^h vertices and every $\mathcal{T}_h$-witness of an n-vertex graph has $O(n)$ nodes. The next two lemmas explain the connection of $\mathcal{T}_h$ to pathwidth and to subdivisions of complete binary trees.

Lemma 6.1.1. *If $G \in \mathcal{T}_h$, then* $\mathrm{pw}(G) \geq h$.

Proof. The proof goes by induction on h. The case $h = 0$ is trivial. Now, assume that $h \geq 1$ and the lemma holds for $h-1$. Since $G \in \mathcal{T}_h$, there are sets $V_1, V_2, V_3 \subseteq V(G)$ interconnected as in the definition of $\mathcal{T}_h$, such that $G[V_i] \in \mathcal{T}_{h-1}$ and thus $\mathrm{pw}(G[V_i]) \geq h-1$ for $i = 1, 2, 3$. Let P be a path decomposition of G. With bags restricted to V_i, it becomes a path decomposition of $G[V_i]$. It follows that for $i = 1, 2, 3$, there is a bag B_i in P such that $|B_i \cap V_i| \geq h$. Assume without loss of generality that B_1, B_2, B_3 occur in this order in P. Since $G[V_1]$ and $G[V_3]$ are connected, there is a path that connects $B_1 \cap V_1$ and $B_3 \cap V_3$ in G avoiding V_2. This path must have a vertex in B_2, so $|B_2 \setminus V_2| \geq 1$ and thus $|B_2| \geq h+1$. This proves that $\mathrm{pw}(G) \geq h$. □

The proof of Lemma 6.1.1 generalizes the well-known proof of the fact that (a subdivision of) a complete binary tree of height h has pathwidth at least $\lceil h/2\rceil$. Actually, it is straightforward to show that such a tree belongs to $\mathcal{T}_{\lceil h/2\rceil}$.

Lemma 6.1.2. *If $G \in \mathcal{T}_h$, then G contains a subdivision of a complete binary tree of height h as a subgraph. Moreover, it can be computed in polynomial time from a $\mathcal{T}_h$-witness for G.*

Proof. We prove, by induction on h, that for every graph $G \in \mathcal{T}_h$ and every $v \in V(G)$, the following structure exists in G: a subdivision S of a complete binary tree of height h with some root r and a path P from v to r such that $V(P) \cap V(S) = \{r\}$. This is trivial for $h = 0$. For the induction step, assume that $h \geq 1$ and the statement holds for $h - 1$. Let $G \in \mathcal{T}_h$ and $v \in V(G)$. Let $V_1, V_2, V_3 \subseteq V(G)$ be as in the definition of $\mathcal{T}_h$. Assume without loss of generality that $v \in V_3$ or v can be connected with V_3 by a path in G avoiding $V_1 \cup V_2$. For $i = 1, 2$, since $G[V_3]$ is connected and G has a path connecting V_i with V_3 and avoiding V_{3-i}, there is also a path in G from v to some vertex $v_i \in V_i$ avoiding $V_1 \cup V_2 \setminus \{v_i\}$. These paths can be chosen so that they first follow a common path P from v to some vertex r in $G - (V_1 \cup V_2)$ and then they split into a path Q_1 from r to v_1 and a path Q_2 from r to v_2 so that r is the only common vertex of any two of P, Q_1, Q_2. For $i = 1, 2$, the induction hypothesis provides an appropriate structure in $G[V_i]$: a subdivision S_i of a complete binary tree of height $h - 1$ with root r_i and a path P_i from v_i to r_i such that $V(P_i) \cap V(S_i) = \{r_i\}$. Connecting r with S_1 and S_2 by the combined paths Q_1P_1 and Q_2P_2, respectively, yields a subdivision S of a complete binary tree of height h with root r in G. The construction guarantees that $V(P) \cap V(S) = \{r\}$.

Clearly, given a $\mathcal{T}_h$-witness for G, the induction step described above can be performed in polynomial time, and therefore the full recursive procedure of computing a subdivision of a complete binary tree of height h in G works in polynomial time. □

6.1.2 Combining Path Decompositions

The following lemma will be used several times in the paper to combine path decompositions.

Lemma 6.1.3. *Let G be a graph and $(T, \{B_x\}_{x \in V(T)})$ be a tree decomposition of G of width $t - 1$.*

1. *If $q \in V(T)$ and every connected component of $G - B_q$ has pathwidth at most ℓ, then there is a path decomposition of G of width at most $\ell + t$ which contains B_q in every bag.*
2. *If Q is the path connecting x and y in T and every connected component of $G - \bigcup_{q \in V(Q)} B_q$ has pathwidth at most ℓ, then there is a path decomposition of G of width at most $\ell + t$ which contains B_x in the first bag and B_y in the last bag.*

In either case, there is a polynomial-time algorithm to construct such a path decomposition of G from the path decompositions of the respective components C of width at most ℓ.

Proof. In case 1, the path decomposition of G is obtained by concatenating the path decompositions of the connected components of $G - B_q$ (which have width at most ℓ) and adding B_q to every bag. Now, consider case 2. For every node q of Q, let T_q be the subtree of T induced on the nodes z such that the path from q to z in T contains no other nodes of Q, and let $V_q = \bigcup_{z \in V(T_q)} B_z$. Apply case 1 to the graph $G[V_q]$, the tree decomposition $(T_q, \{B_z\}_{z \in V(T_q)})$ of $G[V_q]$, and the node $q \in V(T_q)$ to obtain a path decomposition of $G[V_q]$ of width at most $\ell + t$ containing B_q in every bag. Then, concatenate the path decompositions thus obtained for all nodes q of Q (in the order they occur on Q) to obtain a requested path decomposition of G. □

6.2 Proof of Theorem 1.2.2

The statement of Theorem 1.2.2 on a graph G follows from the same statement on every connected component of G. By Lemma 6.1.2, every graph in $\mathcal{T}_h$ contains a subdivision of a complete binary tree of height h. Thus, Theorem 1.2.2 is a direct corollary to the following statement.

Theorem 6.2.1. *For every $h \in \mathbb{N}$, every connected graph with treewidth at most $t - 1$ has pathwidth at most $th + 1$ or belongs to $\mathcal{T}_{h+1}$.*

A tree decomposition of G is *optimal* if its width is equal to $\mathrm{tw}(G)$. For the proof of Theorem 6.2.1, we need optimal tree decompositions with an additional property. Namely, consider a connected graph G and a tree decomposition $(T, \{B_x\}_{x\in V(T)})$ of G. Every edge xy of T splits T into two subtrees: $T_{x|y}$ containing x and $T_{y|x}$ containing y. For every oriented edge xy of T, let $G_{x|y}$ denote the subgraph of G induced on the union of the bags of the nodes in $T_{x|y}$. The property we need is that every subgraph of the form $G_{x|y}$ is connected. It is known that such a tree decomposition always exists [37], but for completeness, we present a short proof of this fact in the following lemma.

Lemma 6.2.2. *Every connected graph G has an optimal tree decomposition $(T, \{B_x\}_{x\in V(T)})$ with the property that $G_{x|y}$ is connected for every oriented edge xy of T.*

Proof. Let $t = \mathrm{tw}(G) + 1$. The *fatness* of an optimal tree decomposition $(T, \{B_x\}_{x\in V(T)})$ of G is the t-tuple $(k_0, \dots, k_{t-1})$ such that k_i is the number of bags B_x of size $t - i$. Let $(T, \{B_x\}_{x\in V(T)})$ be an optimal tree decomposition of G with lexicographically minimum fatness. (The idea of taking such a tree decomposition comes from the proof of existence of optimal "lean" tree decompositions due to Bellenbaum and Diestel [4, Theorem 3.1].) We show that it has the required property.

Suppose it does not. Let xy be an edge of T such that $G_{x|y}$ is disconnected and the number of nodes in the subtree $T_{x|y}$ of T is minimized. Let $\mathcal{C}$ be the family of connected components of $G_{x|y}$, so that $|\mathcal{C}| \geq 2$. Let $Z = N_T(x) \setminus \{y\}$. For every node $z \in Z$, let C_z be the connected component of $G_{x|y}$ that contains $G_{z|x}$, which exists because the choice of xy guarantees that $G_{z|x}$ is connected.

We modify $(T, \{B_x\}_{x\in V(T)})$ into a new tree decomposition of G as follows. We keep all nodes other than x (with their bags B_x) and all edges non-incident to x. We replace the node x by $|\mathcal{C}|$ nodes x_C with bags $B_{x_C} = B_x \cap V(C)$ for each $C \in \mathcal{C}$. We replace the edge xy by $|\mathcal{C}|$ edges $x_C y$ for each $C \in \mathcal{C}$, and we replace the edge xz by an edge $x_{C_z} z$ for each $z \in Z$. It is straightforward to verify that what we obtain is indeed a tree decomposition of G and it has width t.

Since G is connected, we have $B_{x_C} = B_x \cap V(C) \neq \emptyset$ for every $C \in \mathcal{C}$. This and the assumption that $|\mathcal{C}| \geq 2$ imply that $|B_{x_C}| < |B_x|$ for all $C \in \mathcal{C}$. We conclude that the fatness of the new tree decomposition is lexicographically less than the fatness of $(T, \{B_x\}_{x\in V(T)})$, which contradicts the assumption that the latter is lexicographically minimal. □

Proof of Theorem 6.2.1. The proof goes by induction on h. The statement is true for $h = 0$: if a connected graph G has a cycle or a vertex of degree at least 3, then $G \in \mathcal{T}_1$, and otherwise G is a path, so $\mathrm{pw}(G) \leq 1$. For the rest of the proof, assume that $h \geq 1$ and the statement is true for $h - 1$.

Let G be a connected graph width treewidth at most $t - 1$ and $(T, \{B_x\}_{x\in V(T)})$ be an optimal tree decomposition of G obtained from Lemma 6.2.2. Thus $|B_x| \leq t$ for every node x of T and $G_{x|y}$ is connected for every oriented edge xy of T. For every oriented edge xy of T, let $F_{x|y}$ be the subgraph of G induced on the vertices not in $G_{y|x}$, that is, on the vertices that belong only to bags in $T_{x|y}$ and to no other bags. Let E be the set of edges xy of T such that both $F_{x|y}$ and $F_{y|x}$ have a connected component belonging to $\mathcal{T}_h$.

Suppose $E = \emptyset$. It follows that every pair of trees of the form $T_{x|y}$ such that $F_{x|y}$ has a connected component in $\mathcal{T}_h$ has a common node. This implies that all such trees have a common node, say z, by the well-known fact that subtrees of a tree have the Helly property [49, Theorem 4.1]. For every neighbor y of z in T and every connected component C of $F_{y|z}$, since $C \notin \mathcal{T}_h$, the induction hypothesis gives $\mathrm{pw}(C) \leq t(h-1)+1$. Lemma 6.1.3 1 applied with $q = z$ yields $\mathrm{pw}(G) \leq th + 1$.

For the rest of the proof, assume $E \neq \emptyset$. Since every connected supergraph of a graph from $\mathcal{T}_h$ belongs to $\mathcal{T}_h$, the set E is the edge set of some subtree K of T. Let Z be the set of leaves of K. Since K has at least one edge, we have $|Z| \geq 2$.

Suppose $|Z| \geq 3$. Choose any distinct $z_1, z_2, z_3 \in Z$. For each $i \in \{1, 2, 3\}$, let C_i be a connected component of $F_{z_i|x_i}$ that belongs to $\mathcal{T}_h$, where x_i is the unique neighbor of z_i in K. For each $i \in \{1, 2, 3\}$, the subgraph $G_{x_i|z_i}$ is connected, is vertex-disjoint from C_i, and contains the other two of C_1, C_2, and C_3. Consequently, any two of the sets $V(C_1)$, $V(C_2)$, and $V(C_3)$ can be connected by a path in G avoiding the third one. This shows that $G \in \mathcal{T}_{h+1}$.

Now, suppose $|Z| = 2$. It follows that K is a path $x_1 \dots x_m$, where $Z = \{x_1, x_m\}$. For every node x_i of K, every neighbor y of x_i in $T - K$, and every connected component C of $F_{y|x_i}$, since $C \notin \mathcal{T}_h$, the induction hypothesis gives $\mathrm{pw}(C) \leq t(h-1)+1$. Lemma 6.1.3 2 applied with $Q = K$ yields $\mathrm{pw}(G) \leq th+1$. □

6.3 Tightness

Theorem 1.2.2 asserts that every graph with pathwidth at least $th+2$ has treewidth at least t or contains a subdivision of a complete binary tree of height $h+1$. While this statement is true for all positive integers t and h, we remark that the interesting case is when $h > \log_2 t - 2$. Indeed, if $h \leq \log_2 t - 2$, then the second outcome is known to hold for every graph with pathwidth at least t; this follows from a result of Bienstock, Robertson, Seymour, and Thomas [5].[1] We now show that Theorem 1.2.2 is tight up to a multiplicative factor when $h > \log_2 t - 2$.

Theorem 6.3.1. *For any positive integers t and h, there is a graph with treewidth t and pathwidth at least $t(h+1)-1$ that contains no subdivision of a complete binary tree of height $3\max(h+1, \lceil \log_2 t \rceil)$.*

Fix a positive integer t. For a tree T, let $T^{(t)}$ be a graph obtained from T by replacing every node of T with a clique on t vertices and replacing every edge of T with an arbitrary perfect matching between the corresponding cliques. For $h \in \mathbb{N}$, let T_h be a complete ternary tree of height h. The following three claims show that the graph $T_h^{(t)}$ satisfies the three conditions requested in Theorem 6.3.1, thus proving that Theorem 6.3.1 holds for $T_h^{(t)}$.

Claim 6.3.2. *If T is a tree on at least two vertices, then $\mathrm{tw}(T^{(t)}) = t$.*

Proof. For each node x of T, let B_x be the clique of t vertices in $T^{(t)}$ corresponding to x. A tree decomposition of $T^{(t)}$ of width t is obtained from T by taking B_x as the bag of every node x of T and by subdividing every edge xy of T into a path of length $t+1$ with the following sequence of bags, assuming that the vertices $u_1, \dots, u_t$ in B_x are matched to $v_1, \dots, v_t$ in B_y, respectively:

$$\{u_1, \dots, u_t\},\ \{u_1, \dots, u_t\} \cup \{v_1\},\ \{u_2, \dots, u_t\} \cup \{v_1, v_2\},\ \dots,\ \{u_t\} \cup \{v_1, \dots, v_t\},\ \{v_1, \dots, v_t\}.$$

This way, for every matching edge u_iv_i with $1 \leq i \leq t$, there is a bag containing its two endpoints. Consequently, this is a valid tree decomposition of $T^{(t)}$ with bags of size at most $t+1$.

For the proof that $\mathrm{tw}(T^{(t)}) \geq t$, let xy be an edge of T, and assume that the vertices $u_1, \dots, u_t$ of B_x are matched to $v_1, \dots, v_t$ in B_y, respectively, as before. In any tree decomposition of $T^{(t)}$, there is a node x' whose bag contains the clique B_x and a node y' whose bag contains the clique B_y. Walk on the path from x' to y' and stop at the first node whose bag contains some vertex in B_y. This bag must also contain all of B_x, so it has size at least $t+1$. □

Claim 6.3.3. *For every $h \in \mathbb{N}$, we have $\mathrm{pw}(T_h^{(t)}) \geq t(h+1)-1$.*

Proof. We define the *root clique* of $T_h^{(t)}$ as the clique in $T_h^{(t)}$ corresponding to the root of T_h. We prove the following slightly stronger statement, by induction on h: in every path decomposition of $T_h^{(t)}$, there are a bag B of size at least $t(h+1)$ and t vertex-disjoint paths in $T_h^{(t)}$ each having one endpoint in the root clique and the other endpoint in B.

For the base case $h = 0$, the graph $T_0^{(t)}$ is simply a complete graph on t vertices, and the statement is trivial. For the induction step, assume that $h \geq 1$ and the statement is true for $h-1$. Let R be the root clique of $T_h^{(t)}$. Let $(P, \{B_x\}_{x \in V(P)})$ be a path decomposition of $T_h^{(t)}$ of minimum width. The graph $T_h^{(t)} - R$ has three connected components C_1, C_2, and C_3 that are copies of $T_{h-1}^{(t)}$ with root cliques R_1, R_2, and R_3, respectively. For each $i \in \{1,2,3\}$, the induction hypothesis applied to the path decomposition $(P, \{B_x \cap V(C_i)\}_{x \in V(P)})$ of C_i provides a node x_i of P such that

[1]In [5], it is proved that for every forest F, graphs with no F minors have pathwidth at most $|V(F)|-2$. In particular, if G contains no subdivision of a complete binary tree of height $h+1$, then $\mathrm{pw}(G) < 2^{h+2} \leq t$.

- $|B_{x_i} \cap V(C_i)| \geq th$, and
- there are t vertex-disjoint paths in C_i between $B_{x_i} \cap V(C_i)$ and the root clique R_i of C_i.

Assume without a loss of generality that the node x_2 occurs between x_1 and x_3 on P. We prove the induction statement for $B = B_{x_2}$.

For each $i \in \{1, 2, 3\}$, we take the t vertex-disjoint paths from B_{x_i} to R_i in C_i and extend them by the matching between R_i and R to obtain t vertex-disjoint paths from B_{x_i} to R in $T_h^{(t)}[R \cup V(C_i)]$. In particular, there are t vertex-disjoint paths from B_{x_2} to R in $T_h^{(t)}$, as required in the induction statement. Since $|R| = t$, the t paths from B_{x_1} to R and the t paths from B_{x_3} to R together form t vertex-disjoint paths from B_{x_1} to B_{x_3} in $T_h^{(t)}[V(C_1) \cup R \cup V(C_3)]$, which therefore avoid $V(C_2)$. Since x_2 lies between x_1 and x_3 on P, the set $B_{x_2} \setminus V(C_2)$ must contain at least one vertex from each of these t paths. Thus $|B_{x_2} \setminus V(C_2)| \geq t$. Since $|B_{x_2} \cap V(C_2)| \geq th$, we conclude that $|B_{x_2}| \geq t(h+1)$, as required in the induction statement. □

Claim 6.3.4. *For any $h \in \mathbb{N}$, the graph $T_h^{(t)}$ contains no subdivision of a complete binary tree of height $3\max(h+1, \lceil \log_2 t \rceil)$.*

Proof. A simple calculation shows that T_h has $\frac{3^{h+1}-1}{2}$ nodes. Thus $|V(T_h^{(t)})| + 1 \leq 3^{h+1}t$ and so

$$\log_2\big(|V(T_h^{(t)})| + 1\big) \leq \log_2(3^{h+1}t) \leq 3\max(h+1, \lceil \log_2 t \rceil).$$

If a graph G contains a subdivision of a complete binary tree of height c, then $|V(G)| \geq 2^{c+1} - 1$ and so $\log_2(|V(G)| + 1) \geq c + 1$. Therefore, $T_h^{(t)}$ cannot contain a subdivision of a complete binary tree of height $3\max(h+1, \lceil \log_2 t \rceil)$. □

6.4 Related work

In Theorem 1.2.2, we bound pathwidth by a function of treewidth in the absence of a subdivision of a large complete binary tree. In Corollary 1.1.5, we bound treedepth by a function of treewidth in the absence of a subdivision of a large complete binary tree and of a long path. Specifically, it is easy to generalize its proof to a slightly more general version:

Theorem 6.4.1. *Every graph with treewidth t that contains no subdivision of a complete binary tree of height h and no path of order 2^ℓ has treedepth $O(th\ell)$.*

It is natural to ask how large treedepth can be as a function of pathwidth when there is no long path. We offered the following conjecture.

Conjecture 6.4.2. *Every graph with pathwidth p that contains no path of order 2^ℓ has treedepth $O(p\ell)$.*

We are happy to report that this conjecture was recently resolved positively by Hatzel et al. [45].

This conjecture and Theorem 1.2.2 directly imply Theorem 6.4.1 offering an alternative proof (proof from [26] results in better constants though).

Chapter 7

Maximum average degree results

In this Chapter we will focus on the notion of *mad* or *maximum average degree*. Let us recall the main result, Theorem 1.3.1.

Theorem 1.3.1. *For every undirected simple graph G and a positive integer k such that* $\text{mad}(G) \geq k$ *there exists $S \subseteq V(G)$ such that $G[S]$ is $(k-1)$-degenerate and* $\text{mad}(G - S) \leq \text{mad}(G) - k$. *Moreover such S can be computed in polynomial time.*

First, we are going to prove this theorem and then we are going to discuss its multiple consequences.

7.1 Proof of Theorem 1.3.1

In order to prove Theorem 1.3.1 we are going to investigate a flow network that allows us to determine the value of mad in polynomial time. An example of such network can be found in [40], however we are going to use one adjusted to our own use.

Let us define a flow network $F(G, c)$ for given undirected graph G and any nonnegative real number c. The network will consist of one node for each $v \in V(G)$, one node for each $e \in E(G)$ denoted as v_e and two special nodes s and t, respectively source and sink. There will be three layers of directed edges in $F(G, c)$:

- The first layer – Edges of capacity one from s to each node v_e.
- The second layer – Edges of infinite capacity from each v_e where $e = uw \in E(G)$ to u and to w.

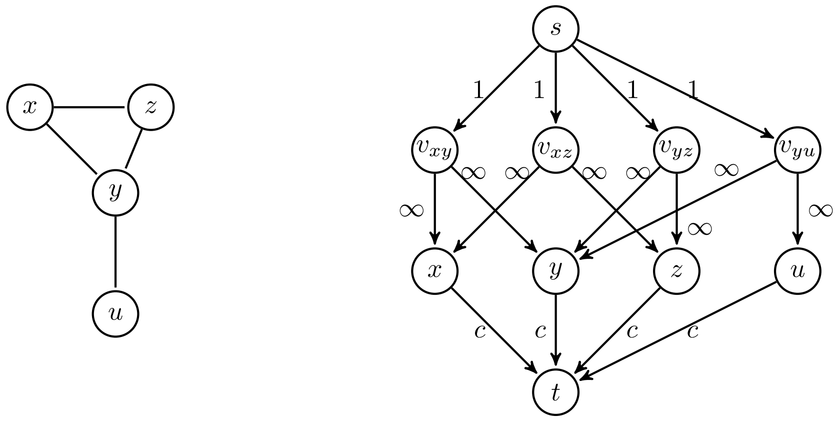

Figure 7.1: Example of graph G and flow network $F(G, c)$ corresponding to it.

- The third layer – Edges of capacity c from each $v \in V(G)$ to t.

Lemma 7.1.1. *For any graph G and any real number c, maximum flow between s and t in $F(G, c)$ is equal to $|E(G)|$ if and only if $2c \geq \operatorname{mad}(G)$.*

Proof. By the max-flow min-cut theorem we know that maximum flow in a graph G is equal to the minimum cut, so we are going to investigate structure of $s - t$ cuts in this graph. We refer to cuts as sets of edges. The set of all edges from first layer form an inclusion-wise minimal cut of weight $|E(G)|$. Since edges in the second layer have infinite capacities they surely do not belong to any minimum cut, so if maximum flow is smaller than $|E(G)|$ then there exists a minimum cut with some edges in third layer. Let us fix some minimal cut $C \subseteq E(F(G, c))$ and let W be the nonempty subset of $V(G)$ of all vertices w such that $\overrightarrow{wt}$ belongs to C. Let $H = G[W]$. If $e \notin E(H)$ then $\overrightarrow{sv_e}$ has to belong to C. Observe that all mentioned edges, that is $\overrightarrow{wt}$ for $w \in W$ and $\overrightarrow{sv_e}$ for $e \notin E(H)$ already form an $s - t$ cut. Its weight is $c|V(H)| + |E(G)| - |E(H)|$. If this value is less than $|E(G)|$ then we know that maximum flow in this graph is less than $|E(G)|$. However, if for any H this value is not smaller than $|E(G)|$ then we know that maxflow in this graph is $|E(G)|$.

We get that maxflow in this graph is smaller than $|E(G)|$ if and only if there exists $H \subseteq G$ such that $c|V(H)| + |E(G)| - |E(H)| < |E(G)| \Leftrightarrow c|V(H)| < |E(H)| \Leftrightarrow c < \frac{|E(H)|}{|V(H)|}$. The maximum value of $\frac{|E(H)|}{|V(H)|}$ equals $\frac{\operatorname{mad}(G)}{2}$, so we get that maxflow in $F(G, c)$ is equal to $|E(G)|$ if and only if $c \geq \frac{\operatorname{mad}(G)}{2}$, as desired. $\square$

Let us note that by using Lemma 7.1.1, observing that $\operatorname{mad}(G) = \frac{a}{b}$ for some $a, b \in \mathbb{Z}$ and $a \leq n^2, b \leq n$ and knowing that we can compute maximum flow in polynomial time, we can conclude that $\operatorname{mad}(G)$ can be computed in polynomial time.

Let us fix any graph G and denote $F := F(G, \frac{\operatorname{mad}(G)}{2})$. Let us define a directed graph G_f for a given $s-t$ flow f in F of capacity $|E(G)|$ by directing some of edges from G and discarding the rest. Flow f routes one unit of flow through each v_{uw}. Node v_{uw} has two outgoing edges to u and to w. If f sends more than $\frac{1}{2}$ unit of flow to w then in G_f we put directed edge $\overrightarrow{uw}$, similarly if f sends more than $\frac{1}{2}$ unit of flow to u we put edge $\overrightarrow{wu}$. Otherwise if f sends exactly $\frac{1}{2}$ unit to both u and w we simply discard this edge.

Lemma 7.1.2. *There exists flow f of capacity $|E(G)|$ in F such that G_f is acyclic. Moreover, it can be determined in polynomial time.*

Proof. From Lemma 7.1.1 we know that there exists at least one flow f between s and t of capacity $|E(G)|$. Let us take f such that number of edges in G_f is as small as possible. Suppose there is a cycle in G_f on vertices $c_1, c_2, \ldots, c_k$ respectively. Denote $c_{k+1} := c_1$ as we are dealing with a cycle. Let x be the minimum amount of flow that f sends through some edge $\overrightarrow{v_{c_i c_{i+1}} c_{i+1}}$ for some valid i. From definition of G_f we deduce that $x > \frac{1}{2}$. Let us define f' by decreasing flow f on edges $\overrightarrow{v_{c_i c_{i+1}} c_{i+1}}$ and increasing it on edges $\overrightarrow{v_{c_i c_{i+1}} c_i}$ by $x - \frac{1}{2}$. The amount of flow leaving and entering each vertex remains unchanged hence f' is also a flow. Moreover, f' still satisfies the capacity constraints. Flow f' for at least one vertex $v_{c_i c_{i+1}}$ sends exactly $\frac{1}{2}$ unit of flow through both edges outgoing from it, so at least one edge on the cycle is no longer present in $G_{f'}$ and edges outside the cycle remain unchanged when compared to G_f. This contradicts the assumption that G_f has the smallest possible number of edges, which implies the existence of such an f.

In order to compute such f in polynomial time let us take any f of capacity $|E(G)|$ in $F(G, \frac{\operatorname{mad}(G)}{2})$ (let us remind the reader that we can determine the value of $\operatorname{mad}(G)$ in the polynomial time). If G_f contains a cycle, we can detect one, determine the corresponding value of x and adjust f in the manner described in previous paragraph to remove this cycle. The number of edges in $G_{f'}$ is strictly smaller than in G_f, so we will not do this more than $|E(G)|$ times, which gives us an algorithm performing a polynomial number of operations. In order to omit dealing with rational numbers we can multiply all capacities in F by $2b$, where $\operatorname{mad}(G) = \frac{a}{b}$ for some coprime integers a, b. That concludes the description of a polynomial time algorithm determining the desired f.

$\square$

Let us fix f from the above lemma. We will present an algorithm in which:

- the routine $NoInEdges(H_f)$ returns any vertex from directed acyclic graph H_f which has no incoming edges (as the graph is acyclic there always exists at least one such vertex)
- the routine $KNeighborhood(H, S, k)$ takes as input a given graph H, a subset of its vertices S and an integer k, and returns the set of all vertices from H outside of S adjacent to at least k vertices from S.

Algorithm 1

```
function SOLVE($H, H_f, k$)
    $S \leftarrow \emptyset$
    while $H_f \neq \emptyset$ do
        $x \leftarrow NoInEdges(H_f)$
        $S \leftarrow S \cup \{x\}$
        $H_f \leftarrow H_f - \{x\} - KNeighborhood(H, S, k)$
return $S$
```

Theorem 7.1.3. *For positive integer k such that $\text{mad}(G) \geq k$ algorithm $Solve(G, G_f, k)$ returns a set $S \subseteq V(G)$ such that $G[S]$ is a $(k-1)$-degenerate and that $\text{mad}(G - S) \leq \text{mad}(G) - k$.*

Proof. First we argue that the graph induced on the set of vertices returned by the algorithm is $(k-1)$-degenerate. In each iteration the vertex x picked by the algorithm is adjacent to at most $k-1$ already picked vertices. So $G[S]$ is $(k-1)$-degenerate indeed.

To show that $\text{mad}(G-S) \leq \text{mad}(G)-k$ we just have to find a flow f' in graph $F' := F(G-S, \frac{\text{mad}(G)}{2} - \frac{k}{2})$ of value $|E(G-S)|$ thanks to Lemma 7.1.1. Observe that F' is a subgraph of F with capacities of edges on the third layer reduced by $\frac{k}{2}$. The flow f' has to saturate all edges from the first layer in order to have value $|E(G-S)|$. On the second layer we define f' using f, for each edge from the second layer of F' flow f' will send exactly the same amount of flow as f on corresponding edge in F. Now we just have to argue that the amount of flow sent by f' to any node between the second and third layer in F' is bounded by $\frac{\text{mad}(G)}{2} - \frac{k}{2}$ i.e. capacity of edge going from that node to sink. Each such node corresponds to vertex from $G - S$, so let us take arbitrary vertex $u \in V(G-S)$. During execution of the algorithm vertex u has been removed from H_f as incident to some k vertices already picked to S. Denote them $x_1, \ldots, x_k$ and let us consider arbitrary x_i. When the algorithm picked x_i from H_f, there were no incoming edges to x_i. In particular, in H_f there was no edge $\overrightarrow{ux_i}$. At that time u still belonged to H_f, so there was no edge $\overrightarrow{ux_i}$ even in G_f. Since u and x_i are adjacent in G, there was either an edge $\overrightarrow{x_iu}$ in G_f which means that flow f sends more than $\frac{1}{2}$ unit of flow from v_{ux_i} to u in F or there was no $\overrightarrow{x_iu}$ and $\overrightarrow{ux_i}$ which means that flow f sends exactly $\frac{1}{2}$ unit of flow from v_{ux_i} to u in F. Through node u in F flow f sends at most $\frac{\text{mad}(G)}{2}$ units of flow and for every $1 \leq i \leq k$ at least $\frac{1}{2}$ unit of flow comes from v_{ux_i} to u. Therefore flow going through u is decreased by at least $\frac{1}{2}$ unit per each x_i in F' what implies that f' sends at most $\frac{\text{mad}(G)}{2} - \frac{k}{2}$ units of flow to vertex u in F'.

□

What is more, procedure $Solve(G, G_f, k)$ can be trivially implemented in a polynomial time. Theorem 1.3.1 directly follows from Theorem 7.1.3. As two notable special cases we mention following corollaries:

Theorem 7.1.4. *For every undirected simple graph G there exists $I \subseteq V(G)$ such that I is an independent set and $\text{mad}(G - I) \leq \text{mad}(G) - 1$. Moreover such I can be computed in polynomial time.*

Theorem 7.1.5. *For every undirected simple graph G there exists $F \subseteq V(G)$ such that $G[F]$ is a forest and $\text{mad}(G - F) \leq \text{mad}(G) - 2$. Moreover such F can be computed in polynomial time.*

7.2 Reconfiguration graphs results

The reconfiguration graph $R_k(G)$ for a positive integer k and a graph G is the graph whose vertex set is the set of k-colourings of G and there is an edge between two colourings if and only if they differ by colour of exactly one vertex.

Let us recall the Cereceda's conjecture [15].

Conjecture 7.2.1. *Let k and $l \geq k+2$ be positive integers and let G be a k-degenerate graph on n vertices. Then $R_l(G)$ has diameter $\mathcal{O}(n^2)$.*

As we have already mentioned, our results imply the positive answer to the last remaining step in the outline of the proof by Eiben and Feghali [32] of the following fact:

Theorem 7.2.2. *Let $k \geq 2$ and $l \geq k+2$ be integers and let G be a graph on n vertices such that $\operatorname{mad}(G) < k+1$. Then $R_l(G)$ has diameter $k^{\mathcal{O}(k^2\sqrt{n})}$.*

Let us remind that any graph with $\operatorname{mad}(G) < k+1$ is k-degenerate, hence the setting of Theorem 7.2.2 can be seen as an easier setting of Conjecture 7.2.1. The obtained bound on the diameter is worse as well. Nonetheless, Theorem 7.2.2 seems interesting as previous proofs regarding the connectivity of such reconfiguration graphs yield only exponential bounds. However, this bound under a slightly less general assumption has already been improved by Feghali [34] to the polynomial one. Nevertheless, the idea of the proof of Theorem 7.2.2 transfers over to a novel analogous result for reconfiguration graphs of colouring viewed as homomorphism to a given graph H, which we shall present now.

For a given graph H, the *H-colouring* of a graph G is any homomorphism $f : V(G) \to V(H)$, that is, a function f such that if $uv \in E(G)$, then $f(u)f(v) \in E(H)$. We call $f(v)$ the colour of v. In particular, if H is loopless and $uv \in E(H)$, then it must hold that $f(u) \neq f(v)$. The *H-reconfiguration graph* of G is the graph whose vertex set consists of all H-colourings of G and two colourings are adjacent if and only if they differ by colour of exactly one vertex.

The main result of this section is the following theorem:

Theorem 7.2.3. *Let $k \geq 2$ be a positive integer and let $G = (V, E)$ be a graph on n vertices with $\operatorname{mad}(G) < k+1$ and let H be a graph such that $\Delta(\overline{H}) \leq d$ and $|V(H)| \geq (d+1)(k+1)+1$. Let α and β be two H-colourings of G. It is possible to get β from α by a sequence of $k^{\mathcal{O}(k^2\sqrt{n})}$ recolourings.*

Note that if we set $H = K_l$ and $d = 0$ we get the exact statement of Theorem 7.2.2, as regular colouring is exactly H-colouring for clique H. Moreover, to prove that fact we mainly follow the outline of the proof by Eiben and Feghali [32] of Theorem 7.2.2.

As a motivation for such generalization, we use the notion of circular colourings. We focus on the following description of the circular colourings [14], also known as (p, q)-colouring [84].

Definition 7.2.4. Let a and b, where $a \geq 2b$, be positive integers. The *circular clique* $G_{a,b}$ has the vertex set $\{0, 1, \ldots, a-1\}$ where ij is an edge when $b \leq |i-j| \leq a-b$. A homomorphism $\phi : G \to G_{a,b}$ is called a *circular colouring* in general, and an (a, b)-colouring of G for the specific pair (a, b).

We remark that $G_{a,1}$ is isomorphic to K_a and so an $(a, 1)$-colouring is simply an a-colouring.

Brewster and Noel [14] have proved that for a given graph G and positive integers a, b if $\frac{a}{b} \geq 2l+2$, where l is the degeneracy of G, then the reconfiguration graph of (a, b)-colourings of G is connected. However, the bound on its diameter that follows from their proof is exponential. Based on our H-colourings result, we are able to deduce the following bound on this diameter.

Corollary 7.2.5. *Let $k \geq 2$ and $G = (V, E)$ be a graph on n vertices with $\operatorname{mad}(G) < k+1$. Let $a \geq 2b$ be positive integers. If $\frac{a}{b} \geq 2k+2$, then the (a, b)-reconfiguration graph of G is connected and has diameter $k^{\mathcal{O}(k^2\sqrt{n})}$.*

Proof. We use Theorem 7.2.3 for $H := G_{a,b}$. The complement of $G_{a,b}$ is $(2b-2)$-regular, hence we set $d := 2b-2$. If $\frac{a}{b} \geq 2k+2$, then $V(H) = a \geq b(2k+2) = (d+2)(k+1) \geq (d+1)(k+1)+1$, hence all assumptions of that theorem are satisfied and the conclusion follows. □

7.2.1 Proof of the bound on the diameter of reconfiguration graphs of H-colourings of graphs with bounded maximum average degree.

The overall structure of proof of Theorem 7.2.3 consists of combining the generalization of Lemma 2 from [32] with Theorem 7.1.4 in the same way that Lemmas 8, 9 and 10 in [35] are combined to obtain Theorem 6 in [35]. It is the generalization of the outline of the analogous result for standard colourings from [32].

Lemma 7.2.6. *Let $k, d \geq 0$, and let G be the graph on n vertices such that $\operatorname{mad}(G) < k+1$. Let $\{u_1, \ldots, u_s\}$ be the set of vertices of G of degree at least $k+2$. Let H be a graph such that $\Delta(\overline{H}) \leq d$ and $|V(H)| \geq (d+1)(k+1)+1$ and let α be a H-colouring of G. Let F be any subset of $V(H)$ of size at most $d+1$. It is possible to reconfigure α to some H-colouring α' of G such that $\alpha'(V(G)) \subseteq V(H) - F$ by using at most $n^2 \prod_{i=1}^{s} deg(u_i)$ recolourings.*

Let us note that the case $d = 0$ coincides with Lemma 1 from [32] with the slight change in the assumption on G. The proof presented here is a generalization of its proof.

Proof. Since $\operatorname{mad}(G) < k+1$, we know that G is k-degenerate, hence we can fix its k-degenerate ordering $\sigma = v_1, \ldots, v_n$ and without loss of generality, let u_i appear before u_j in σ whenever $i < j$. In the following, we will describe an algorithm *Recolour*(h, F_h), which given an index $h \in [n]$ and the subset $F_h \subseteq V(H)$ of forbidden colours for v_h such that $|F_h| \leq d+1$, outputs a sequence of recolourings with the following properties:

- for $i < h$, v_i is not recoloured,
- for $i \geq h$, v_i is recoloured at most $\prod_{j=l}^{s} deg(u_j)$ times, where u_l is the first vertex of degree at least $k+2$ with index at least h in σ
- v_h ends up with a colour from $V(H) - F_h$ (in particular, if $f(v_h) \in V(H) - F_h$, then an empty sequence is a feasible one, where $f(v_h)$ denotes the current colour of v_h)

Notice that the algorithm takes at most $n \prod_{i=1}^{s} deg(u_i)$ recolourings to recolour v_h. Hence, by repeatedly calling *Recolour*(i, F) for $i = 1, \ldots, n$, we obtain the colouring α' in which colours from F do not appear by using at most $n^2 \prod_{i=1}^{s} deg(u_i)$ recolourings, as required.

Given $h \in [n]$, k-degenerate ordering $\sigma = v_1, \ldots, v_n$ of G and the set F_h of forbidden colours, the algorithm *Recolour*(h, F_h) works as follows:

1. If $f(v_h) \notin F_h$, then terminate.
2. If v_h has degree at most k, then
    - Let c be a colour not belonging to $N_{\overline{H}}[f(N(v_h))] \cup F_h$. Such colour exists since $\Delta(\overline{H}) \leq d$, so $|N_{\overline{H}}[f(N(v_h))] \cup F_h| \leq (d+1)k + (d+1) < |V(H)|$.
    - Recolour v_h to c.
3. If v_h has degree at least $k+1$, then
    - Let c be a colour not belonging to $N_{\overline{H}}[f(Z)] \cup F_h$, where Z is the set consisting of first k neighbours of v_h in ordering σ. Such colour exists since $\Delta(\overline{H}) \leq d$, so $|N_{\overline{H}}[f(Z)] \cup F_h| \leq (d+1)k + (d+1) < |V(H)|$.
    - Let $v_{i_1}, \ldots, v_{i_t}$ be the neighbours of v_h outside Z with $i_1 < i_2 < \ldots < i_t$. Note that $h < i_1$, since G is k-degenerate.
    - For each $j \in [t]$ in the ascending order call *Recolour*$(i_j, N_{\overline{H}}[c])$.
    - Recolour v_h to c.

It is clear that this algorithm is correct. To estimate the number of used recolourings, it is sufficient to observe that the recursion branches only on vertices of degree at least $k+2$. □

Lemma 7.2.7. *Let $k \geq 1, d \geq 0$ and let G be a graph on n vertices and with $\operatorname{mad}(G) < k+1$. Let H be a graph such that $\Delta(\overline{H}) \leq d$ and $|V(H)| \geq (d+1)(k+1)+1$ and let α be a H-colouring of G. Let F be a subset of $V(H)$ of size at most $d+1$. It is possible to reconfigure α to some H-colouring α' of G such that $\alpha'(V(G)) \subseteq V(H) - F$ by $\max(k,2)^{\mathcal{O}(k^2\sqrt{n})}$ recolourings.*

Let us note again that the case $d = 0$ coincides with Lemma 2 from [32] and that the proof presented here is a generalization of its proof.

Proof. We will call H-colourings of G *small* if they do not use colours from F. We will denote $k' = \max(k,2)$.

We shall prove by the induction on the size $n := |V(G)|$ that we can reconfigure α to a small H-colouring α', such that each vertex in G is recoloured at most $n^2 \cdot k'^{9k'^2\sqrt{n}}$ times, which implies the lemma.

As the base case we distinguish graphs P on p vertices such that $\operatorname{mad}(P) < k+1$ that contain at most $2(k+1)\sqrt{p}$ vertices of degree at most k. Let $\{u_1, \ldots, u_s\}$ be a set of vertices of P of degree at least $k+2$. In the case of such graphs $\prod_{i=1}^{s} deg(u_i)$ is bounded from above by $(2(k+1))^{2k(k+1)\sqrt{p}} \leq (k'^3)^{3k'^2\sqrt{p}} = k'^{9k'^2\sqrt{p}}$ as proven by Eiben and Feghali as part of the proof of Lemma 2 in [32]. Thus, in this case we can use algorithm from Lemma 7.2.6 and prove that we can get α' using at most $p^2 \prod_{i=1}^{s} deg(u_i) \leq p^2 k'^{9k'^2\sqrt{p}}$ recolourings in total, in particular the number of recolourings of each particular vertex is bounded by $p^2 k'^{9k'^2\sqrt{p}}$.

For the inductive step, suppose that G contains more than $2(k+1)\sqrt{n}$ vertices of degree at most k and that we can reconfigure any subgraph P of G with $p < n$ vertices to some small H-colouring α_P such that each vertex gets recoloured at most $p^2 k'^{9k'^2\sqrt{p}}$ times. Let S be an independent set in G of size at least $2\sqrt{n}$ containing only vertices of degree at most k. Note that G is k-degenerate, so it can be partitioned into $k+1$ independent sets and one of them has to contain at least $2\sqrt{n}$ vertices of degree at most k, so such S exists. Using an inductive argument, we can recolour the graph $P = G - S$ to some small H-colouring through a recolouring sequence R. We can extend this sequence of recolourings in P to a sequence in G in the following way. Let a recolouring of $v \in V(P)$ from c_1 to c_2 be one of the recolouring operations from R and let $u \in S$ be a neighbour of v. Let C be the sum of $\{c_1, c_2\}$ and colours present in the neighbours of u in P other than v. We have that $|C| \leq k+1$, so $N_{\overline{H}}[C] \leq (d+1)(k+1) < |V(H)|$, so there exists a colour $c \in V(H) - N_{\overline{H}}[C]$. In the reconfiguration sequence extended to G we put an operation of recolouring u to c before recolouring v. This way we get a valid recolouring sequence for G. Finally, we recolour each vertex $u \in S$ to any colour not belonging to $F \cup N_{\overline{H}}[f(N_G(u))]$. It is possible as $|F \cup N_{\overline{H}}[f(N_G(u))]| \leq (d+1)(k+1) < |V(H)|$. For each vertex outside S we recoloured it exactly as many times as in R, whereas for each vertex in S we recoloured it at most as many times as all its neighbours were recoloured in total in R plus one.

Let $g(n)$ be the maximum number of times that a vertex of a graph on n vertices is recoloured in this process. In the base case we require at most $n^2 k'^{9k'^2\sqrt{n}}$ recolourings, while in the inductive step case we require at most $k \cdot g(\lfloor n - 2\sqrt{n} \rfloor) + 1$ recolourings, hence we have $g(n) \leq \max(n^2 \cdot k'^{9k'^2\sqrt{n}}, k \cdot g(\lfloor n - 2\sqrt{n} \rfloor) + 1)$. However, simple calculations show that $k \cdot k'^{9k'^2\sqrt{n-2\sqrt{n}}} + 1 \leq k'^{9k'^2\sqrt{n}}$ (as $\sqrt{n} > \sqrt{n - 2\sqrt{n}} + 1$), so $n^2 \cdot k \cdot k'^{9k'^2\sqrt{n-2\sqrt{n}}} + 1 \leq n^2 \cdot k'^{9k'^2\sqrt{n}}$, hence $g(n) \leq n^2 \cdot k'^{9k'^2\sqrt{n}}$, which concludes this proof that each vertex can be recoloured at most $n^2 \cdot k'^{9k'^2\sqrt{n}}$ times. Thus, in total we get a sequence of at most $n^3 \cdot k'^{9k'^2\sqrt{n}} = \max(2,k)^{\mathcal{O}(k^2\sqrt{n})}$ recolourings. □

Remark. If we were a bit more meticulous, then in Lemma 7.2.6 it would be possible to get the bound of $\mathcal{O}(n^2 \prod_{i=1}^{s}(deg(u_i) - k))$ instead of $\mathcal{O}(n^2 \prod_{i=1}^{s} deg(u_i))$. In consequence, it would be possible to improve the bound from Lemma 7.2.7 to $2^{\mathcal{O}(k^2\sqrt{n})}$, but proving that would lead to significantly longer exposition and the improvement would not be that significant, hence we decided not to present it.

Now we are ready to prove Theorem 7.2.3.

Proof. Let $k \geq 0$ be a positive integer and let $G = (V, E)$ be a graph on n vertices with $\operatorname{mad}(G) < k+1$ and let H be a graph such that $\Delta(\overline{H}) \leq d$ and $|V(H)| \geq (d+1)(k+1)+1$. We will define a H-colouring of G called γ and then prove that any H-colouring δ can be reconfigured to γ by a sequence of at most $\max(2,k)^{\mathcal{O}(\max(k^2,1)\sqrt{n})}$ recolourings, which clearly implies the thesis by using twice this statement for $\delta = \alpha$ and for $\delta = \beta$.

We will prove this inductively on k. The base case $k = 0$ is trivial: if $\mathrm{mad}(G) < 1$, then G is an edgeless graph and every colouring is valid. We can set γ as an arbitrary colouring of G. Now we can recolour each vertex directly from colour from δ to colour from γ using at most n recolourings in total.

Now we present the inductive step. Let us consider an independent set I from Theorem 7.1.4 such that $mad(G - I) < k$ and any colour $u \in V(H)$. For each $v \in I$ we set $\gamma(v) = u$. Thanks to Lemma 7.2.7 for $F := N_{\overline{H}}[u]$, we are able to reconfigure δ to a colouring δ' that does not use colours from $N_{\overline{H}}[u]$. Afterwards, we reconfigure δ' to δ'' by recolouring each vertex from I to the colour u. We define $G' = G - I$ and $H' = H - N_{\overline{H}}[u]$. Graphs G' and H' meet the assumptions of the inductive hypothesis for $k' = k - 1$. So there exists γ' independent of δ'' and a sequence of recolourings R' configuring δ'' to γ'. Now we just need to concatenate all those reconfiguration sequences and set $\gamma(v) = \gamma'(v)$ for $v \in G'$. This concatenation yields a valid reconfiguration sequence because colour c is connected to all colours from H'.

To build the final reconfiguration sequence we used the construction from Lemma 7.2.7 exactly k times. Apart from that, each vertex is recoloured at most once, to its final colour, so in total we used at most $n + n \cdot max(2, k)^{\mathcal{O}(k^2\sqrt{n})} \leq \max(2, k)^{\mathcal{O}(\max(k^2,1)\sqrt{n})}$ recolourings, as desired.

□

It is worth noting that all parts of this proof were constructive, hence we can compute such sequence of recolourings in $k^{\mathcal{O}(k^2\sqrt{n})}$ time and polynomial space.

7.3 Conclusions and open problems

Our main results imply many results for some specific classes of graphs as a direct consequence and here we mention a few of them.

Following folklore fact will come in handy in deriving some of the consequences:

Fact 3. For every planar graph G we have $(\mathrm{mad}(G) - 2)(g(G) - 2) < 4$.

Based on Theorems 7.1.4, we are able to improve Theorem 1 from [29] and one of its consequences.

Theorem 7.3.1 ([29])**.** *Let M be a real number such that $M < 3$. Let $d \geq 0$ be an integer and let G be a graph with $\mathrm{mad}(G) < M$. If $d \geq \frac{2}{3-M} - 2$, then $V(G)$ can be partitioned into $A \uplus B$ such that $G[A]$ is an independent set and $G[B]$ is a forest with maximum degree at most d.*

We are able to strengthen this to the following theorem:

Theorem 7.3.2. *Let M be a real number such that $M < 3$. Let $d \geq 0$ be an integer and let G be a graph with $\mathrm{mad}(G) < M$. If $d \geq \frac{2}{3-M} - 2$, then $V(G)$ can be partitioned into $A \uplus B$ such that $G[A]$ is an independent set and $G[B]$ is a forest whose connected components have size at most $d + 1$.*

Proof. By using Theorem 7.1.4 one can partition $V(G)$ into $A \uplus B$ such that $G[A]$ is an independent set and $\mathrm{mad}(G[B]) < M - 1$. Let us bound $M - 1$ from above using assumed inequalities:

$$d \geq \frac{2}{3-M} - 2 \;\Rightarrow\; d+2 \geq \frac{2}{3-M} \;\Rightarrow\; 3 - M \geq \frac{2}{d+2} \;\Rightarrow\; M - 1 \leq 2 - \frac{2}{d+2}.$$

$G[B]$ does not contain a cycle because $\mathrm{mad}(G[B]) < M-1 \leq 2-\frac{2}{d+2} < 2$. Assume that $G[B]$ contains a tree T on $d+2$ vertices as a subgraph. Then $\mathrm{mad}(G[B]) \geq \frac{2|E(T)|}{|V(T)|} = \frac{2(d+1)}{d+2} = 2 - \frac{2}{d+2} \geq M - 1 > \mathrm{mad}(G[B])$. Note that if $G[B]$ contains a component which is a tree on at least $d + 2$ vertices then it contains tree on exactly $d + 2$ vertices as a subgraph, hence shown contradiction finishes the proof that connected components of $G[B]$ are trees on at most $d + 1$ vertices.

□

Dross et al. [29] use their theorem and Fact 3 to deduce the following corollary:

Corollary 7.3.3. *For every planar graph G with $g(G) \geq 10$, the vertex set $V(G)$ can be partitioned into $A \uplus B$ such that A is an independent set and $G[B]$ is a forest with maximum degree* 2.

We are able to strengthen this to the following corollary:

Corollary 7.3.4. *For every planar graph G with $g(G) \geq 10$, the vertex set $V(G)$ can be partitioned into $A \uplus B$ such that A is an independent set and $G[B]$ is a forest whose connected components have size at most* 3.

Borodin et al. proved in [11] that the vertex set of any planar graph with $g(G) \geq 7$ admits a partition into an independent set and a set that induces graph with maximum degree at most 4. Dross et al. proved in [29] that the vertex set of any planar graph with $g(G) \geq 7$ admits a partition into an independent set and a set that induces forest of max degree at most 5. In Corollary 7.3.5 we add another partition result for the class of planar graphs with girth at least 7.

Corollary 7.3.5. *For every planar graph G with $g(G) \geq 7$, the vertex set $V(G)$ can be partitioned into $A \uplus B$ such that A is an independent set and $G[B]$ is a forest where every connected component has at most* 9 *vertices.*

Proof. Since $g(G) \geq 7$ we deduce that $\mathrm{mad}(G) < 1 + \frac{9}{5}$, so based on Theorem 7.1.4 we get that there exist A and B such that $V(G) = A \uplus B$, A is an independent set and $\mathrm{mad}(G[B]) < \frac{9}{5}$. It can be readily verified that class of graphs with value of their mad smaller than $\frac{9}{5}$ is class of graphs which are forests with connected components of size at most 9. □

Recently, independently of our work, Cranston and Yancey [24] improved Corollaries 7.3.4 and 7.3.5. Namely, they claim that every planar graph G of girth at least 9 (resp. 8, 7) has a partition of $V(G)$ into an independent set I and a set F such that $G[F]$ is a forest with each component of order at most 3 (resp. 4, 6).

Apart from that, based on Theorem 7.1.4 and 7.1.5 and Fact 3 we are able to deduce following corollaries:

Corollary 7.3.6. *For every planar graph G, the vertex set $V(G)$ can be partitioned into $A \uplus B \uplus C$ such that $G[A], G[B], G[C]$ are forests.*

Proof. Every planar graph satisfies $\mathrm{mad}(G) < 6$, so using Theorem 7.1.5 we can partition $V(G)$ into A and D such that $\mathrm{mad}(G[A]) < 2$ and $\mathrm{mad}(G[D]) < 4$ and then using Theorem 7.1.5 again we can partition D into B and C such that $\mathrm{mad}(G[B]) < 2$ and $\mathrm{mad}(G[C]) < 2$. Hence $G[A], G[B], G[C]$ are forests. □

Corollary 7.3.7. *For every planar graph G without triangles, the vertex set $V(G)$ can be partitioned into $A \uplus B$ such that $G[A], G[B]$ are forests.*

Proof. Based on Fact 3 we know that if G has no triangles then $g(G) \geq 4 \Rightarrow g(G) - 2 \geq 2 \Rightarrow \mathrm{mad}(G) < 4$. Therefore using Theorem 7.1.5 we deduce that there exist A, B such that $V(G) = A \uplus B$ and $G[A], G[B]$ are forests. □

Corollary 7.3.8. *For every planar graph G without cycles of length* 3 *and* 4*, the vertex set $V(G)$ can be partitioned into $A \uplus B$ such that $G[A]$ is a forest and $\Delta(G[B]) \leq 1$.*

Proof. Since $g(G) \geq 5$ we deduce that $\mathrm{mad}(G) < 2 + \frac{4}{3}$, so based on Theorem 7.1.5 we get that there exist A and B such that $V(G) = A \uplus B$ and $\mathrm{mad}(G[A]) < 2$ and $\mathrm{mad}(G[B]) < \frac{4}{3}$. Therefore $G[A]$ is a forest and $\Delta(G[B]) \leq 1$, because if $G[B]$ contains a vertex with degree ≥ 2 then this vertex together with its two neighbours induce a graph with mad at least $\frac{4}{3}$. □

Corollary 7.3.9. *For every planar graph G with $g(G) \geq 6$ its vertex set $V(G)$ can be partitioned into $A \uplus B$ such that $G[A]$ is a forest and B is an independent set.*

Proof. Since $g(G) \geq 6$ we deduce that $\mathrm{mad}(G) < 3$, so based on either Theorem 7.1.5 or Theorem 7.1.4 we get that there exist A and B such that $V(G) = A \uplus B$ and $\mathrm{mad}(G[A]) < 2$ and $\mathrm{mad}(G[B]) < 1$. Therefore $G[A]$ is a forest and B is an independent set. □

However, Corollaries 7.3.6, 7.3.7, 7.3.8 and 7.3.9 have already been proven and even improved before. Corollary 7.3.6 was proven in [16] and later improved in [74]. An improved version of Corollary 7.3.7 was proven in [75]. Theorem improving both Corollaries 7.3.8 and 7.3.9 was proven in [9].

7.3.1 Open problem

As the main open problem in the area of partitionability of graphs with bounded *mad*, we recall the following conjecture.

Conjecture 7.3.10. *For every graph G and positive real numbers c_1, c_2 if $\mathrm{mad}(G) < c_1 + c_2$ then there exists a partition of the vertex set $V(G) = A \uplus B$ such that $\mathrm{mad}(G[A]) < c_1$ and $\mathrm{mad}(G[B]) < c_2$.*

Our main result shows that this conjecture is true for $c_2 \in \{1, 2\}$. Moreover, since for positive k we have that k-degenerate graphs fulfill $\mathrm{mad}(G) < 2k$ we can deduce that for every integer $k \geq 2$ and a graph G that satisfies $\mathrm{mad}(G) < c_1 + k$ there exists a partition of the vertex set $V(G) = A \uplus B$ such that $\mathrm{mad}(G[A]) < c_1$ and $\mathrm{mad}(G[B]) < 2k - 2$.

Bibliography

[1] *Open problems for the Barbados graph theory workshop 2019.* Edited by L. Cook and S. Spirkl, Available at: `https://sites.google.com/site/sophiespirkl/open-problems/2019-open-problems-for-the-barbados-graph-theory-workshop`.

[2] M. Agrawal, N. Kayal, and N. Saxena. Primes is in P. *Annals of Mathematics*, 160, 9 2002.

[3] A. Bassa, J. Burns, J. Campbell, A. Deshpande, J. Farley, M. Halsey, S.-Y. Ho, D. Kleitman, S. Michalakis, P.-O. Persson, P. Pylyavskyy, L. Rademacher, A. Riehl, M. Rios, J. Samuel, B. Tenner, A. Vijayasarathy, and L. Zhao. Partitioning a planar graph of girth 10 into a forest and a matching. *Studies in Applied Mathematics*, 124:213 – 228, 01 2010.

[4] P. Bellenbaum and R. Diestel. Two short proofs concerning tree-decompositions. *Combinatorics, Probability and Computing*, 11(6):541–547, 2002.

[5] D. Bienstock, N. Robertson, P. Seymour, and R. Thomas. Quickly excluding a forest. *Journal of Combinatorial Theory, Series B*, 52(2):274–283, 1991.

[6] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[7] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016.

[8] B. Bollobás and B. Manvel. Optimal vertex partitions. *Bulletin of the London Mathematical Society*, 11(2):113–116, 1979.

[9] O. V. Borodin and A. Glebov. On the partition of a planar graph of girth 5 into an empty and an acyclic subgraph. *Diskretnyj Analiz i Issledovanie Operatsij. Seriya 1*, 01 2001.

[10] O. V. Borodin and A. V. Kostochka. Vertex decompositions of sparse graphs into an independent vertex set and a subgraph of maximum degree at most 1. *Siberian Mathematical Journal*, 52(5):796, Oct 2011.

[11] O. V. Borodin and A. V. Kostochka. Defective 2-colorings of sparse graphs. *Journal of Combinatorial Theory, Series B*, 104:72 – 80, 2014.

[12] O. V. Borodin, A. V. Kostochka, N. N. Sheikh, and G. Yu. Decomposing a planar graph with girth 9 into a forest and a matching. *Eur. J. Comb.*, 29(5):1235–1241, 2008.

[13] N. Bousquet and M. Heinrich. A polynomial version of cereceda's conjecture. *Journal of Combinatorial Theory, Series B*, 155:1–16, 2022.

[14] R. Brewster and J. Noel. Mixing homomorphisms, recolorings, and extending circular precolorings. *Journal of Graph Theory*, 80(3):173–198, 2015.

[15] L. Cereceda. Mixing graph colourings. PhD thesis, London School of Economics, 2007.

[16] G. Chartrand, H. V. Kronk, and C. E. Wall. The point-arboricity of a graph. *Israel Journal of Mathematics*, 6(2):169–175, Apr 1968.

[17] C. Chekuri and J. Chuzhoy. Polynomial bounds for the grid-minor theorem. *Journal of the ACM*, 63(5):Article No. 40, 2016.

[18] J. Chen, W. Czerwiński, Y. Disser, A. E. Feldmann, D. Hermelin, W. Nadara, M. Pilipczuk, M. Pilipczuk, M. Sorge, B. Wróblewski, and A. Zych-Pawlewicz. Efficient fully dynamic elimination forests with applications to detecting long paths and cycles. arXiv:2006.00571, 2020.

[19] J. Chen, W. Czerwiński, Y. Disser, A. E. Feldmann, D. Hermelin, W. Nadara, M. Pilipczuk, M. Pilipczuk, M. Sorge, B. Wróblewski, and A. Zych-Pawlewicz. Efficient fully dynamic elimination forests with applications to detecting long paths and cycles. In *2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 796–809. SIAM, 2021.

[20] M. Chen, W. Yu, and W. Wang. On the vertex partitions of sparse graphs into an independent vertex set and a forest with bounded maximum degree. *Applied Mathematics and Computation*, 326:117–123, 2018.

[21] J. Chuzhoy and Z. Tan. Towards tight(er) bounds for the excluded grid theorem. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1445–1464. SIAM, 2019.

[22] J. Chuzhoy and Z. Tan. Towards tight(er) bounds for the Excluded Grid Theorem. *Journal of Combinatorial Theory, Series B*, 146:219–265, 2021.

[23] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[24] D. W. Cranston and M. P. Yancey. Vertex partitions into an independent set and a forest with each component small. *SIAM Journal on Discrete Mathematics*, 35(3):1769–1791, 2021.

[25] 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 *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011*, pages 150–159. IEEE Computer Society, 2011.

[26] W. Czerwiński, W. Nadara, and M. Pilipczuk. Improved bounds for the excluded-minor approximation of treedepth. *SIAM J. Discret. Math.*, 35(2):934–947, 2021.

[27] R. Diestel. *Graph Theory, 2nd Edition.* Graduate texts in mathematics. Springer, 2000.

[28] G. Ding, B. Oporowski, D. P. Sanders, and D. Vertigan. Partitioning graphs of bounded tree-width. *Combinatorica*, 18(1):1–12, Jan 1998.

[29] F. Dross, M. Montassier, and A. Pinlou. Partitioning sparse graphs into an independent set and a forest of bounded degree. *Electr. J. Comb.*, 25(1):P1.45, 2018.

[30] Z. Dvořák, M. Kupec, and V. Tůma. A dynamic data structure for MSO properties in graphs with bounded tree-depth. In *Proceedings of the 22nd Annual European Symposium on Algorithms, ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 2014.

[31] Z. Dvořák, A. C. Giannopoulou, and D. M. Thilikos. Forbidden graphs for tree-depth. *European Journal of Combinatorics*, 33(5):969–979, 2012. EuroComb '09.

[32] E. Eiben and C. Feghali. Toward Cereceda's conjecture for planar graphs. *Journal of Graph Theory*, 94(2):267–277, 2020.

[33] G. Fan, Y. Li, N. Song, and D. Yang. Decomposing a graph into pseudoforests with one having bounded degree. *Journal of Combinatorial Theory, Series B*, 115:72 – 95, 2015.

[34] C. Feghali. Reconfiguring colorings of graphs with bounded maximum average degree. *Journal of Combinatorial Theory, Series B*, 147:133–138, 2021.

[35] C. Feghali, M. Johnson, and D. Paulusma. A reconfigurations analogue of Brooks' theorem and its consequences. *Journal of graph theory.*, 83(4):340–358, December 2016.

[36] U. Feige, M. Hajiaghayi, and J. R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, 2008.

[37] P. Fraigniaud and N. Nisse. Connected treewidth and connected graph searching. In J. R. Correa, A. Hevia, and M. Kiwi, editors, *LATIN 2006: Theoretical Informatics*, volume 3887 of *Lecture Notes in Computer Science*, pages 479–490. Springer, Berlin, Heidelberg, 2006.

[38] M. Fürer and H. Yu. Space saving by dynamic algebraization. In *9th International Computer Science Symposium in Russia, CSR 2014*, volume 8476 of *Lecture Notes in Computer Science*, pages 375–388. Springer, 2014.

[39] J. Gajarský, S. Kreutzer, J. Nešetřil, P. Ossona de Mendez, M. Pilipczuk, S. Siebertz, and S. Toruńczyk. First-order interpretations of bounded expansion classes. *ACM Trans. Comput. Log.*, 21(4):29:1–29:41, 2020.

[40] A. V. Goldberg. Finding a maximum density subgraph. Technical Report UCB/CSD-84-171, EECS Department, University of California, Berkeley, 1984.

[41] C. Groenland, G. Joret, W. Nadara, and B. Walczak. Approximating pathwidth for graphs of small treewidth. *ACM Trans. Algorithms*, dec 2022. Just Accepted.

[42] M. Grohe and S. Kreutzer. Methods for algorithmic meta theorems. In *AMS-ASL Joint Special Session on Model Theoretic Methods in Finite Combinatorics*, volume 558 of *Contemporary Mathematics*, pages 181–206. American Mathematical Society, 2009.

[43] M. Grohe, S. Kreutzer, R. Rabinovich, S. Siebertz, and K. Stavropoulos. Coloring and covering nowhere dense graphs. *SIAM Journal on Discrete Mathematics*, 32(4):2467–2481, 2018.

[44] L. Grout and B. Moore. The pseudoforest analogue for the strong nine dragon tree conjecture is true. *Journal of Combinatorial Theory, Series B*, 145:433 – 449, 2020.

[45] M. Hatzel, G. Joret, P. Micek, M. Pilipczuk, T. Ueckerdt, and B. Walczak. Tight bound on treedepth in terms of pathwidth and longest path, 2023.

[46] W. He, X. Hou, K. Lih, J. Shao, W. Wang, and X. Zhu. Edge-partitions of planar graphs and their game coloring numbers. *Journal of Graph Theory*, 41(4):307–317, 2002.

[47] F. Hegerfeld and S. Kratsch. Solving connectivity problems parameterized by treedepth in single-exponential time and polynomial space. In *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020*, volume 154 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2020.

[48] R. Hickingbotham. Graph minors and tree decompositions. Bachelor's thesis, School of Mathematics, Monash University, Melbourne, 2019.

[49] W. A. Horn. Three results for trees, using mathematical induction. *Journal of Research of the National Bureau of Standards, Series B: Mathematical Sciences*, 76B(1–2):39–43, 1972.

[50] K. Kawarabayashi and B. Rossman. A polynomial excluded-minor approximation of treedepth. In A. Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 234–246. SIAM, 2018.

[51] K. Kawarabayashi and B. Rossman. A polynomial excluded-minor approximation of treedepth. *Journal of the European Mathematical Society*, 24(4):1449–1470, 2022.

[52] H. A. Kierstead and D. Yang. Orderings on graphs and game coloring number. *Order*, 20(3):255–264, 2003.

[53] S.-J. Kim, A. V. Kostochka, D. B. West, H. Wu, and X. Zhu. Decomposition of sparse graphs into forests and a graph with bounded degree. *Journal of Graph Theory*, 74(4):369–391, 2013.

[54] M. Kopreski and G. Yu. Maximum average degree and relaxed coloring. *Discrete Mathematics*, 340(10):2528–2530, 2017.

[55] T. Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 184–192. IEEE, 2021.

[56] L. Kowalik, M. Mucha, W. Nadara, M. Pilipczuk, M. Sorge, and P. Wygocki. The PACE 2020 Parameterized Algorithms and Computational Experiments challenge: Treedepth. In *15th International Symposium on Parameterized and Exact Computation, IPEC 2020*, volume 180 of *LIPIcs*, pages 37:1–37:18. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2020.

[57] J. Kun, M. P. O'Brien, and B. D. Sullivan. Treedepth bounds in linear colorings. *CoRR*, abs/1802.09665, 2018.

[58] L. Lovász. On decomposition of graphs. *Studia Sci. Math. Hungar.*, 1:237–238, 1966.

[59] E. A. Marshall and D. R. Wood. Circumference and pathwidth of highly connected graphs. *Journal of Graph Theory*, 79(3):222–232, 2015.

[60] P. Mihók. Minimal reducible bounds for the class of k-degenerate graphs. *Discrete Mathematics*, 236(1):273 – 279, 2001. Graph Theory.

[61] W. Nadara, M. Pilipczuk, and M. Smulewicz. Computing treedepth in polynomial space and linear fpt time. *CoRR*, abs/2205.02656, 2022.

[62] W. Nadara, M. Pilipczuk, and M. Smulewicz. Computing treedepth in polynomial space and linear FPT time. In S. Chechik, G. Navarro, E. Rotenberg, and G. Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 79:1–79:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[63] W. Nadara and M. Smulewicz. Decreasing the maximum average degree by deleting an independent set or a \$d\$-degenerate subgraph. *Electron. J. Comb.*, 29(1), 2022.

[64] J. Nederlof, M. Pilipczuk, C. M. F. Swennenhuis, and K. Węgrzycki. Hamiltonian Cycle parameterized by treedepth in single exponential time and polynomial space. In *46th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 12301 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2020.

[65] J. Nesetril and P. O. de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006.

[66] J. Nesetril and P. O. de Mendez. On low tree-depth decompositions. *Graphs and Combinatorics*, 31(6):1941–1963, 2015.

[67] J. Nešetřil and P. Ossona de Mendez. Grad and classes with bounded expansion II. Algorithmic aspects. *Eur. J. Comb.*, 29(3):777–791, 2008.

[68] J. Nešetřil and P. Ossona de Mendez. *Sparsity — Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

[69] J. Nešetřil and P. Ossona de Mendez. A distributed low tree-depth decomposition algorithm for bounded expansion classes. *Distributed Comput.*, 29(1):39–49, 2016.

[70] M. P. O'Brien and B. D. Sullivan. Experimental evaluation of counting subgraph isomorphisms in classes of bounded expansion. *CoRR*, abs/1712.06690, 2017.

[71] M. Pilipczuk and S. Siebertz. Polynomial bounds for centered colorings on proper minor-closed graph classes. *J. Comb. Theory, Ser. B*, 151:111–147, 2021.

[72] M. Pilipczuk, S. Siebertz, and S. Toruńczyk. Parameterized circuit complexity of model-checking on sparse structures. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 789–798. ACM, 2018.

[73] M. Pilipczuk and M. Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM Trans. Comput. Theory*, 9(4):18:1–18:36, 2018.

[74] K. S. Poh. On the linear vertex-arboricity of a planar graph. *Journal of Graph Theory*, 14(1):73–75, 1990.

[75] A. Raspaud and W. Wang. On the vertex-arboricity of planar graphs. *European Journal of Combinatorics*, 29(4):1064 – 1075, 2008. Homomorphisms: Structure and Highlights.

[76] F. Reidl, P. Rossmanith, F. Sánchez Villaamil, and S. Sikdar. A faster parameterized algorithm for treedepth. In *41st International Colloquium on Automata, Languages, and Programming, ICALP 2014*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014.

[77] N. Robertson and P. Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986.

[78] N. Robertson and P. D. Seymour. Graph minors I-XXII. 1982-2010.

[79] J. B. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6(1):64 – 94, 1962.

[80] A. A. Schäffer. Optimal node ranking of trees in linear time. *Inf. Process. Lett.*, 33(2):91–96, 1989.

[81] P. Scheffler. *Die Baumweite von Graphen als ein Maß für die Kompliziertheit algorithmischer Probleme*. PhD thesis, Akademie der Wissenschaften der DDR, Berlin, 1989.

[82] D. R. Wood. Acyclic, star and oriented colourings of graph subdivisions. *Discrete Mathematics and Theoretical Computer Science*, 7(1):37–50, 2005.

[83] D. R. Wood. Personal communication, 2013.

[84] X. Zhu. Recent developments in circular colouring of graphs. In M. Klazar, J. Kratochvíl, M. Loebl, J. Matoušek, P. Valtr, and R. Thomas, editors, *Topics in Discrete Mathematics*, pages 497–550, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[85] X. Zhu. Colouring graphs with bounded generalized colouring number. *Discret. Math.*, 309(18):5562–5568, 2009.