ABOUT TREEDEPTH AND RELATED NOTIONS

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

DIPLOM-INFORMATIKER
FERNANDO SÁNCHEZ VILLAAMIL

aus

AACHEN

Berichter: Univ.-Prof. Dr. Peter Rossmanith
Univ.-Prof. Dr. Henning Fernau

Tag der mündlichen Prüfung: 10.11.2017

F. Sánchez Villaamil: *About Treedepth and Related Notions*.

To the void

There seems to be an inborn drive in all human beings not to live in a steady emotional state, which would suggest that such a state is not tolerable to most people. [...] it's the same old lesson: everything in this life—I repeat, everything—is more trouble than it's worth. And simply being alive is the basic trouble.

— Thomas Ligotti [164]

ABSTRACT

In this thesis we present several results relating to treedepth. First, we provide the fastest linear-time fpt algorithm to compute the treedepth of a graph. It decides if a graph has treedepth d in time $2^{O(d^2)} \cdot n$. In the process we answer an open question by Nešetřil and Ossona de Mendez, which asked for a simple linear-time fpt algorithm.

We then proceed to compare treewidth to treedepth. We give lower bounds for the running time and space consumption of any dynamic programming algorithm (for a reasonable definition of dynamic programming on tree/path/treedepth decompositions which we introduce) for the problems Vertex Cover, 3-Coloring and Dominating Set on either a tree, a path or a treedepth decomposition. These bounds match the best known running times for these problems to date. It is not difficult to see that there are linear-time fpt algorithms for Vertex Cover and 3-Coloring parameterized by a given treedepth decomposition of depth d with a space consumption bounded by $\text{poly}(d) \cdot \log n$. We show the same is possible for Dominating Set.

We analyze the random intersection graph model, which attempts to model real-world networks where the connections between actors represent underlying shared attributes. We show that this model, when configured such that it generates degenerate graphs, produces with high probability graphs which belong to a class of bounded expansion, otherwise the graphs are asymptotically almost surely somewhere dense. We then present an algorithm for motif/subgraph counting on bounded expansion graphs which exploits a characterization of bounded expansion graph classes via decompositions into parts of bounded treedepth.

Finally, we present a heuristic to compute tree decompositions which starts by computing a treedepth decomposition and show that it is competitive against other known heuristics.

ZUSAMMENFASSUNG

In dieser Doktorarbeit stellen wir verschiedene Ergebnisse in Bezug auf Baumtiefe vor. Als Erstes liefern wir den schnellsten Linearzeit-*fpt*-Algorithmus, um die Baumtiefe eines Graphen zu berechnen. Er entscheidet, ob ein Graph Baumtiefe d in $2^{O(d^2)} \cdot n$ Schritten hat. Dabei beantworten wir eine offene Frage von Nešetřil und Ossona de Mendez, in der nach einem einfachen Linearzeit-*fpt*-Algorithmus gefragt wurde.

Als Nächstes vergleichen wir Baumweite und Baumtiefe. Wir beweisen untere Schranken für die Laufzeit und den Platzverbrauch von *dynamic programming*-Algorithmen (auf der Basis einer sinnvollen Definition von einem *dynamic programming*-Algorithmus) für Vertex Cover, 3-Coloring und Dominating Set auf einer Baum-, Pfad- oder Baumtiefenzerlegung. Diese Schranken stimmen mit den besten Laufzeiten von bekannten Algorithmen für diese Probleme überein. Es ist nicht schwierig, sich davon zu überzeugen, dass man Vertex Cover und 3-Coloring, parametrisiert mit einer gegebenen Baumtiefenzerlegung mit Tiefe d, mit einem Platzverbrauch von $\text{poly}(d) \cdot \log n$ lösen kann. Wir zeigen, dass das Gleiche für Dominating Set möglich ist.

Wir analysieren das *random intersection graph*-Modell, das versucht, Netzwerke zu modellieren, bei denen Verbindungen gemeinsame Attribute der Knoten darstellen. Wir zeigen, dass dieses Modell, derart konfiguriert, dass es degenerierte Graphen erzeugt, mit hoher Wahrscheinlichkeit Graphen generiert, die zu einer *bounded expansion*-Graphklasse gehören. Weiterhin beweisen wir, dass dieses Modell auf andere Weise konfiguriert, Graphen erzeugt, die asymptotisch fast sicher *somewhere dense* sind. Wir stellen dann einen Algorithmus für *motif/subgraph counting* auf *bounded expansion*-Graphen vor, der eine Charakterisierung von *bounded expansion*-Graphklassen mittels einer Dekomposition des Graphen in Teilen mit beschränkter Baumtiefe ausnutzt.

Zuletzt beschreiben wir eine Heuristik, die zuerst eine Baumtiefenzerlegung und daraus eine Baumzerlegung des Graphen berechnet. Wir zeigen, dass diese mit anderen bekannten Heuristiken für Baumzerlegungen konkurrieren kann.

PREFACE

My work started when I joined Peter Rossmanith's Group in 2012. There, Felix Reidl, also a PhD student working at the chair, introduced me to bounded expansion graph classes, which I immediately found fascinating. I thought that the idea of using the complexity of the model of a minor to define new graph classes was intriguing. Through bounded expansion I came to learn about treedepth, since graph classes of bounded expansion can also be characterized via so-called low treedepth colorings.

During my stay at the RWTH I often had the opportunity to participate in the AMT (Aachen–Metz–Trier) workshop, a regular meeting of Dieter Kratsch's chair at Metz, Henning Fernau's chair at Trier and my own chair. Often we had the pleasure of being joined by Alexander Grigoriev, Mathieu Liedloff and some of his students. At the first one I participated, while looking for questions to work on, an open question by Ossona de Mendez and Nešetřil asking for a simple linear-time fpt algorithm to compute the treedepth of a graph was discussed. Felix and I shortly touched on the idea that, in the context of this problem, one can approximate a tree decomposition basically for free and that it might be possible to perform dynamic programming by computing tables on small trees, e.g. by keeping the number of leaves bounded by the size of the bags. Not much more work was made at this meeting about this problem.

After the workshop, I decided to attempt to tackle this idea and develop an algorithm to compute the treedepth of a graph given a tree decomposition. The idea of using the concept of *restrictions*, basically only keeping partial treedepth decompositions where all the leaves belong to the current bag, seemed immediately like a good idea. Together with the idea of *nice treedepth decompositions*, which allow to exploit the concept of *topological generalizations*, where we relate partial decompositions just by the structure of their ancestor relationship, we were able to answer the open question. I wrote then the complete proof, in a way that only I could understand. Felix then worked heavily with me on streamlining the proof. While writing this thesis I realized, that there is a simpler algorithm based on the same ideas, whose correctness is also easier to prove. Both proofs are presented in this thesis.

After this, I joined Felix in working on his idea of showing that real-world networks have bounded expansion. Together with Erik Demaine, Peter Rossmanith, Somnath Sikdar and Blair Sullivan we attempted to pull through this framework. In my opinion we were successful at it and from this effort the paper "Structural Sparsity of Complex Networks: Bounded Expansion in Random Models and Real-World Graphs" originated. During this work we had the privilege of attending the ICERM seminar "Towards Efficient Algorithms Exploiting Graph Structure," which was a great research environment. The ideas in this rather out-of-left-field paper turned out to be difficult to convey, as such the paper underwent many iterations. To this day, I still consider this to be the most interesting work I have been involved with and I am very grateful for Felix for inviting me to participate.

During this time Blair Sullivan roped Felix and me into a collaboration of hers with Matthew Farrell, Timothy Goodrich and Nathan Lemons, where they were analyzing the hyperbolicity of random intersection graphs, which attempts to model real-world networks where relations represent common attributes between actors. We started analyzing the conditions under which this model generates graphs of bounded expansion with Dr. Sullivan during ICERM. On our return to Aachen, Felix and I proved that it produces graphs of bounded expansion precisely when it produces degenerate graphs.

During this time, I returned to thinking about treedepth, since we were trying to exploit it via low treedepth colorings, a decomposition of a graph of bounded expansion into graphs of bounded treedepth. I realized two things during this time which are relevant to the contents of this thesis: First, I noticed that one could in principle analyze dynamic programming on a treedepth decomposition not by its treedepth, but by the longest distance between a node and an ancestor of the treedepth decomposition connected by an edge of the graph. This lead to the idea of deriving a heuristic for treewidth from computing a treedepth decomposition and attempting to minimize this distance. We gave this idea to Tobias Oelschlaegel as part of his Bachelor Thesis and he did a superb job at developing and implementing a concrete instance of this idea. Second, I noticed that for some problems like 3-Coloring and Vertex Cover it is easy to design linear-time fpt branching algorithms on treedepth decomposition, which use polynomial space in the depth of the decomposition and logarithmic in the size of the instance. When I told this to Peter, he told me he had also been wondering about where this could be pushed. He also told Felix and me about his idea of somehow formalizing the notion of a dynamic programming algorithm and proving lower bounds for the space consumption of any such algorithm for certain problems. Felix and I worked on it and figured out that we would need to have double exponentially sized families of graphs, such that an algorithm which is only allowed to read his input once cannot be prepared for all eventualities without using a considerable amount of memory. We used this notion to develop gadgets for 3-Coloring and Vertex Cover proving that under a reasonable definition of dynamic programming, no such algorithm could use less than $\Omega((3-\varepsilon)^s \cdot \log^{O(1)} n)$ and $\Omega((2-\varepsilon)^s \cdot \log^{O(1)} n)$ space respectively. I then developed the gadget for the more complex case of Dominating Set, proving an $\Omega((3-\varepsilon)^s \cdot \log^{O(1)} n)$ lower bound. Around the same time together with Li-Hsuan Chen, who was visiting our chair at that time, we developed a branching algorithm for Dominating Set on treedepth decompositions, whose space consumption is polynomial in the treedepth and logarithmic in the size of the graph.

ACKNOWLEDGMENT

First and foremost, I would like to thank my supervisor Peter Rossmanith. During my time in Aachen I had the opportunity to hone many different skills which I am sure will prove valuable in my future endeavors. I enjoyed the freedom and the interesting challenges in which I had the opportunity to participate as part of the group. My relation to the chair, which started way before my PhD, has made my academical career infinitely more enriching. I furthermore thank Professor Fernau for being so kind as to report on my thesis.

We must all be thankful to Birgit Willms for not only keeping the chaos we create on a regular basis from exploding in our faces, but also being a friend. Thanks to her, going to work was sure to be more fun and interesting. It is only appropriate to also thank her for her seemingly bottomless well of patience.

I am thankful to all our student assistants Jan Dreier, Fred Grossman, Philipp Kuinke and Kevin Jasik. Their diligence and capacity for independent thought made my life so much easier.

Much of my work would not have been possible without the funding provided by the Deutsche Forschungsgesellschaft under the RO 927/13-1 "Pragmatic Parameterized Algorithms" grant.

It is impossible for me to conceptualize how my PhD would have been had I not shared the journey with Felix Reidl. I am deeply grateful for his friendship. It is a rare privilege that one can share such an experience in the company of a good friend. Despite my generally dour personality, I regularly remind myself how lucky I am to have such a supportive and positive person by my side.

I am grateful to Blair Sullivan for our close collaboration on several projects under the umbrella of using deep theory in practical settings. I truly cherished the opportunity to participate in ICERM and my visits to Raleigh.

During my research career I had the pleasure to collaborate with many other researchers. Beyond the people already mentioned this list includes Aaron Adcock, Katrin Casel, Li-Hsuan Chen, Erik Demaine, Martin Demaine, Markus Dregi, Jan Dreier, Henning Fernau, Fedor Fomin, Jakub Gajarský, Moritz Gobbert, Petr Hlinený, Ling-Ju Hung, Ton Kloks, Kyle Kloster, Philipp Kuinke, Philipp Kranen, Stephan Kreutzer, Erik Jan van Leeuwen, Daniel Lokshtanov, Jan Obdrzálek, Michael O'Brien, Sebastian Ordyniak, Marcin Pilipczuk, Michał Pilipczuk, Andrew van der Poel, Saket Saurabh, Markus Schmid and Thomas Seidl. I am thankful to all of them for the opportunity to work together.

Pål Grønås Drange and Pim van ’t Hof deserve special mention for being wonderful hosts during our stays in Bergen. I am also thankful to Konstantinos Stavropoulos, not only for the research collaborations, but for the camaraderie and the deep philosophical conversations about science and everything else.

I am also very thankful to my family and friends for their continued unwavering support. If I showed fortitude in the face of adversity, it was in great part thanks to their trust in my capabilities.

Last but not least, I am thankful to Helena for being the best thing to have happened to me.

CONTENTS

Part I

INTRODUCTION

1 TREEDEPTH BY ANY OTHER NAME

Almost every computational question is easy on trees. Since trees form a very restrictive graph class, a natural arising question is if problems remain easy when a graph is close to being a tree. A useful way to formally capture this notion of similarity is through the widely studied graph measure *treewidth* [160]. When we look at graph classes with bounded treewidth, a host of NP-hard problems become solvable in polynomial time. Most famously, by Courcelle's Theorem, all problems expressible in MSO (and later MSO-OPT), a rather general logic capable of expressing a wide array of problems, are solvable in linear time on graphs of bounded treewidth [13, 58]. Nevertheless, expecting a graph to have low treewidh is still rather restrictive. Is it worth studying measures even more restrictive than treewidth? This is arguably the case if the structural properties measured by such a notion are algorithmically exploitable beyond what treewidth allows and/or appear naturally when studying some other concept. Extensive analysis of measures that bound treewidth, such as pathwidth/vertex separation number [40, 62, 84, 147, 223, 234], bandwidth [15, 20, 85, 106, 129, 233], the size of a vertex cover [5, 85, 86, 129, 143, 151], etc., can be found in the literature. In this thesis we present results related to another such measure, called *treedepth*,[1] which can be said to capture the similarity of a graph to a star (i.e. trees of depth one). We start with an historical overview of treedepth and equivalent notions to show how it is both algorithmically exploitable and arises organically in several contexts.

BASIC DEFINITION

Intuitively, in the same way that treewidth measures how tree-like a graph is, treedepth measures how star-like a graph is. Formally, a *treedepth decomposition* of a graph G is a pair (F, ψ), where F is a rooted forest and $\psi\colon V(G) \to V(F)$ is an injective mapping such that if $uv \in E(G)$ then either $\psi(u)$ is an ancestor of $\psi(v)$ in F or vice versa. Whenever we deal with treedepth decompositions in this thesis, the mapping ψ will usually be implicit as we will have $V(G) \subseteq V(F)$. The *depth of a treedepth decomposition* is the depth of its rooted forest, i.e. the maximum number of nodes in a path from a root to a leaf.

Definition 1 (Treedepth). The *treedepth* $\mathbf{td}(G)$ of a graph G is the minimum depth of any treedepth decomposition of G.

1 The spelling "tree-depth" is more common in the literature. Treewidth is written without a hyphen in most of the rather extensive literature. Since in this work we often talk simultaneously about treewidth and treedepth we decided to write both in a consistent manner.

A different way to look at treedepth is as a quantification of the "depth" of a graph by measuring the number of steps which are necessary to make the graph disappear by iteratively removing a node from every component. Given a treedepth decomposition of depth d, we know we can do this in d steps by iteratively removing the root nodes of the remaining tree decomposition. This is formally expressed in the following equivalent definition.

Definition 2 (Treedepth)**.** The *treedepth* of a graph G with connected components $G_1, \ldots, G_\ell$ is defined as follows:

$$\mathbf{td}(G) = \begin{cases} 1 & G = K_1 \\ \max_{1 \leqslant i \leqslant \ell} \mathbf{td}(G_i) & \ell > 1 \\ 1 + \min_{v \in V(G)} \mathbf{td}(G - v) & \text{otherwise} \end{cases}$$

If we are given a treedepth decomposition of a graph it is clear that, by always selecting the root of a component in the third case, the measure as defined above is at most the depth of the decomposition. In the other direction, we can recursively construct a treedepth decomposition from the above definition by starting a new subtree for every component and choosing the root of the subtree to be the node selected by the min operator in third case. The treedepth is then in both cases the recursion depth and thus the two definitions are equivalent.

HISTORY OF TREEDEPTH AND RELATION TO TREEWIDTH

It is not difficult to show that the treedepth of a graph is bounded by its treewidth. Even stronger, a treedepth decomposition of a graph immediately provides a path decomposition, by taking as bags the nodes in paths from root to leave and arranging them in by the order of the leaves in any planar embedding of the decomposition [193]. The history of these two width-measures is both interesting and complicated. Treewidth appeared in the literature first as *dimension* in 1972 [26] and was rediscovered by Rudolf Halin in the context of *S-functions* in 1976 [116], which he had already introduced previously as *sZ-treues Feinheitsmaß* [115]. Since S-functions generalize several graph measures, treedepth could be related to treewidth by also being an instance of an S-function. This is not the case, since a fundamental characteristic of S-functions is that the definition is recursive in such a way that the measure does not necessarily decrease in the smaller parts. This is precisely the contrary of what we want in treedepth, were the measure for a connected graph should decrease when removing the root node of the decomposition.

Definition 3 (S-function [116])**.** A function f into the integers defined for all finite graphs is an *S-function* if it fulfills the following properties:

1. if H is a minor of $G \implies f(H) \leqslant f(G)$
2. $f(\varnothing) = 0$
3. $f(G') = f(G) + 1$, where G' is G with an added universal vertex
4. $G = G' \cup G''$, $G' \cap G'' = K_s$ for $s \geqslant 0 \implies f(G) = \max\{f(G'), f(G'')\}$

Treedepth fulfills the first three conditions but does not fulfill the more relevant fourth condition. This is exemplified by the graph G in Figure 1.1. The subgraphs $G' = G[\{1,2,3,4\}]$ and $G'' = G[\{1,2,3,5\}]$ fulfill the requirements of condition 4. Nevertheless, the treedepth of G is four, but the treedepth of G' and G'' is three and thus $\mathbf{td}(G) \neq \max\{\mathbf{td}(G'), \mathbf{td}(G'')\}$. This gives us an insight into a fundamental difference between treewidth and treedepth. Both can be defined recursively, but in one case the measure will stay the same in the subgraphs and in the other it *must* decrease.

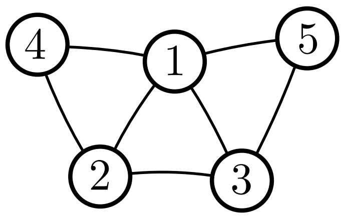

Figure 1.1: A counter-example for treedepth being an S-function

The origin of the notion of treedepth is not easy to pinpoint. How easy it is to take a graph apart by removing nodes is the kind of question that arises naturally. In fact, this is precisely how it developed in research about Cholesky factorizations, a common way to solve systems of linear equations. In this context, one can use a tree structure which recursively decomposes the graph underlying the system of equations to parallelize the problem and break a sparse instance into small dense parts. In this context the notions of an *elimination tree* and *elimination height*, which are related to treedepth decompositions and treedepth, respectively, are frequently used.

The question of the elimination height of a graph being equivalent to treedepth is somewhat muddled since elimination trees are defined for *chordal graphs*. A graph is chordal if every induced cycle has length three, i.e. if every cycle of length greater than three has a *chord*, an edge connecting two non-consecutive nodes of the cycle. Every chordal graph has at least one ordering of the nodes called a *perfect elimination*, such that for every node all neighbors that come later in the ordering form a clique. With these definitions at hand, elimination trees are defined as follows:

Definition 4 (Elimination tree)**.** The *elimination tree T* of a chordal graph G is a tree on the nodes of G such that all neighbors of a node u in G which are ancestors of u in T form a clique in G.

This definition implies that recursively eliminating the leaves of the elimination tree results in a perfect elimination of the chordal graph.

To define the elimination height of a (possibly non-chordal) graph we need to define a *triangulation*: A triangulation of a graph G is a supergraph on the same node set which is chordal. The elimination height of a graph G is the minimum height over all elimination trees of any triangulation of G. A treedepth decomposition T implicitly describes a triangulation for a graph such that T is a valid elimination tree of the resulting chordalization: Take every path from a root to a leaf and make it a clique in the graph. Thus, the problem of deciding the elimination height of a graph is the same as deciding its treedepth. This generalization of elimination trees via elimination height to non-chordal graphs was not explicitly stated in the earliest works on this notion, however, later texts on treedepth credit these early works as having established the connection between elimination trees and treedepth implicitly [204].

The seemingly oldest reference that defines elimination trees (sometimes shortened to *e-trees*) is in a technical report by Pieck from 1980 [200]. The presentation of elimination trees takes a whole Chapter in Kees' doctoral thesis [145], who references the aforementioned technical report, the technical report, in turn, states[2] that its content is based on preliminary work by Jess and Kees [132, 133], in which they also introduce the notion of elimination trees.[3] In its survey "The role of elimination trees in sparse factorization" Liu claims that Schreiber [218] is "perhaps the first one to formally define the elimination tree structure," (Schreiber does not give it a name though). The previously provided references that define elimination trees (actually calling them elimination trees) are from around the same time or older and as such this claim by Liu seems inaccurate. Liu also comments that the term elimination tree had been previously used by Duff to refer to a slightly different structure in his paper "Full matrix techniques in sparse Gaussian elimination" [75]. The term elimination tree does not appear in this paper, nevertheless Duff does use the term in two related papers from 1983 [74, 76], which is around the same time Jess and Kees were working with this concept. Several authors claim this notion was used implicitly in previous work on factorization [133, 170]. Kees himself points out [145] that this notion relates to the much older notions of *tearing* and *decomposition into bordered block diagonal form* [47, 56, 146, 153, 215]. Since elimination trees are closely tied to chordal graphs, computing an optimal elimination tree for such a graph can be done in polynomial time [169].

The NP-hardness of deciding the treedepth of a graph was proven in this context by Pothen, who showed that finding a chordalization such that the correspondent optimal elimination tree's height is minimized is NP-hard [204]. Interestingly, deciding the treedepth of a chordal graph is NP-hard [68] (here the slight difference in the definitions of elimination tree and treedepth makes a noticeable difference, since computing

2 The technical report is in Dutch and as such I have derived what it says with the help of my knowledge of German and some guesswork.

3 It is worth mentioning that the definition by Pieck forces the nodes in the neighborhood of a node u which are *not* ancestors of u to be a clique, which contradicts the later definition by Jess and Kees.

the elimination height and an optimal elimination tree for a chordal graph are two distinct problems).

The definition of elimination height being based on finding certain chordalizations of graphs points to an interesting relation between treedepth and treewidth: Computing the treewidth of a graph is equivalent to finding a triangulation with smallest clique size [12]. Finding a triangulation of a graph such that its corresponding elimination tree is of minimum depth is equivalent to finding a triangulation G_T with minimal clique size, such that G_T is not only chordal but *trivially perfect*. A graph is trivially perfect when for all of its induced subgraphs the size of the maximum independent set equals the number of maximal cliques. This characterization of elimination height/treedepth follows from trivially perfect graphs being precisely those graphs which are *closures* of forests [107], since the closure of a forest is the graph which results from adding all edges between any node and all its ancestors.

EQUIVALENT NOTIONS

Elimination trees are arguably the oldest studied concept that formalizes the notion of treedepth, but the notion of the "depth" of a graph has been independently studied in the past under other names and definitions, which are equivalent or strongly related to treedepth.

We have already discussed how we can characterize treedepth via a process of iteratively removing nodes. This way of thinking about it can be found in literature under the notion of a 1-*partition tree* [125]. A natural alternative way to think about a treedepth decomposition is as a process of iteratively removing separators instead of nodes. The separator removed becomes then a path of nodes in the tree, such that all nodes except the deepest one have only one child. Obviously, removing such a path starting at the root of a treedepth decomposition will result in the graph falling apart into more components, thus making the nodes in the path a separator. This way of looking at decomposing a graph has been independently studied under the name *separation game*, which was introduced in the context of studying the complexity of finding local optima and proven to be NP-hard [171] around the same time as Pothen showed elimination height to be NP-hard.

We can find the claim in the literature that the notion of separator/partition trees, which is used in VLSI design, is related to some notion equivalent to treedepth [126, 127, 138, 142, 151, 155, 208]. One should be careful with this statement since, as several papers explicitly state, separator trees are only equivalent to treedepth if we attempt to minimize the height [126, 127, 142, 155]. It seems important to mention that the depth of separator trees does not appear to be a crucial factor in VLSI design [162] and that partition trees are explicitly defined to be binary [228]. As such, it is not clear what applicability concepts equivalent to treedepth have for VLSI design. The separator tree structure has been applied in the context of Cholesky factorization, but not in a direct attempt to minimize the height of the elimination tree [168].

The notion of treedepth has also been studied under the names *vertex ranking* and *ordered coloring*, which both share the same definition.

Definition 5 (Ordered coloring/vertex ranking)**.** A *d-ranking* or *ordered d-coloring* of a graph $G = (V, E)$ is a vertex coloring $c\colon V \to \{1, \ldots, d\}$ such that for any two vertices of the same color, any path connecting them contains a vertex with a higher color. The minimum value of d for which such a coloring exists is the the *vertex ranking number* or the *ordered chromatic number* of the graph respectively.

It is easy to see that if we color every level of a treedepth decomposition of depth d with a different color, we get a d-ranking/ordered d-coloring of the graph. The proof of the other direction goes as follows: In a connected component of a graph with a d-ranking/ordered d-coloring there can only be one node with the label d', where d' is the greatest label appearing in the component. From every component, remove the node with the greatest label and add them as roots to the treedepth decomposition. Recurse into the remaining components and keep building the treedepth decomposition in the same fashion. This results in a treedepth decomposition of depth at most d.

Several reasons are given in the literature to study this problem: As a more restrictive version of graph coloring [142]; an in-between step in the approximation of an *edge ranking* [63]; making sure that communications in a network always have to go through a node with a higher rank, so that they can be monitored [125, 126] and the scheduling of assembly steps in manufacturing systems [67, 126].

More recently, Ossona de Mendez and Nešetřil reintroduced the concept in the guise of *treedepth* in their monograph "Sparsity" [193]. They show that treedepth has important connections to the structure of sparse graphs by proving that a very general class of sparse graphs, the so-called graphs of *bounded expansion*, which generalize even topological minor free graph classes, can be decomposed into pieces of bounded treedepth.

Proposition 1 (Low treedepth colorings [190])**.** *Let $\mathcal{G}$ be a graph class of bounded expansion. There exists a function f such that for every $G \in \mathcal{G}$, $r \in \mathbb{N}$, the graph G can be colored with $f(r)$ colors so that any $i < r$ color classes induce a graph of treedepth $\leqslant i$ in G.*

When using treedepth to give this alternative characterization of bounded expansion graph classes they originally used the equivalent notion of a *centered coloring*.

Definition 6 (Centered coloring)**.** A coloring of the nodes of a graph G is called centered if for every connected induced subgraph of G there is a color which appears exactly once.

The *size* of a centered coloring is simply defined as the number of colors used. The argumentation as to why this is equivalent to treedepth is very similar to the one for a vertex ranking/ordered coloring. If we give every level of a given treedepth decomposition a different color, this is clearly a centered coloring, since every connected component corresponds to a subtree of the treedepth decomposition and thus the root has a unique color. To construct a treedepth decomposition, we take every unique color of every component of the graph and set them as the roots of the treedepth decomposition. We remove these nodes and continue building the treedepth decomposition in this fashion. This results in a treedepth decomposition of depth at most the number of colors of the centered coloring.

The idea of centered colorings and treedepth can be traced to work by Nešetřil and Shelah while studying notions related to homomorphisms of graphs, where they introduce the treedepth-equivalent notion of a *ranking* [188]. Later, Nešetřil and Ossona de Mendez, while also studying homomorphisms, re-introduced this notion and called it treedepth [186]. The notion thus arose again naturally.

Even more recently, Gruber and Holzer noticed a connection between the notion of the *cycle rank* and treedepth [110]. Cycle rank is a measure of digraphs which is connected to results on the *star height* of regular languages [79, 111], itself a useful parameterization to analyze the relative sizes of an automaton and a regular expression which express the same language. If one defines *undirected* cycle rank by forcing the directed graph to be symmetrical this is immediately equivalent to treedepth.

alternative characterizations of treedepth

Since treedepth is a minor-closed property, i.e. the treedepth of a graph cannot increase after contracting edges and deleting nodes, it follows from the graph minor theorem that graphs of treedepth d can be characterized by a finite set of forbidden minors. This set grows at least like a double exponential and at most like a triple exponential of d [77]. A polynomial approximation of treedepth is possible by excluding a few simple minors [144], in a similar vein as what the polynomial grid minor theorem [53] implies about treewidth.

Treedepth can be characterized in several ways by *cops-and-robbers games*, just like treewidth and pathwidth can be characterized by variations of such games. In one version [99, 109] of the cops-and-robbers game there are d cops and one robber. The turns of the game alternate between the cops and the robber as follows. The cops start and have full knowledge about the robber's position. At the beginning of their move they must announce on what vertex they want to place a cop. Cops cannot be removed after being positioned. The robber can then move along a path of any length as long as none of its nodes is already occupied by a cop. Then a new cop is set at the announced position and the game repeats. The cops win if they set a cop where the robber is. A graph has treedepth d if only if d cops have a winning strategy for the graph.

There are several more versions of a cops-and-robbers game that are equivalent to treedepth. These are called *LIFO-search games* [101, 121]. Here the cops have two possible moves. They can either position a cop on the graph up to a maximal number of d cops or remove the last cop that was positioned on the graph. Interestingly, the cops have a winning strategy with d cops if and only if the graph has treedepth d irrespective of the cops being able to see the robber or of their tactic's monotonicity (i.e. whether the cops' strategy provides the robber access to already searched areas). These LIFO-games lead to a characterization of treedepth via *shelters* [101], in a similar way that treewidth can be characterized via *brambles* [219].

Definition 7 (Shelter)**.** Let $\mathcal{S}$ be a non-empty set of connected subgraphs of G partially ordered by the subgraph relation. $\mathcal{S}$ is called a *shelter* if for every $H \in \mathcal{S}$ one of the following holds:

- H is minimal in $\mathcal{S}$ w.r.t. the subgraph relation.
- For every $x \in V(H)$ there exists an $H' \in \mathcal{S}$ such that $H' \subset H$ and $x \notin V(H')$.

The *thickness* of a shelter is minimal length of a maximal chain of $\mathcal{S}$.

Proposition 2 ([101])**.** *The treedepth of a graph is the maximum thickness of a shelter of G.*

APPLICATIONS

Treedepth has turned out to be the right tool to characterize certain dichotomies. It was shown that MSO and FO have the same expressive power on a graph class $\mathcal{C}$ if and only if $\mathcal{C}$ has bounded treedepth [82]. This is related to a previous result showing that MSO-definable problems are solvable by uniform constant-depth circuit families when restricted to input structures of bounded treedepth [80, 83]. In a similar characterization, a monotone class of graphs has bounded treedepth if and only if it is well quasi-ordered for the induced-subgraph relation [71, 187].

The parameter *modulator to bounded treedepth*, i.e. a set of nodes whose removal from the graph results in a graph of constant treedepth, was fundamental in the development of meta-kernelization results for classes of bounded expansion and beyond [97]. Bougeret and Sau showed that this could also be used as a parameter for a polynomial kernel of Vertex Cover on general graphs [45]. This is especially interesting since there is no polynomial kernel when the parameter is a modulator to bounded treewidth. They furthermore showed, that this does not work for Dominating Set. The existence of a polynomial kernel for the problem of computing a modulator to bounded treedepth, the size of the modulator itself being the parameter, has been developed in the context of the $\mathcal{F}$-Minor-Free Deletion problem [102]. Other research has explored the question of when q-Coloring can be solved in time $O((q-\varepsilon)^k \cdot \text{poly}(n))$ for some $\varepsilon > 0$ when parameterized by modulator to some graph class $\mathcal{C}$ [128]. The research indicates that $\mathcal{C}$ having or not having bounded treedepth plays a fundamental role.

A further property of treedepth is that it works as a parameter when other related parameters fail to make the problem fpt. Gutin, Jones and Wahlstöm proved that the Mixed Chinese Postman Problem, which is W[1]-hard parameterized by treewidth, is fpt parameterized by treedepth [112]. It is not difficult to see that the Firefighter Problem, which is NP-hard on graphs of bounded treewidth [87], is MSO-expressible for graphs of bounded path length. Since the path length is bounded in graphs of bounded treedepth [193] it follows by Courcelle's Theorem that the Firefighter Problem is fpt parameterized by the treedepth of the graph [130]. It is still an open question whether Metric Dimension is fpt parameterized by treewidth [70], but it is clear that it is fpt when parameterized by treedepth for the same reason [130]. Furthermore, H-Coloring Reachability, which is not even fpt when parameterized by the bandwidth of the graph (which is an upper bound of the treewidth), is nevertheless fpt when parameterized by treedepth [233]. Another problem which cannot be parameterized by bandwidth but allows parameterization by treedepth is 1-Planar Drawing [20]. A recent result presents tight bounds on the running time with which (k,r)-Center can be solved when parameterized by treedepth [143] and provides a faster algorithm than what is possible when parameterizing by treewidth (assuming SETH) [44].

At least twice in the recent past researchers have realized that the correct parameter for their analysis was treedepth after parameterizing by both treewidth and the height of the tree decomposition. This happened in the context of counting perfect matchings using little space [95] and when investigating the space complexity of deciding MSO formulas on graphs of bounded treewidth [81].

It is interesting to notice that the last results mentioned are about the space usage of algorithms. Treedepth allows for branching and thus some problems such as Vertex Cover, Independent Set and 3-Coloring admit fpt algorithms that use very little space. In the case of 3-Coloring, the space consumption is bounded by $O(d + \log n)$. Based on this algorithm, Pilipczuk and Wrochna showed that computations on treedepth decompositions correspond to a model of non-deterministic machines that work in polynomial time and logarithmic space, with access to an auxiliary stack of maximum height equal to the decomposition's depth [201]. Treedepth is also key in characterizing which homomorphism problems can be solved in logarithmic space [54].

This condensed review of literature relating to the concept of treedepth makes it clear that treedepth is a concept that is re-discovered over and over again. Furthermore, recent research indicates that real-world networks belong to classes of bounded expansion [65]. Since there is a strong connection between bounded expansion graph classes and treedepth, there might be practical applications for this measure, despite it being likely big for most real-world graphs.

In this thesis we present further results on computing treedepth, exploiting a treedepth decomposition to solve problems using little space, analyze the structure of certain real-world complex networks, design an algorithm exploiting treedepth for such networks and strengthen the relation to treewidth by developing a heuristic for treewidth which starts by computing a treedepth decomposition. A more detailed overview of the results in this thesis can be found in Section 2.

2 ORGANIZATION AND SUMMARY OF RESULTS

This thesis is divided in five parts. Part i introduces the concept of treedepth, gives some historical context. Furthermore, certain basic preliminary concepts and results are introduced.

In Part ii an fpt algorithm to compute the treedepth of a graph is presented. We give an algorithm to compute a treedepth decomposition of depth d in time $2^{O(wd)} \cdot n$ given a tree decomposition of width w. We achieve this results by proving we only need to be able to find *nice treedepth decompositions*, a concept that we introduce. We then show how this algorithm can be extended to a simple algorithm that does not require to be given a tree decomposition as part of the input and runs in time $2^{2^{O(d^2)}} \cdot n$. This solves an open question posed by Ossona de Mendez and Nešetřil [193]. We can also use this result to give the fastest known exact parameterized algorithm to date, with a running time of $2^{O(d^2)} \cdot n$, using a previous result that provides a constant factor approximation for treewidth in single-exponential time [38].

We then compare in Part iii what can be done via dynamic programming to what can be done via branching given a tree/path/treedepth decomposition. We introduce machinery to capture the workings of common dynamic programming algorithms on treewidth. We show that it is neither possible to solve via dynamic programming 3-coloring or Dominating Set using less than $O\big((3-\varepsilon)^s \cdot \log n\big)$ space nor Vertex Cover using less than $O\big((2-\varepsilon)^s \cdot \log n\big)$ for any $\varepsilon > 0$. As a lower bound for time complexity these match the lower bounds previously proven based on SETH for any algorithm working on a tree decomposition [172]. We thus conclude that (presuming our machinery captures the basic procedure of dynamic programming) the assumption that one cannot do better than dynamic programming to exploit a tree decomposition leads to even tighter bounds than assuming SETH. Since as good as all linear time algorithms on tree decompositions for NP-hard problems are dynamic programming algorithms, this is not a far-fetched assumption. When given a treedepth decomposition it is easy to see that Vertex Cover and 3-Coloring can be solved using space bounded polynomially in the treedepth d and logarithmically in the number of nodes. We show that the same is possible for Dominating Set. This, together with other results from the literature, indicates that an advantage of treedepth is low space consumption.

Previous results show that many real-world networks are likely to be structurally sparse [65, 207]. This argument is defended by showing that certain random graph models used to model real-world networks produce graphs which belong to a class of bounded expansion with high probability. In Part iv we show that this is also true for *random intersection graphs* whenever the model produces sparse graphs. These random graphs are used to model networks where relationships express common attributes, such as film actors having appeared on the same movie. We already mentioned how bounded expansion graphs can be characterized by being decomposable into parts of bounded treedepth. Using this decomposition we present a fast algorithm to count subgraphs given a treedepth decomposition in linear time, assuming the network has bounded expansion. Subgraph counting appears in the network science literature in the form of *motif counting* and the *graphlet degree distribution*, a way of analyzing/fingerprinting real-world networks [179, 206, 209].

Finally, in Part v we strengthen the relation between treewidth and treedepth by showing how a heuristic for treedepth can be used as a fast heuristic for treewidth. For this we introduce the notion of the *stretch* of a treedepth decomposition T, which we define to be the maximum distance of any two nodes x and y in T which are connected by an edge of G. We show that the stretch of a treedepth decomposition is an upper bound on the graph's treewidth. By manipulating the treedepth decomposition we can attempt to minimize its stretch. Then we can derive an elimination scheme from the manipulated treedepth decomposition. A comparison with thirteen well-established heuristics shows that the resulting heuristic is indeed competitive in quality and speed.

3 PRELIMINARIES

All our graphs are finite and simple. Given a graph G, we use $V(G)$ to denote its vertex set and $E(G)$ to denote its edge set. In the context of this work, n will always be the number of vertices of the graph, unless otherwise stated. We assume that $V(G)$ is a totally ordered set and use uv instead of $\{u,v\}$ to denote the edges of G. For a graph G and a vertex $x \in V(G)$, the set $N_G(x)$ denotes the neighbors of x in G. We extend this notation to vertex sets via $N_G(S) = \bigcup_{x\in S} N_G(x) \setminus S$. We write $N_G[x]$ to denote the closed neighborhood of x in G and extend this notation to vertex sets via $N_G[S] = \bigcup_{x\in S} N_G[x]$. We will drop G in the subscript if the graph is clear from the context. We let $G[X]$ denote the subgraph of G *induced* by some set $X \subseteq V(G)$, where a subgraph H of G is induced if for every pair of vertices $u,v \in V(H)$ the edge uv exists in H if and only if it exists in G. We denote the complete graph on s nodes by K_s. By $d_G(u,v)$ we will denote the distance between the nodes u and v in the graph G, i.e. the number of edges in a shortest path of G between u and v. Here we might also drop the subscript if the graph is clear from the context. An $(\leqslant r)$-*subdivision* of a graph H is the graph that results from replacing every edge of the graph by a path with at most r nodes. Given a node x of a graph G we express the graph that results after deleting x by $G - x$. Given an edge $e = uv$ of a graph G, we let G/e denote the graph obtained from G by *contracting* the edge e, which amounts to deleting the endpoints of e, introducing a new vertex w_{uv} and making it adjacent to all vertices in $(N(u) \cup N(v)) \setminus \{u,v\}$. For an edge $e = uv$, by *contracting v into u*, we mean contracting e and renaming the vertex w_{uv} to u. A graph H is minor of a graph G if H can be constructed from G by contracting edges and deleting edges and nodes. For a function $f\colon X \to Y$ and a set $X' \subseteq X$ we will define applying the function on such a set to be $f(X') = \{f(x) \mid x \in X'\}$. By $f|_{X'}$ we will refer to the function we get by restricting the input set of f to X'. For sets A, B, C we write $A \uplus B = C$ to express that A, B partition C. We write the the symmetric difference between two sets A and B as $A \triangle B = \{a \mid a \in A \setminus B \text{ or } a \in B \setminus A\}$. All logarithms are base two.

We will work extensively on trees and forests. In this context, a *rooted tree* is a tree with a specially designated node known as the *root*. Let T be a rooted tree with root r and let $x \in V(T)$. Then an *ancestor* of x is any node (other than itself) on the path from r to x. Similarly a *descendant* of x is any node (other than itself) on a path from x to a leaf of T. In particular, x is neither an ancestor nor a descendant of itself. We denote by P_x the set of ancestors of x in T.

A *rooted forest* is a disjoint union of rooted trees. Whenever we refer to a forest we will mean a rooted forest. For a node x in a tree T of a forest, the *depth* of x in the forest is the number of vertices in the path from the root of T to x (thus the depth of the root is one). The *height of a forest* is the maximum depth of a node of the forest. The *closure* $\operatorname{clos}(F)$ of a forest F is the graph with node set $V(F)$ and edge set $\{xy \mid x \text{ is an ancestor of } y \text{ in } F\}$. Furthermore we will need the notions of a *subtree* and the *height* of a node.

Definition 8 (Subtree rooted at a node)**.** Let x be a node of a tree T and let S be all the descendants of x in T. The *subtree of T rooted at x*, denoted by T_x, is the subtree of T induced by the node set $S \cup \{x\}$ with root x.

Definition 9 (Subtree rooted at a node with child selection)**.** Let x be a node of a tree T, let C be a set of children of x in T and let S be all descendants of nodes of C in T. The tree denoted by T_x^C, is the subtree of T induced by the node set $S \cup C \cup \{x\}$ with root x.

Definition 10 (Height of a node)**.** Let x be a node of a tree T and let T_x be the subtree of T rooted at x. Then we define the *height* of x in T ($\text{height}_T(v)$) to be the height of T_x.

TREEDEPTH

If we consider a spanning tree of a graph given by a depth first search, we know all edges of the graph will either be part of the tree, or forward/back edges. This means that such a tree is a treedepth decomposition of the graph. Furthermore, since a path of length greater than $2^d - 1$ has treedepth greater than d and treedepth is a minor-closed property, it follows that no graph of treedepth d contains a path of length greater than $2^d - 1$. Thus, if a graph has bounded treedepth, it is easy to find to find a treedepth decomposition of bounded treedepth of the graph in linear time.

Proposition 3 ([193])**.** *Let G be a graph of treedepth d. Then a treedepth decomposition which is the tree given by a depth first search of G has treedepth at most 2^d.*

TREEWIDTH

One of the most famous width measures is *treewidth*, which measures the similarity of a graph to a tree.

Definition 11 (Treewidth)**.** Given a graph $G = (V, E)$, a *tree decomposition of G* is an ordered pair $(T, \{W_x \mid x \in V(T)\})$, where T is a tree and $\{W_x \mid x \in V(T)\}$ is a collection of subsets of $V(G)$ such that the following hold:

1. $\bigcup_{x \in V(T)} W_x = V(G)$;
2. for every edge $e = uv$ in G, there exists $x \in V(T)$ such that $u, v \in W_x$;

3. for each vertex $u \in V(G)$, the set of nodes $x \in V(T)$ such that $u \in W_x$ induces a subtree of T.

We call the vertices of T *nodes*. The vertex sets W_x are usually called *bags*. The *width* of a tree decomposition is the size of the largest bag minus one. The *treewidth* of G, denoted by $\mathbf{tw}(G)$, is the smallest width of a tree decomposition of G.

In the definition above, if we restrict T to being a path, we obtain the well-known notions of a *path decomposition* and *pathwidth*. We let $\mathbf{pw}(G)$ denote the pathwidth of G. Let $(T, \{W_x \mid x \in V(T)\})$ be a tree-decomposition; let $x \in V(T)$ and, let S be the set of descendants of x. Then we define $V(\mathcal{T}_{W_x}) = \bigcup_{y \in S \cup \{x\}} W_y$.

We will only work on *nice tree decompositions*, which are tree decompositions with the following characteristics:

- Every node has either zero, one, or two children.
- Bags associated with leaf nodes contain a single vertex.
- If x is a node of T with a single child x' and if X and X' are the bags assigned to these nodes, then either $|X \setminus X'| = 1$ or $|X' \setminus X| = 1$. In the first case, X is called an *introduce bag* and, in the second, a *forget bag*.
- If x is a node with two children x_1 and x_2 and if X, X_1, X_2 are the bags assigned to them, then $X = X_1 = X_2$. We call such a bag X a *join bag*.

Proposition 4 ([149]). *Given a graph G with n vertices and a tree decomposition of G of width w it is possible to compute a nice tree decomposition of G of width w with at most $4n$ bags in linear time.*

The main property of tree decompositions that we will exploit is the fact that each bag X associated with an internal node is a vertex separator of G. Hence with each bag X of a nice tree decomposition we can associate two (forget, introduce) or three (join) well-defined terminal subgraphs with terminal set X. For further information on treewidth and tree decompositions, we refer the reader to Bodlaender's survey [34].

There is a clear relation between the treedepth and the treewidth of a graph. It is not difficult to see that we can create a path decomposition of a graph G out of bags which contains the paths from root to leaf of a treedepth decomposition of G. Furthermore, it is easy to balance the separator tree given by a tree decomposition $\mathcal{T}$ of a graph G such that it's depth is logarithmic in the number of vertices of G. We can thus create a treedepth decomposition T from $\mathcal{T}$ by converting paths of bags into paths of nodes. The depth of T will thus only grow logarithmically in the number of nodes of G. These relations are succinctly captured in the following proposition.

Proposition 5 ([193]). *For a graph G, it holds that $\mathbf{tw}(G) \leqslant \mathbf{pw}(G) \leqslant \mathbf{td}(G) - 1$ and $\mathbf{td}(G) \leqslant \mathbf{tw}(G) \cdot \log n$.*

FIXED PARAMETER TRACTABILITY

Parameterized complexity deals with algorithms for decision problems with instances consisting of a pair (x, k), where k is a secondary measurement known as the *parameter*. A major goal in parameterized complexity is to investigate whether a problem with parameter k admits an algorithm with running time $f(k) \cdot |x|^{O(1)}$, where f is a function depending only on the parameter and $|x|$ represents the input size. Parameterized problems that admit such algorithms are called *fixed-parameter tractable* and the class of all such problems is denoted FPT. For an introduction to the area please refer to existing literature [72, 88, 196]. It is sometimes useful in the context of fixed parameterized tractability to use the O^* notation, which is the big O notation with suppressed polynomial factors. We say an algorithm runs in *linear fpt time* if its running can be expressed as $f(k) \cdot |x|$.

PROBABILITY

When we use the terms *asymptotically almost surely* (a.a.s.) and *with high probability* (w.h.p.), we do so using the following conventions: For each integer n, let $\mathcal{G}_n$ define a distribution on graphs with n vertices (for example, coming from a random graph model). We say the events E_n defined on $\mathcal{G}_n$ hold *asymptotically almost surely* (a.a.s.) if $\lim_{n\to\infty} \mathbb{P}[E_n] = 1$. We say an event occurs *with high probability* (w.h.p.) if for any $c \geqslant 1$ the event occurs with probability at least $1 - f(c)/n^c$ for n greater than some constant, where f is some function only depending on c. As a shorthand, we will simply say that $\mathcal{G}_n$ *has some property* a.a.s. (or w.h.p.).

STRONG EXPONENTIAL TIME HYPOTHESIS

We will mention certain results from the literature which assume the *strong exponential time hypothesis* (SETH) to be true. This hypothesis was first proposed as part of an open question by Impagliazzo and Paturi [122] while presenting results relating to the *exponential time hypothesis* (ETH) [123]. It was then given this name and formalized by Calabro, Impagliazzo and Paturi [51]. These are both conjectures about the time complexity of solving k-SAT. Let $s_k = \inf\{\delta\colon k\text{-SAT can be solved in time } 2^{\delta n}\}$, where n is the size of the input instance. The exponential time hypothesis is that $s_3 > 0$ and the strongly exponential time hypothesis is that $\lim_{k\to\infty} s_k = 1$. For justifications and support of these hypotheses see the referenced literature.

BOUNDED EXPANSION

Some of our results with pertain the notion of *bounded expansion graph classes*. We provide now the basic characterization of these classes and some equivalent ones we will make heavy use of later. First some preliminary definitions.

Definition 12 (Shallow topological minor, nails, subdivision vertices)**.** A graph M is an *r-shallow topological minor of G* if a $(\leqslant 2r)$-subdivision of M is isomorphic to a subgraph G' of G. We call G' a *model of M in G*. For simplicity, we assume by default that $V(M) \subseteq V(G')$ such that the isomorphism between M and G' is the identity when restricted to $V(M)$. The vertices $V(M)$ are called *nails* and the vertices $V(G') \setminus V(M)$ *subdivision vertices*. The set of all r-shallow topological minors of a graph G is denoted by $G \mathbin{\widetilde{\nabla}} r$.

Definition 13 (Topological grad)**.** For a graph G and integer $r \geqslant 0$, the *topological greatest reduced average density (grad) at depth r*, is defined as

$$\widetilde{\nabla}_r(G) = \max_{H \in G \mathbin{\widetilde{\nabla}} r} |E(H)|/|V(H)|.$$

For a graph class $\mathcal{G}$, define $\widetilde{\nabla}_r(\mathcal{G}) = \sup_{G \in \mathcal{G}} \widetilde{\nabla}_r(G)$.

With the help of these definition we can define bounded expansion.

Definition 14 (Bounded expansion)**.** A graph class $\mathcal{G}$ has *bounded expansion* if there exists a function f such that for all r, we have $\widetilde{\nabla}_r(\mathcal{G}) < f(r)$.

When introduced, bounded expansion was originally defined using an equivalent characterization based on the notion of *shallow minors* [189]: H is a r-shallow minor of G if H can be obtained from G by contracting disjoint subgraphs of radius at most r and deleting vertices. In the context of this thesis, however, the topological shallow minor variant proves more useful, so we restrict our attention to this setting. Let us point out that bounded expansion implies bounded degeneracy, with $2f(0)$ being an upper bound on the degeneracy of the graphs.

Nowhere dense is a generalization of bounded expansion in which we measure the *clique number* instead of the edge density of shallow minors. Let $\omega(G)$ denote the size of the largest complete subgraph of a graph G and let $\omega(\mathcal{G}) = \sup_{G \in \mathcal{G}} \omega(G)$ be the natural extension to graph classes $\mathcal{G}$.

Definition 15 (Nowhere dense [191, 192])**.** A graph class $\mathcal{G}$ is *nowhere dense* if there exists a function f such that for all $r \in \mathbb{N}$ it holds that $\omega(\mathcal{G} \mathbin{\widetilde{\nabla}} r) < f(r)$.

There are many equivalent definitions [193]. A graph class is *somewhere dense* precisely when it is not nowhere dense.

As previously mentioned the other characterization of bounded expansion that will prove to be useful exploits treedepth. This characterization was first presented via the notion of a *p-centered coloring* which is based on the notion of a centered coloring (cf. Definition 6), which, as mentioned before, is equivalent to treedepth.

Definition 16 (p-centered coloring [189]). Given a graph G, let $c\colon V(G) \to \{1,\dots,r\}$ be a vertex coloring of G with r colors. We say that the coloring c is *p-centered*, for $p \geqslant 2$, if any connected subgraph of G either receives at least p colors or contains some color exactly once. Define $\chi_p(G)$ to be the minimum number of colors needed for a $(p+1)$-centered coloring.

While this definition looks rather cryptic, it is easy to see that every graph has a p-centered coloring for any p: simply assign a distinct color to each vertex of the graph. Note that p-centered colorings are proper colorings for $p \geqslant 2$ and in particular, χ_1 is precisely the chromatic number. Typically, the number of colors q is much larger than p and one is interested in minimizing q.

The following structural property, which follows directly from the equivalence between centered colorings and treedepth, make them an attractive tool for algorithm design.

Proposition 6 (Low treedepth colorings [190]). *Let $\mathcal{G}$ be a graph class of bounded expansion. There exists a function f such that for every $G \in \mathcal{G}$, $r \in \mathbb{N}$, the graph G can be colored with $f(r)$ colors so that any $i < r$ color classes induce a graph of treedepth $\leqslant i$ in G. Such a coloring can be computed in linear time.*

Nešetřil and Ossona de Mendez show that graph classes of bounded expansion are precisely those for which there exists a function f such that every member G of the graph class satisfies $\chi_p(G) \leqslant f(p)$ (see Theorem 7.1 [189]). The authors also showed how to obtain a p-centered coloring with at most $P(f(p))$ colors for each fixed p in linear time, where P is some polynomial of degree roughly 2^{2^p} [190]. We will make use of this algorithm in Section 21.

Part II

COMPUTING TREEDEPTH

4 COMPUTING TREEDEPTH IN LINEAR TIME

Formally, the Treedepth problem is to decide, given a graph G and an integer d, whether G has treedepth at most d. This decision problem is NP-complete even on co-bipartite graphs as shown by Pothen [204] and later by Bodlaender et al. [37]. On trees, the problem can be decided in linear time [216]. Deogun et al. [66] showed that Treedepth can be computed in polynomial time on the following graph classes: permutation, circular permutation, interval, circular-arc, trapezoid graphs and also on co-comparability graphs of bounded dimension. It is, however, NP-hard on chordal graphs [68]. The best-known approximation algorithm is due to Bodlaender et al. [39] and has performance ratio $O(\log^2 n)$. The best-known exact algorithm for this problem is due to Fomin, Giannopoulou and Pilipczuk [91] and runs in time $O^*(1.9602^n)$. For practical applications, several simple heuristics exist (see Section 24 and 26).

Concerning parameterized complexity, it is not difficult to prove that Treedepth is fixed-parameter tractable parameterized by the solution size. This follows from the fact that graphs of bounded treedepth are minor-closed and hence, by the celebrated Graph Minors Theorem of Robertson and Seymour, are characterized by a finite set of forbidden minors. One can test whether H is a minor of a graph G in time $O(f(h) \cdot n^3)$, where h is the number of vertices in H and f is some recursive function [210]. Therefore, for every fixed d, one can decide whether a graph contains as minor a member of the (finite) set that characterizes graphs of treedepth d in time $O(g(d) \cdot n^3)$, for some recursive function g which implies that the problem is fpt. We can do one better and use this property to show that there is a linear-fpt time algorithm thanks to bounded treedepth implying bounded treewidth. Testing if a minor exists is MSO-expressible. Therefore we can apply Courcelle's Theorem to test if a graph contains one of the forbidden minors, as pointed out by Ossona de Mendez and Nešetřil [193]. They also present the following as an open problem:

Problem ([193])**.** *Is there a simple linear time algorithm to check* $\mathbf{td}(G) \leqslant d$ *for fixed d?*

Bodlaender et al. developed a dynamic programming algorithm that takes as input a graph G and a tree decomposition of G of width w and decides whether G has treedepth at most d in time[1] $2^{O(w^2 d)} \cdot n^2$ [37]. In this paper we present a linear time algorithm that decides whether $\mathbf{td}(G) \leqslant d$ in time $2^{O(wd)} \cdot n$, improving both the dependence on w and n. If indeed $\mathbf{td}(G) \leqslant d$, then the algorithm also constructs a treedepth

1 We point out that the running time analysis in this work simply states that the algorithm runs in *polynomial* time for a fixed d and w. However, it is not difficult to restate the running time to include d and w as parameters, which is what we have done. In personal communication, H. Bodlaender suggested that the running time can be improved to $2^{O(w^2 d)} n$ [41].

decomposition within this time. That a better dynamic programming algorithm can be achieved using treedepth leads us to believe that representing the ranking of the vertices as a tree might be algorithmically helpful in other cases.

We can then, by using previous known characteristics of treedepth, easily extend this result to get the following two algorithms:

- A simple algorithm which runs in time $2^{2^{O(d)}} \cdot n$.
- A fast algorithm which runs in time $2^{O(d^2)} \cdot n$ using a 5-approximation for treewidth by Bodlaender et al. [38].

5 NICE TREEDEPTH DECOMPOSITIONS AND RESTRICTIONS

In this section we will introduce the necessary notions and lemmas we will need for the dynamic programming algorithm we present in Section 6.

NICE TREEDEPTH DECOMPOSITIONS

A treedepth decomposition of a graph is not unique. This is especially true since the definition allows to add unnecessary components to the treedepth decomposition without increasing its height. We introduce the notion of *trivially improvable treedepth decomposition* so that we can differentiate between treedepth decompositions which have such unnecessary nodes and those who do not.

Definition 17 (Trivially Improvable Treedepth Decompositions)**.** A treedepth decomposition T of a graph G is *trivially improvable* if $V(G) \subsetneq V(T)$.

We will also use extensively a special kind of treedepth decompositions that we will call *nice treedepth decompositions*. This notion is similar to that of *minimal trees* [91], the difference being that the properties that are directly enforced by the definition of a nice treedepth decomposition are only implied as a consequence of the definition in minimal trees.

Definition 18 (Nice Treedepth Decomposition)**.** A treedepth decomposition T of G is *nice* if the following conditions are met:

- T is not trivially improvable.
- For every node $x \in V(T)$, the subgraph of G induced by the nodes in T_x is connected.

In this section we will, for the sake of completeness, re-prove some known properties that can be enforced in treedepth decompositions. We will work with decompositions that are *not* trivially improvable. The next lemmas shows that one can always obtain such a decomposition from a trivially improvable one without increasing the height.

Lemma 1. *Let T be a trivially improvable treedepth decomposition of a graph G of height h. Let $x \in V(T) \setminus V(G)$ be a root of some tree in the decomposition T. Then the decomposition obtained by removing x is a treedepth decomposition of G with height at most h.*

Proof. Since $x \notin V(G)$, we have that $G \subseteq \operatorname{clos}(T - x)$. Thus $T - x$ is a treedepth decomposition of G. Clearly the height does not increase on deleting x. $\square$

Lemma 2. *Let T be a trivially improvable treedepth decomposition of a graph G with height h. Suppose that $x \in V(T) \setminus V(G)$ be a non-root node and let y be its parent in T. Then the treedepth decomposition obtained by contracting the edge xy is a treedepth decomposition of G with height at most h.*

Proof. Suppose T' is the forest obtained by contracting the edge xy. Then the height of T' is at most h. If $a, b \in V(T)$ is an ancestor-descendant pair that represents an edge of G, then these vertices form an ancestor-descendant pair in T' too. Thus T' is a treedepth decomposition of G with height at most h. □

Corollary 1. *Given a trivially improvable treedepth decomposition T of a graph G, one can obtain a decomposition of G that is not trivially improvable and a minor of T in time polynomial in $|T|$.*

Proof. Apply either Lemma 1 or 2 until $V(T) = V(G)$. □

The operations described in Lemma 1 and Lemma 2 do not increase the height of a decomposition. It therefore suffices to work with decompositions that are not trivially improvable. We will now use these results to prove certain properties of nice treedepth decompositions. In a sense, nice treedepth decompositions are those whose *structure* cannot be easily improved.

Lemma 3. *Every graph G admits a nice treedepth decomposition of height* $\mathbf{td}(G)$.

Proof. Let us assume G to be connected. If G has more than one component then we can apply this argument to each component in turn. By Corollary 1, it is sufficient to show that, given an optimal treedepth decomposition that is not trivially improvable, one can construct a decomposition of the same height that is nice. Therefore, let T be an optimal decomposition of G with root r that is not trivially improvable and let $x \in V(T)$ be a node at which the niceness condition is violated that has no descendant with the same property. That is, the subgraph $G[V(T_x)]$ of G induced by the vertices in the subtree of T rooted at x has more than one component. Let C be the set of children of x in T. For all $c \in C$ we are assuming that $G[V(T_c)]$ has a single component. These are precisely the components of $G[V(T_x) \setminus \{x\}]$. Let C' be the maximal set of children of x such that x *does not* have a neighbor in $V(T_{c'})$ for all $c' \in C'$. Compute a new treedepth decomposition T' by deleting every xc' edge for $c' \in C'$ and adding an edge between c' and the deepest node y in the path from r to x such that $yc' \in E(G)$ and no edge if no such node y exists. T' is clearly a valid treedepth decomposition of G. Notice that $G[T'_x]$ now has a single component and that we have not introduced any node which breaks the second property of nice treedepth decompositions. Thus this operation strictly decreases the number of nodes which break the property and by applying it repeatedly we can compute a nice treedepth decomposition in polynomial time. □

Computing a nice treedepth decomposition from a general treedepth decomposition can be done in $O(n + m \cdot \alpha(m))$ amortized time, where α is the inverse of the Ackermann function (see Section 24). As a result of Corollary 1 and the proof of Lemma 3, we obtain the following.

Corollary 2. *Let T be a treedepth decomposition of a graph G. One can compute in time polynomial in $|G|$, a nice treedepth decomposition T' with the following properties:*

1. *$height(T') \leqslant height(T)$;*
2. *for each vertex $x \in V(G)$, $height_{T'}(x) \leqslant height_T(x)$;*
3. *for any node $x \in V(T')$, we have that $A' \subseteq A$, where A are the ancestors of x in T and A' are the ancestors of x in T';*
4. *for any node $x \in V(T')$, we have that $D' \subseteq D$, where D are the descendants of x in T and D' are the descendants of x in T'.*

Given that one can transform any treedepth decomposition T into one that is nice and not trivially improvable in time polynomial in $|V(T)|$, we will henceforth assume that the treedepth decompositions we deal with have this property. Lastly, we prove some lemmas about nice treedepth decompositions that will be useful later.

Lemma 4. *Let T be a nice treedepth decomposition of a graph G. Let $x \in V(G)$ be a vertex such that x is not a leaf in T. If y is a child of x in T, then there exists an edge $xc \in E(G)$, for some $c \in V(T_y)$.*

Proof. Since T is a nice treedepth decomposition, the subtree T_x rooted at x induces a connected subgraph of G. From the definition of a treedepth decomposition, it follows that there can be no edge in G adjacent to a node of $V(T_y)$ and a node of $(V(T_x) \setminus \{x\}) \setminus V(T_y)$. From this it follows that for $G[V(T_x)]$ to be connected, there must be an edge between x and some node of T_y. $\square$

Thus every inner node in a nice treedepth decomposition has an edge to at least one of its descendants (in the graph represented by the decomposition).

Lemma 5. *Given a nice treedepth decomposition T of a graph G, let $x \in V(G)$ and let C be the children of x in T. For $C' \subseteq C$, let $T_x^{C'}$ denote the tree obtained from T_x by deleting the subtrees rooted at the vertices of $C \setminus C'$. Then $G[V(T_x^{C'})]$ is a connected subgraph of G.*

Proof. Since T is a nice treedepth decomposition, it follows that for every $c \in C$ the subtree T_c of T rooted at c induces a connected subgraph of G. From Lemma 4, it follows that x is connected to a node of T_c. Thus the lemma follows. $\square$

ROOTED GRAPHS

We will show that it suffices to work on *rooted graphs*. This is not fundamental to the algorithm, but it will make its description and proof of correctness easier, since it helps us to avoid dealing with forests which are not trees and special-casing our operations for the empty set.

Definition 19 (Rooted graph). A *rooted graph* $G = (V, E, r)$ is a graph with the specified *universal vertex* $r \in V(G)$ which is adjacent to every other vertex of G.

Lemma 6. *Let $G = (V, E, r)$ be a rooted graph with root r. Then there is an optimal treedepth decomposition T of G such that r is its root.*

Proof. Suppose that T' is an optimal treedepth decomposition of G with root $r' \neq r$ (since G is connected, T' is actually a tree). We assume that T' is not trivially improvable so that every node of T' is a vertex of G. Let $x_0, x_1, \ldots, x_p$ denote the vertices on the (r', r)-path in T', where $x_0 = r'$ and $x_p = r$. Then note that since T' is a treedepth decomposition and r is a universal vertex, for $0 \leqslant i \leqslant p-1$, x_i has exactly one child x_{i+1} in T'. That is, T' consists of the path $r', x_1, \ldots, x_{p-1}, r$ with subtrees attached to r. Transform T' to obtain T by exchanging the position of r and r'. Notice that $\operatorname{clos}(T) = \operatorname{clos}(T')$ and $\operatorname{height}(T) = \operatorname{height}(T')$. □

Corollary 3. *Let G be a rooted graph obtained by adding a universal vertex r to a graph G'. Then $\mathbf{td}(G) = \mathbf{td}(G') + 1$.*

Proof. To see that $\mathbf{td}(G) \leqslant \mathbf{td}(G') + 1$, take any optimal treedepth decomposition T' of G' and add edges between r and the roots of the forest of T'. This yields a treedepth decomposition of G of height $\mathbf{td}(G') + 1$. To see that $\mathbf{td}(G') \leqslant \mathbf{td}(G) - 1$, take an optimal treedepth decomposition T of G with root r (Lemma 6 guarantees the existence of such a decomposition). Now delete r from T to obtain a treedepth decomposition of G'. □

Lemma 6 motivates the following definition of treedepth decompositions of rooted graphs.

Definition 20 (Treedepth Decomposition of a Rooted Graph). A *treedepth decomposition T of a rooted graph $G = (V, E, r)$* is a treedepth decomposition of G whose root is r.

RESTRICTIONS AND PARTIAL DECOMPOSITIONS

In standard dynamic programming algorithms on tree decompositions a table is computed for every bag of the decomposition. The entries of these tables represent many partial solutions. By a partial solution we mean a solution that "covers" the graph restricted to the nodes in the current bag X and all bags that are descendants of X. For a more in-depth explanation of how standard dynamic programming algorithms on tree decompositions work see Section 13. We will now define what entries our tables will contain, namely structures we call *partial decompositions*. Then we will introduce a relation between treedepth decompositions and partial decompositions by defining what we call the *restriction* of a tree.

Definition 21 (Partial decomposition)**.** A *partial decomposition* is a triple (F, X, h), where

- F is a forest of rooted trees with $X \subseteq V(F)$; and,
- $h\colon V(F) \to \mathbb{N}^+$ is a *height function* which obeys the property that for nodes $x, y \in V(F)$ where x is an ancestor of y, $h(x) > h(y)$.

Definition 22 (Restriction of a partial decomposition)**.** The *restriction of a partial decomposition (F, X, h) to $\emptyset \neq X' \subseteq X$* is the partial decomposition (F', X', h'), where F' is obtained by iteratively deleting the leaves of the forest F that are *not* in X'. The height function h' is obtained from h by restricting it to $V(F')$.

Notice that F' is an induced subgraph of F in the above definition. We want the restriction of a tree to be closed under isomorphism. For this we introduce *partial decomposition equivalency*. This notion will also be key in keeping the tables during the dynamic programming small.

Definition 23 (Partial decomposition equivalency)**.** Two given partial decompositions (F_1, X_1, h_1) and (F_2, X_2, h_2) are *equivalent* if $X_1 = X_2$ and there exists a bijective function $\psi\colon V(F_1) \to V(F_2)$ such that

- the function ψ expresses an isomorphism between F_1 and F_2,
- $\psi|_{X_1}$ is the identity function,
- $h_1(v) = h_2(\psi(v))$ for every node v in the forest F_1.

We can now formalize a relation between trees (and as such treedepth decompositions) and partial decompositions.

Definition 24 (Restriction of a tree)**.** Given a tree T, let $(T, V(T), h)$ be the partial decomposition where $h(x)$ is the height of x in T for all $x \in V(T)$. A partial decomposition (F', X, h') is a *restriction* of T if (F', X, h') is equivalent to the restriction (F, X, h) we get from restricting $(T, V(T), h)$ to X. We call the function $\psi\colon V(F') \to V(F)$ that witnesses the equivalency as per Definition 23 of these two restrictions the *witness of the restriction*.

Notice that given a tree T and its restriction (F, X, h) to X with the corresponding witness ψ it follows that the induced subgraph $T[\psi(V(F))]$ is isomorphic to F. Since we are going to use partial decompositions to represent treedepth decompositions of a graph we need to introduce some notion of their height.

Definition 25 (Height of a partial decomposition)**.** Let (F, X, h) be a partial decomposition and let R be the set of all roots in F. The height of (F, X, h) is $\max_{x \in R} h(x)$.

Clearly two equivalent partial decompositions have the same height. For a specific set X and a graph G, the restrictions to X define equivalence classes for all treedepth decompositions of G. Later we will show that it suffices to keep a representative for certain equivalence classes during the dynamic programming.

As we move from the leaves to the root of the tree decomposition we will need a relationship between the entries of the table from the previous step and the new ones for the current table, such that the predecessor relationship is maintained. The following definitions will be used to make sure that this relation is kept intact.

Definition 26 (Topological generalization)**.** Let F_1, F_2 be rooted forests and let X be a set of vertices such that $X \subseteq V(F_1) \cap V(F_2)$. We say F_1 *topologically generalizes* F_2 under X if there exists an injective mapping $f \colon V(F_2) \to V(F_1)$ where the following conditions hold:

- $f|_X$ is the identity function.
- For any node $x \in V(F_2)$ and an ancestor y of x, $f(y)$ is an ancestor of $f(x)$ in F_1.

We say that a partial decomposition (F_1, X_1, h_1) *topologically generalizes a partial decomposition* (F_2, X_2, h_2) if $X_2 \subseteq X_1$ and F_1 topologically generalizes F_2 under X_2.

We will now prove some basic properties of restrictions which will be useful later on.

Lemma 7. *Let (F, X, h) be a partial decomposition. For $X' \subseteq X$, let (F', X', h') be the restriction of (F, X, h) to X'. Then for any $X'' \subseteq X'$, the restrictions of (F', X', h') and (F, X, h) to X'' are identical.*

Proof. First observe that if x is a leaf in F then for any $y \neq x$, x is a leaf in $F - y$. Moreover if we restrict the decomposition (F, X, h) to X'', then the only leaves of the forest are elements of X''. Suppose that the restrictions of (F', X', h') and (F, X, h) to X'' yields (respectively) the decompositions $(\widetilde{F}', X'', \widetilde{h}')$ and $(\widetilde{F}, X'', \widetilde{h})$. Let $s_1 = v_1, \ldots, v_p$ be the sequence in which vertices were deleted to obtain $(\widetilde{F}', X'', \widetilde{h}')$ from (F, X, h); and, $s_2 = w_1, \ldots, w_q$ were the vertices that were deleted to obtain $(\widetilde{F}, X'', \widetilde{h})$ from (F, X, h).

Suppose that there exists a node y in the sequence s_1 that does *not* occur in s_2 and suppose that $v_{\ell+1}$ is the first such node of s_1. Note that $v_{\ell+1}$ is a leaf after the vertices $v_1, \ldots, v_\ell$ are deleted, irrespective of the order of deletion. Since the vertices $v_1, \ldots, v_\ell$ occur in s_2, suppose that w_i is the last of these that occurs in s_2. Then after the deletion of w_i (in the sequence s_2), the node $v_{\ell+1}$ remains as a leaf and this fact does not change with further deletions down the sequence s_2. But $v_{\ell+1}$ was deleted in the sequence s_1 and hence $v \notin X''$ and the fact that v does not appear in the sequence s_2 implies that $\widetilde{F}$ has a leaf node that is not an element of X'', a contradiction. This shows that every node of s_1 appears in s_2. Reversing the argument, one sees that every node in s_2

appears in s_1. Hence s_1 and s_2 contain the same vertices, possibly in a different order. Therefore $V(\widetilde{F}) = V(\widetilde{F}')$ and the partial decompositions $(\widetilde{F}', X'', \widetilde{h}')$ and $(\widetilde{F}, X'', \widetilde{h})$ are identical. □

Lemma 7 immediately implies the following.

Corollary 4. *Let (F, X, h) be a partial decomposition and let $X' \subseteq X$. The restriction of (F, X, h) on X' is unique up to isomorphism.*

Importantly, the number of vertices in the forest of a restriction is at most $|X| \cdot d$, where d is the treedepth of the graph. This follows since every leaf of the forest is an element of X and the number of vertices from any root to leaf path is at most d. Notice furthermore that the height of a restriction of a partial decomposition or a tree is the same as the height of the partial decomposition or tree respectively.

Lemma 8. *Let T be a (not trivially improvable) treedepth decomposition of a graph G, $X' \subseteq X \subseteq V(G)$ and let F and F' be the forests of the decomposition T when restricted to the sets X and X', respectively. Then F topologically generalizes F' under X'.*

Proof. Note that $V(F') \subseteq V(F)$ and hence the function $f\colon V(F') \to V(F)$ defined by $f(x) = x$ for all $x \in V(F')$ witnesses that F topologically generalizes F'. □

We not only need to understand the relations between partial decompositions and treedepth decompositions, but also how these relate to each other when considering subgraphs.

Lemma 9. *Let $G = (V, E, r)$ be a rooted graph, let $G' = (V', E', r)$ be a rooted subgraph of G and $X \subseteq V(G')$ be a set of nodes. Further, let T be a nice treedepth decomposition of G and T' be a nice treedepth decomposition of G' computed from T as per Corollary 2. Let $(F, X, h), (F', X, h')$ be respective restrictions of T, T' to X. Then for every pair of functions ψ and ψ' that witness that (F, X, h) is a restriction of T to X and (F', X, h') is a restriction of T' to X respectively, it holds that $\psi'(F') \subseteq \psi(F)$.*

Proof. Assume to the contrary that there exist functions ψ, ψ' such that there exists $v \in F'$ with $\psi'(v) \notin \psi(F)$. First note that $v \notin X$ as $X \subseteq \psi(F)$. Since v is retained in F', there exists a successor $y \in X$ of v in F'. But by Corollary 2, the ancestor relationship of vertices in T' is preserved in T, therefore y is also a successor of v in T. But then, by construction of F, the vertex v must also be contained in $\psi(F)$. □

6 DYNAMIC PROGRAMMING ALGORITHM

In this section we present an algorithm which takes as input a triple $(G, \mathcal{T}, d)$, where G is a graph, $\mathcal{T}$ a tree decomposition of G of width w and d an integer, and decides whether $\mathbf{td}(G) \leqslant d$ in time $2^{O(wd)} \cdot n$. For yes-instances, the algorithm can be modified to output a treedepth decomposition by backtracking.

MAIN ALGORITHM

Our algorithm is a dynamic programming algorithm. It works by creating tables of *partial decompositions*. The operations will be the standard join, forget and introduce operations for dynamic programming algorithms on tree decompositions. Thus, every operation of the algorithm will take one or two sets of partial decompositions and create a new set of partial decompositions. More specifically, such an operation will be done for every bag of the tree decomposition. These partial decompositions will be restrictions to the current bag of tree decompositions of the part of the graph we have seen up to this point.

Definition 27 (Forgetting a vertex from a partial decomposition)**.** Let G be a graph, let $X \subseteq V(G)$ and let R' be a set of partial decompositions on the set X. For a vertex $u \in X$, the forget operation on u denoted by *forget*(R', X, u) is defined to be a set A of partial decompositions obtained as follows: Initially set $A \leftarrow \varnothing$; for every partial decomposition $(F', X', h') \in R'$, consider its restriction to the set $X \setminus \{u\}$ and add it to the set A only if it is *not* equivalent to any member in A.

The introduce operation is somewhat more involved. The general idea is that given a set R' of partial decompositions of the form (F', X', h') where $X' \subseteq V(G)$, the result of introducing $u \in V(G) \setminus X'$ is a set A of partial decompositions whose elements (F, X, h) are computed as follows:

1. For every reasonable forest F look for a partial decomposition $(F', X', h') \in R'$ such that F topologically generalizes F'. If no such partial decomposition exists, discard F.
2. Given F and F', for every function f that witnesses F topologically generalizing F', create a partial decomposition of the form $(F, X = X' \cup \{u\}, h)$, for some appropriate h.
3. Add (F, X, h) to A if its height is smaller than d and there is no equivalent partial decomposition already contained in A.

Formally, the process outlined in the list above translates to the following.

Definition 28 (Vertex introduction into a partial decomposition)**.** Let $G = (V, E, r)$ be a rooted graph, let $X' \subseteq V(G)$ and let R' be a set of partial decompositions of the form (F', X', h'). For a vertex $u \in V(G) \setminus X'$ and an integer d, the introduction operation on u, denoted by $\mathit{intro}_d(R', X', u, G)$, is defined to be a set A of partial decompositions constructed as follows:

Let $X = X' \cup \{u\}$. Initialize $S \leftarrow \varnothing$. Generate every tree F with up to $|X| \cdot d$ vertices which fulfills the following properties:

- The node r is the root of F.
- The depth of F is $\leqslant d$.
- The set $X \subseteq V(F)$.
- All leaves of F are in X.
- $E(G[X]) \subseteq E(\text{clos}(F)[X])$.

For every partial decomposition $(F', X', h') \in R'$ and every function $f \colon V(F') \to V(F)$ that witnesses that F topologically generalizes F' on the set $X \setminus \{u\}$ add the tuple $(F, (F', X', h'), f)$ to S if $f(F') = V(F) \setminus \{u\}$.

For every $(F, (F', X', h'), f) \in S$ compute the partial decomposition (F, X, h), where h is defined recursively by visiting the vertices of F in depth-first post-order fashion: Let $z \in F$ and let C be the set of children of z in F. When z is visited, if $z \neq u$ and there exists a node $z' \in V(F')$ such that $f(z') = z$, set $h(z) = \max\{\max_{c \in C} h(c) + 1, h'(z')\}$. Else for any other node $z \in V(F)$ set $h(z) = \max_{c \in C} h(c) + 1$, where we define the maximum over the empty set to be zero. Finally add the partial decomposition (F, X, h) to the set A, if its height is smaller that d and A does not contain an equivalent partial decomposition to (F, X, h).

Lastly, we describe the join operation. Here we take two tables of restrictions on X for two graphs G_1 and G_2 which intersect in X and compute a single table containing restrictions on the union of G_1 and G_2.

Definition 29 (Joining Partial Decompositions)**.** Let $G = (V, E, r)$ be a rooted graph. Let R_1 and R_2 be two sets of partial decompositions on $X \subseteq V(G)$. Let d be an integer. Then the join operation join_d is defined via $\mathit{join}_d(X, R_1, R_2, G) = A$, where A is a set of partial decompositions which is constructed as follows:

Initialize $S \leftarrow \varnothing$. Generate every tree F with up to $|X| \cdot d$ vertices which fulfills the following properties:

- r is the root of F.
- $X \subseteq V(F)$.
- All leaves of F are in X.

Take every pair of partial decompositions $(F_1, X, h_1) \in R_1$ and $(F_2, X, h_2) \in R_2$ and every pair of functions f_1 and f_2 which witness that F topologically generalizes F_1 and F_2 on the set X respectively. Add the tuple $(F, (F_1, X, h_1), (F_2, X, h_2), f_1, f_2)$ to S if $f_1(F_1) \cap f_2(F_2) = X$ and $f_1(F_1) \cup f_2(F_2) = V(F)$.

For every $(F, (F_1, X, h_1), (F_2, X, h_2), f_1, f_2) \in S$ we get one partial decomposition (F, X, h) where h is defined as follows: The function h is defined recursively by visiting the vertices of F in depth-first post-order fashion. Let $z \in F$ and let C be the set of children of z in F. Let $\alpha_1 = h_1(z_1)$ if there exists a node z_1 such that $f_1(z_1) = z$ and $\alpha_1 = 1$ otherwise. Analogously, let $\alpha_2 = h_2(z_2)$ if there exists a node z_2 such that $f_2(z_2) = z$ and $\alpha_2 = 1$ otherwise. Then we compute the height of z as $h(z) = \max\{\max_{c \in C} h(c) + 1, \alpha_1, \alpha_2\}$.

Finally add the partial decomposition (F, X, h) to the set A, if its height is smaller that d and A does not contain an equivalent partial decomposition to (F, X, h).

The main algorithm can be found in Algorithm 1. We claim that this algorithm correctly decides, given an n-vertex graph G and a tree decomposition of width at most w, whether G has treedepth at most d into time $2^{O(wd)} \cdot n$.

CORRECTNESS OF DYNAMIC PROGRAMMING ALGORITHM

Our proof can be divided into the following steps:

1. We showed that every graph G admits a nice treedepth decomposition of height $\mathbf{td}(G)$ (Lemma 3).
2. We showed that it is sufficient to work with rooted graphs and that such graphs have an optimal nice treedepth decomposition T such that root of T is the root of graph (Lemma 6).
3. We defined the restriction of a tree. Since in this context we treat treedepth decompositions as trees, this will provide a relationship between treedepth decompositions and partial decompositions (Definition 31).
4. We will show that for any nice treedepth decomposition of the graph, our table contains its restriction (Lemma 10);
5. and that every partial decomposition contained in the table is a restriction of some treedepth decomposition of the graph (Lemma 11).

All this together achieves the desired result.

As seen Algorithm 1, we use the contents of the bags of a tree decomposition as the set on which we restrict. Since we work on rooted graphs and the root r of the graph is contained in every bag any restriction of a tree on the set X will be a tree. We will enforce that r is always the root of this tree. Since its depth will be at most d, its size will be bounded by $|X| \cdot d$.

```
Input: A graph $G'$, an integer $d$ and a nice rooted tree decomposition $\mathcal{T}'$ of $G'$ with root
       bag $X$.
Output: True if the treedepth of $G'$ is at most $d$ and False otherwise.
Add a universal vertex $r \notin V(G')$ to the graph $G'$ to obtain $G$;
Obtain a nice tree decomposition $\mathcal{T}$ of $G$ as follows;
    start with $\mathcal{T} := \mathcal{T}'$;
    add $r$ to every bag of $\mathcal{T}$;
    for every leaf bag of $\mathcal{T}$, add $\{r\}$ as a child-bag;
$R :=$ treedepth-rec($G, d+1, \mathcal{T}, X$);
return $R \neq \emptyset$;
```

Algorithm 1: treedepth

```
Input: A rooted graph $G = (V, E, r)$, an integer $d$ and a tree decomposition $\mathcal{T}$ of $G$
       containing $r$ in every bag and a bag $X$ of $\mathcal{T}$.
Output: A set $R$ of partial decompositions.
$R := \emptyset$;
if $X$ is a leaf then
    $r :=$ the only vertex contained in $X$;
    $F :=$ a tree consisting of just the node $r$;
    $h$ is a function which is only defined for $r$ and $h(r) = 1$;
    $R := \{(F, \{r\}, h)\}$;
else if $X$ is a forget bag then
    $u :=$ forgotten vertex;
    $X' :=$ the child of $X$;
    $R' :=$ treedepth-rec($G, d, \mathcal{T}, X'$);
    $R := forget(R', X', u)$;
else if $X$ is an introduce bag then
    $u :=$ introduced vertex;
    $X' :=$ the child of $X$;
    $R' :=$ treedepth-rec($G, d, \mathcal{T}, X'$);
    $R := intro_d(R', X', u, G)$;
else if $X$ is a join bag then
    $\{X_1, X_2\} :=$ the set of children of $X$;
    $R_1 :=$ treedepth-rec($G, d, \mathcal{T}, X_1$);
    $R_2 :=$ treedepth-rec($G, d, \mathcal{T}, X_2$);
    $R := join_d(X, R_1, R_2, G)$;
return $R$;
```

Algorithm 2: treedepth-rec

Lemma 10. *Let Algorithm 2 be called on $(G, d, \mathcal{T}, X)$, where G is a graph rooted at r, the remaining parameters $d, \mathcal{T}, X$ are as described in the algorithm. Then for every nice treedepth decomposition T of height at most d rooted at r of $G[V(\mathcal{T}_X)]$, the set R returned by the algorithm contains a restriction of T to the set X.*

Proof. We will prove this by structural induction over tree decompositions: Consider the case that the tree decompositions consists of a single leaf bag. Remember that Algorithm 2 works on nice tree decompositions whose leaves contain a single vertex. The returned set R then consists of the unique partial decomposition for a graph with a single vertex. In the following we will often consider induced graphs $G[V(\mathcal{T}_X)]$ for some bag X. Notice that by the way the algorithm works the root r is contained in all bags and as such is also a root of such a subgraph. We will thus assume in the following that any treedepth decomposition of such a induced subgraph has r as its root.

FORGET CASE If X is a forget bag whose single child in $\mathcal{T}$ is the bag X', then the if-clause at line 7 is entered. By induction hypothesis, we assume that R' contains a restriction to the set X' of every nice treedepth decomposition T rooted at r of the graph $G[V(\mathcal{T}_{X'})]$. Fix such a T and let $(F', X', h') \in R'$ be a restriction of T to the set X'. Notice that $G[V(\mathcal{T}_{X'})] = G[V(\mathcal{T}_X)]$. Therefore by Corollary 4, we can restrict (F', X', h') to the set X to obtain a restriction (F, X, h) of T to X. By the definition of the forget operation (Definition 32) the restriction of (F', X', h') to X is added to R.

INTRODUCE CASE If X is an introduce bag whose single child in $\mathcal{T}$ is the bag X', then the if-clause at line 12 is entered. Fix a nice treedepth decomposition T rooted at r of the graph $G[V(\mathcal{T}_X)]$ of height at most d. We want to show that a partial decomposition (F, X, h) is contained in R, which is a restriction of T to X. Note that the treedepth decomposition T is also a treedepth decomposition of $G[V(\mathcal{T}_{X'})]$. Let T' be the nice treedepth decomposition of $G[V(\mathcal{T}_{X'})]$ computed from T using Corollary 2. The difference between T and T' is a single contraction of the introduced node u into its parent. By induction hypothesis we assume that R' contains a restriction (F', X', h') of T' to X'. Since (F, X, h) is a restriction, the height of F is at most d, its leaves are in X and r is its only root. Therefore F has at most $|X| \cdot d$ vertices. This means that at some point the introduce operation will generate F, since all trees which comply with these characteristics are enumerated. By Lemma 8 the tree F topologically generalizes F'. Thus a tuple (F, X, h) will be added to the set R of the introduce function from Definition 28 and it is left to show that h is computed correctly.

Let (F_T, X, h'_T) be the restriction of $(T, V(T), h_T)$ to X, where h_T is the height function for the nodes of T. By definition, F and F_T must be isomorphic and there exists a witness of this fact ψ such that $\psi|_X$ is the identity function. Let ψ' be function that witnesses that (F', X', h') is a restriction of T' to X'. Consider the subgraph $F'_{T'} = T'[\psi'(V(F'))]$. Since T' is the result of contracting u into its parent in T, together with F_T and $F'_{T'}$ being connected subgraphs of T and T' respectively such that $V(F'_{T'}) \subset V(F_T)$ and $V(F_T) \setminus V(F'_{T'}) = u$ it follows that contracting u into its parent in F_T results in $F'_{T'}$. By inductive hypothesis (F', X', h') is a restriction of T' to X' and as such it follows that $h'(\psi'^{-1}(x))$ is the height of x in T' for any node $x \in V(F'_{T'})$. The height of such a node x differs in T and T' only if u is an ancestor of x in T. This means that $h(y) = h'_T(\psi(y))$ for any $y \in V(F)$ which is not an ascendant of u in F, especially for all descendants of u in F. This implies that $h(u) = h'_T(u)$. The function furthermore only updates the value of an ascendant a of u, if the subtree T_c rooted at a child c of a that contains u now decides the height of a. This means that h is correctly set.

JOIN CASE Finally, if X is a join bag with two children X_1 and X_2 in $\mathcal{T}$, then the if-clause at line 17 is entered. Let again T be a nice treedepth decomposition rooted at r of the graph $G[V(\mathcal{T}_X)]$. Then T is also a treedepth decomposition of both $G[V(\mathcal{T}_{X_1})]$ and $G[V(\mathcal{T}_{X_2})]$. Notice that by the properties of tree decompositions, $V(\mathcal{T}_{X_1}) \cap V(\mathcal{T}_{X_2}) = X$. Let (F, X, h) be a restriction of T to the set X for the graph $G[V(\mathcal{T}_X)]$. Let T_1 and T_2 be nice treedepth decomposition computed from T by Corollary 2 for the graphs $G[V(\mathcal{T}_{X_1})]$ and $G[V(\mathcal{T}_{X_2})]$ respectively. By inductive hypothesis there exists partial decompositions $(F_1, X, h_1) \in R_1$ and $(F_2, X, h_2) \in R_2$ such that (F_1, X, h_1) is a restriction of T_1 and (F_2, X, h_2) is a restriction of T_2. At some point the introduce operation will generate F for the same reason as in the introduce case. By Lemma 8 F is a topological generalization of F_1 and F_2. We now need to show that there exist two witness functions f_1 and f_2 respectively such that the intersection of their images is exactly X.

Let (F_T, X, h'_T) be the restriction of $(T, V(T), h_T)$ to X, where h_T is the height function for the nodes of T. By definition, F and F_T must be isomorphic and there exists a witness of this fact ψ such that $\psi|_X$ is the identity function. Let ψ_1, ψ_2 witness that $(F_1, X, h_1), (F_2, X, h_2)$ are restrictions of T_1 and T_2 to X, respectively. By Lemma 9 we have that $\psi_1(V(F_1)) \subseteq \psi(V(F))$ and $\psi_2(V(F_2)) \subseteq \psi(V(F))$. Therefore we can construct $f_1 = \psi^{-1} \circ \psi_1$ and $f_2 = \psi^{-1} \circ \psi_2$, both of which are well-defined. It remains to show that $f_1(V(F_1)) \cap f_2(V(F_2)) = X$. By construction we already see that $f_1(X) = f_2(X) = X$. Since $\psi_1(V(F_1)) \subseteq V(T_1)$ and $\psi_2(V(F_2)) \subseteq V(T_2)$ with $V(T_1) \cap V(T_2) = X$, the claim follows. Since f_1, f_2 exist, the join operation will generate them at some point. Therefore a partial decomposition whose tree is F will be added to the result set. It remains to show that the height function as computed in the join operation is correct.

Let us first show the following: let $z \in V(T) \setminus V(F_T)$ be a node whose parent is contained in F_T. Then either $V(T_z) \cap V(\mathcal{T}_{X_1}) = \emptyset$ or $V(T_z) \cap V(\mathcal{T}_{X_2}) = \emptyset$. Assume to the contrary that T_z contains vertices of both $V(\mathcal{T}_{X_1})$ and $V(\mathcal{T}_{X_2})$. Since X separates these two sets in $G[V(\mathcal{T}_X)]$ and by assumption no vertex of X is contained in T_z this implies that $G[V(T_z)]$ has more than one connected component. This contradicts T being a nice treedepth decomposition.

The remaining proof parallels the proof for the introduce case. Let z be a leaf of F_T, C_1 be the set of children of z in T contained in $V(\mathcal{T}_{X_1})$ and C_2 the set of children contained in $V(\mathcal{T}_{X_2})$. Notice that since z is a leaf of F_T, the set $C_1 \cup C_2$ does not contain any element of X. By Lemma 5, the tree $T_z^{C_1}$ induces a connected subgraph in $G[V(\mathcal{T}_{X_1})]$ and the tree $T_z^{C_2}$ induces a connected subgraph in $G[V(\mathcal{T}_{X_2})]$. The trees $T_z^{C_1}, T_z^{C_2}$ are by construction subtrees of T_1 and T_2, respectively.

We will now show that the height function h is computed correctly for the leaves of F. The height of z in T is either the height of $T_z^{C_1}$ or of $T_z^{C_2}$. By the previous observation these two trees are subtrees of respectively T_1 and T_2 and by induction hypothesis their heights are given by $h_1(z)$ and $h_2(z)$, respectively. As the height of z is computed as $h(z) = \max\{h_1(z), h_2(z)\}$ we conclude that the height of the leaves of F is correct.

We can now prove inductively that the height $h(z)$ for any internal node z is also computed correctly. Let C be the set of children of z in T which are not nodes of F. Define $C_1 = C \cap V(\mathcal{T}_{X_1})$ and $C_2 = C \cap V(\mathcal{T}_{X_2})$, both of which could potentially be empty. As previously stated, $T_z^{C_1}$ and $T_z^{C_2}$ induce connected subgraphs in $G[V(\mathcal{T}_{X_1})]$ and $G[V(\mathcal{T}_{X_2})]$ and are subtrees of T_1 and T_2, respectively. From Corollary 2 we know that the height of z in T is at least the height of z in T_1 (if it is contained in T_1) and the height of z in T_2 (if it is contained in T_2).

Thus it follows that if the height of z in T equals the height of $T_z^{C_1}$, then this it also equals the height of z in T_1. Analogously, if the height of z in T equals the height of $T_z^{C_2}$, then this it also equals the height of z in T_2. Taking the maximum of $h_1(\psi_1^{-1}(z))$ and $h_2(\psi_2^{-1}(z))$ (if the inverse values exist) and all the children of $\psi^{-1}(z)$ in F therefore yields the correct value for $h(\psi^{-1}(z))$.

Since these are all the possible execution paths of the algorithm, it follows by induction that the lemma is correct. □

We have now shown that our algorithm will contain a partial decomposition representing any nice treedepth decomposition of height at most d. This is not sufficient to prove the correctness of the algorithm since our tables could still contain partial decomposition which are not restrictions of treedepth decompositions of height at most d. The next lemma proves precisely that this is not the case.

Lemma 11. *Let Algorithm 2 be called on $(G, d, \mathcal{T}, X)$, where G is a graph rooted at r, the remaining parameters $d, \mathcal{T}, X$ are as described in the algorithm. Then every member of R returned by the algorithm is a restriction of a treedepth decomposition of $G[V(\mathcal{T}_X)]$ to X.*

Proof. We will prove this by structural induction over tree decompositions: Consider the case that the tree decomposition consists of a single leaf bag containing only a single vertex. The returned set R then consists of the unique partial decomposition for this graph.

FORGET CASE For the forget case, the correctness of the statement follows directly from Lemma 7 using the induction hypothesis.

INTRODUCE CASE Consider the case that the bag X with single child X' introduces the vertex u. The set R' contains, by induction hypothesis, only restrictions of treedepth decompositions. We have to show that the operation of introducing u generates again only restrictions of treedepth decompositions. Consider any $(F, X, h) \in R$. First let us show that every edge incident to u in $G[V(\mathcal{T}_X)]$ is contained in $\text{clos}(F)$. Because X' separates u from $G[V(\mathcal{T}_{X'}) \setminus X]$, any such edge has its other endpoint necessarily in X'. Since the introduce operation by construction only returns restrictions with $E(G[X]) \subseteq E(\text{clos}(F)[X])$, we conclude that every edge incident to u in $G[V(\mathcal{T}_X)]$ is contained in the closure of F.

Consider $(F', X', h') \in R'$ such that F topologically generalizes F' and such that $(F, X, h) \in \mathit{intro}_d(\{(F', X', h')\}, X', u, G)$. Such a restriction must, by the definition of the introduce operation, exist and by induction hypothesis be a restriction of a treedepth decomposition T' of $G[V(\mathcal{T}_{X'})]$. Note that every edge $vw \in E(G[V(\mathcal{T}_X)])$ with $v \neq u \neq w$ is by induction hypothesis contained in the closure of T'.

We will now show that we can construct a treedepth decomposition T of $G[V(\mathcal{T}_X)]$ from T' of which (F, X, h) is a restriction. Let ψ' witness that (F', X', h') is a restriction of T' to X'. Let $f \colon V(F') \to V(F)$ be a function that witnesses that F topologically generalizes F' with $u \notin f(F')$. We first construct $(\hat{F}, X, \hat{h}), (\hat{F}', X', \hat{h}')$ which are equivalent to $(F, X, h), (F', X', h')$, respectively, such that $V(\hat{F}') \subset V(\hat{F}) \subseteq V(T') \cup \{u\}$ and so that the function f carried over to $\hat{F}', \hat{F}$ is simply the identity.

By Definition 31, there exists $\hat{F}' \subseteq T'$ and $\hat{h}'$ such that $(\hat{F}', X', \hat{h}')$ is a restriction of T' to X'. Let $\hat{\psi}' \colon V(\hat{F}') \to V(F')$ be the function that witnesses the equivalency of $(\hat{F}', X', \hat{h}')$ and (F', X', h'). Then $\hat{F}$ is the tree with nodes $V(\hat{F}) = V(\hat{F}') \cup \{u\}$ isomorphic to F where the isomorphism is witnessed by the bijection $\phi \colon V(\hat{F}) \to V(F)$ defined via

$$\phi(v) = \begin{cases} v & \text{for } v = u \\ f(\hat{\psi}'(v)) & \text{otherwise} \end{cases}$$

and $\hat{h} = h \circ \phi$. We finally construct T as follows: take the rooted forest $T' \setminus \hat{F}'$ and add $\hat{F}$ to it, then add the edge set $\{xy \in E(T') \mid x \in \hat{F}', y \notin \hat{F}'\}$.

Let us first verify that $(\hat{F}, X, \hat{h})$, and thus by equivalency also (F, X, h), is indeed a restriction of T to X. By construction it is immediately apparent that the iterative deletion of leaves of T not in X indeed yields the tree $\hat{F}$. However, we also need to verify that the height function $\hat{h}$ is correct.

We prove the correctness of $\hat{h}$ inductively beginning at the leaves of $\hat{F}$: consider a leaf $v \in \hat{F}$ with $v \neq u$. The introduce operation sets the value $\hat{h}(v)$ to $h(\phi(v)) = h'(\hat{\psi}'(v)) = \hat{h}'(v)$. By construction, v in T inherits the subtrees of v in T', thus $\text{height}_T(v) = \text{height}_{T'}(v) = \hat{h}(v)$. Next assume u is a leaf in $\hat{F}$: then the introduce operation sets $\hat{h}(v) = 1$. By construction of T, u will then not have any children and we conclude that $\text{height}_T(u) = \hat{h}(u)$ in this case. The statement now follows by induction: consider any internal node $v \in \hat{F}$, $v \neq u$ with children C in T. Let C' be the set of children of v in T'.

By induction hypothesis, for all $w \in C \cap V(\hat{F})$, $\text{height}_T(w) = \hat{h}(w)$. By construction of T, it holds that

$$\max_{w \in C \setminus V(\hat{F})} \text{height}_T(w) = \max_{w \in C' \setminus V(\hat{F}')} \text{height}_{T'}(w). \tag{6.1}$$

By construction of T and the fact that $\hat{F}$ is a topological generalization of $\hat{F}'$, it holds that no node can have fewer descendants in $\hat{F}$ than $\hat{F}'$ and thus

$$\max_{w \in C' \cap V(\hat{F}')} \text{height}_{T'}(w) \leqslant \max_{w \in C \cap V(\hat{F})} \text{height}_T(w). \tag{6.2}$$

Further note that by the induction hypothesis

$$\begin{aligned} \hat{h}'(v) - 1 &= \max_{w \in C'} \text{height}_{T'}(w) \\ &= \max\{ \max_{w \in C' \setminus V(\hat{F}')} \text{height}_{T'}(w), \max_{w \in C' \cap V(\hat{F}')} \text{height}_{T'}(w) \}. \end{aligned} \tag{6.3}$$

Therefore it holds that

$$\begin{aligned} \max_{w \in C} \text{height}_T(w) &= \max\{ \max_{w \in C \setminus V(\hat{F})} \text{height}_T(w), \max_{w \in C \cap V(\hat{F})} \text{height}_T(w) \} \\ &= \max\{ \max_{w \in C' \setminus V(\hat{F}')} \text{height}_{T'}(w), \max_{w \in C \cap V(\hat{F})} \text{height}_T(w) \} && \text{(by 6.1)} \\ &= \max\{ \max_{w \in C' \setminus V(\hat{F}')} \text{height}_{T'}(w), \max_{w \in C' \cap V(\hat{F}')} \text{height}_{T'}(w), \\ &\qquad \max_{w \in C \cap V(\hat{F})} \text{height}_T(w) \} && \text{(by 6.2)} \\ &= \max\{ \hat{h}'(v) - 1, \max_{w \in C \cap V(\hat{F})} \text{height}_T(w) \} && \text{(by 6.3)} \\ &= \max\{ \hat{h}'(v) - 1, \max_{w \in C \cap V(\hat{F})} \hat{h}(w) \} && \text{(by induction)} \\ &= \hat{h}(v) - 1 && \text{(by introduce operation.)} \end{aligned}$$

The proof for $\hat{h}(u)$ works analogously, with the slight difference that u will not have any children that are not in $\hat{F}$. We conclude that $(\hat{F}, X, \hat{h})$ and therefore (F, X, h) is a restriction of T to X.

It remains to show that T is a treedepth decomposition of $G[V(\mathcal{T}_X)]$. Note that $V(T) = V(\mathcal{T}_X)$. By construction of F and thus $\hat{F}$, edges incident to u are contained in $\text{clos}(\hat{F})$ and thus in $\text{clos}(T)$. Since $\hat{F}$ is a topological generalization of $\hat{F}'$, $\text{clos}(\hat{F}') \subseteq \text{clos}(\hat{F})$ and therefore every edge of $G[V(\mathcal{T}_X)]$ that lives in $V(\hat{F}')$ is contained in the closure of T. As $T' \setminus \hat{F}'$ is a subgraph of T, the edges contained in $\text{clos}(T' \setminus \hat{F}')$ are contained in $\text{clos}(T)$. It remains to show that every edge xy that has one endpoint $x \in \hat{F}'$ and the other endpoint $y \in T' \setminus \hat{F}'$ will also be covered by the closure of T. Consider the x-y-path in T': this path contains a node $z \in \hat{F}'$ whose successor is not contained in $\hat{F}'$. Because $\hat{F}$ is a topological generalization of $\hat{F}'$, the node x is an ancestor of z in $\hat{F}$ and thus in T. Furthermore, by construction of T, the node z is an ancestor of y in T; it follows by transitivity that $xy \in \mathrm{clos}(T)$. Therefore T is a treedepth decomposition of $G[V(\mathcal{T}_X)]$ and the lemma follows for the introduce-case.

JOIN CASE Consider the case of a bag X with children $X_1 = X_2 = X$. The sets R_1, R_2 contain, by induction hypothesis, only restrictions of treedepth decompositions. We have to show that the operation of joining X_1, X_2 generates only restrictions of treedepth decompositions. Consider any $(F, X, h) \in R$, $(F_1, X, h_1) \in R_1$ and $(F_2, X, h_2) \in R_2$ such that F topologically generalizes both F_1 and F_2 and (F, X, h) is the result of joining the other two, i.e. $(F, X, h) \in join_d(X, \{(F_1, X, h_1)\}, \{(F_2, X, h_2)\}, G)$. The restrictions (F_1, X, h_1) and (F_2, X, h_2) must, by the definition of the join operation, exist and by induction hypothesis they are restrictions of treedepth decompositions T_1, T_2 of $G[V(\mathcal{T}_{X_1})]$ and $G[V(\mathcal{T}_{X_2})]$, respectively. Note that every edge $vw \in E(G[V(\mathcal{T}_X)])$ is either contained in the closure of T_1 or the closure of T_2. We will now show that we can construct a treedepth decomposition T of $G[V(\mathcal{T}_X)]$ from T_1, T_2 of which (F, X, h) is a restriction.

For $i \in \{1, 2\}$, let ψ_i witness that (F_i, X, h_i) is a restriction of T_i to X_i. Let $f_i \colon V(F_i) \to V(F)$ be a function that witnesses that F topologically generalizes F_i. We first construct $(\hat{F}, X, \hat{h}), (\hat{F}_i, X, \hat{h}_i), i \in \{1, 2\}$ which are equivalent to $(F, X, h), (F_i, X, h_i)$, respectively, such that $V(\hat{F}_i) \subseteq V(\hat{F}) \subseteq V(T_1) \cup V(T_2)$ and so that the functions f_i that witness the topological generalization of F_i by F simply become the identity on $\hat{F}_i, \hat{F}$. By Definition 31, there exists a subgraph $\hat{F}_i$ of T_i and a function $\hat{h}_i$ such that $(\hat{F}_i, X, \hat{h}_i)$ is a restriction of T_i to $X_i = X$. Let $\hat{\psi}_i \colon V(\hat{F}_i) \to V(F_i)$ be the function that witnesses the equivalency of $(\hat{F}_i, X, \hat{h}_i)$ and (F_i, X, h_i). Then $\hat{F}$ is the tree with nodes $V(\hat{F}) = V(\hat{F}_1) \cup V(\hat{F}_2)$ isomorphic to F where the isomorphism is witnessed by the bijection $\phi \colon V(\hat{F}) \to V(F)$ defined via

$$\phi(v) = f_i(\hat{\psi}_i(v)) \qquad v \in V(\hat{F}_i),$$

where we use the fact that for any $v \in X$, $\hat{\psi}_i(v) = v$ and $f_i(v) = v$. We further set $\hat{h} = h \circ \phi$. We finally construct T as follows: Take the union of the rooted forests $T_1[V(T_1) \setminus V(\hat{F}_1)]$, $T_2[V(T_2) \setminus V(\hat{F}_2)]$ and $\hat{F}$, then add to the resulting forest the edge sets $\{xy \in E(T_i) \mid x \in V(\hat{F}_i), y \notin V(\hat{F}_i)\}$ for $i \in \{1, 2\}$.

Let us first verify that $(\hat{F}, X, \hat{h})$, and thus by equivalency also (F, X, h), is indeed a restriction of T to X. By construction it is immediately apparent that the iterative deletion of leaves of T not in X indeed yields the tree $\hat{F}$. However, we also need to verify that the height function $\hat{h}$ is correct, i.e., that for all $v \in \hat{F}$, $\hat{h}(v) = \mathrm{height}_T(v)$. We prove the correctness of $\hat{h}$ inductively beginning at the leaves of $\hat{F}$: consider a leaf $v \in \hat{F}$. Since $v \in X$, the join operation calculates h as $h(v) = \max_{i \in \{1,2\}} h_i(v)$ and thus $\hat{h}$ as $\hat{h}(v) = \max_{i \in \{1,2\}} h_i(v)$. By construction, v in T inherits the subtrees of v in T_1 and of v in T_2, thus $\mathrm{height}_T(v) = \max_{i \in \{1,2\}} \mathrm{height}_{T_i}(v) = \hat{h}(v)$. Consider now any internal node $v \in \hat{F}$ with children C in T. For $i \in \{1,2\}$, let C_i be the set of children of v in T_i. By induction hypothesis, for all $w \in C \cap V(\hat{F})$, $\text{height}_T(w) = \hat{h}(w)$. By construction of T and the fact that $\hat{F}$ is a topological generalization of $\hat{F}_1, \hat{F}_2$, it holds that

$$\max_{w \in C \setminus V(\hat{F})} \text{height}_T(w) = \max_{i \in \{1,2\}} \max_{w \in C_i \setminus V(\hat{F}_i)} \text{height}_{T_i}(w). \tag{6.4}$$

By construction of T and the fact that $\hat{F}$ is a topological generalization of $\hat{F}_1$ and $\hat{F}_2$, it holds that every node of $\hat{F}$ has at least all descendants it has in $\hat{F}_1$ and $\hat{F}_2$ and thus

$$\max_{i \in \{1,2\}} \max_{w \in C_i \cap V(\hat{F}_i)} \text{height}_{T_i}(w) \leqslant \max_{w \in C \cap V(\hat{F})} \text{height}_T(w). \tag{6.5}$$

Further note that

$$\begin{aligned} \hat{h}_i(v) - 1 &= \max_{w \in C_i} \text{height}_{T_i}(w) \\ &= \max\{ \max_{w \in C_i \setminus V(\hat{F}_i)} \text{height}_{T_i}(w), \max_{w \in C_i \cap V(\hat{F}_i)} \text{height}_{T_i}(w) \}. \end{aligned} \tag{6.6}$$

Therefore it holds that

$$\begin{aligned} \max_{w \in C} \text{height}_T(w) &= \max\{ \max_{w \in C \setminus V(\hat{F})} \text{height}_T(w), \max_{w \in C \cap V(\hat{F})} \text{height}_T(w) \} \\ &= \max\{ \max_{i \in \{1,2\}} \max_{w \in C_i \setminus V(\hat{F}_i)} \text{height}_{T_i}(w), \max_{w \in C \cap V(\hat{F})} \text{height}_T(w) \} && \text{(by 6.4)} \\ &= \max\{ \max_{i \in \{1,2\}} \{ \max_{w \in C_i \setminus V(\hat{F}_i)} \text{height}_{T_i}(w), \max_{w \in C_i \cap V(\hat{F}_i)} \text{height}_{T_i}(w) \}, \\ &\qquad \max_{w \in C \cap V(\hat{F})} \text{height}_T(w) \} && \text{(by 6.5)} \\ &= \max\{ \max_{i \in \{1,2\}} \hat{h}_i(v) - 1, \max_{w \in C \cap V(\hat{F})} \text{height}_T(w) \} && \text{(by 6.6)} \\ &= \max\{ \max_{i \in \{1,2\}} \hat{h}_i(v) - 1, \max_{w \in C \cap V(\hat{F})} \hat{h}(w) \} && \text{(by induction)} \\ &= \hat{h}(v) - 1 && \text{(by join operation.)} \end{aligned}$$

It remains to show that T is a treedepth decomposition of $G[V(\mathcal{T}_X)]$. Note that $V(T) = V(\mathcal{T}_X)$. Since $\hat{F}$ is a topological generalization of $\hat{F}_i$ for $i \in \{1,2\}$, it holds that $\text{clos}(\hat{F}_i) \subseteq \text{clos}(\hat{F})$ and therefore every edge of $G[V(\mathcal{T}_X)]$ that lives in $V(\hat{F}_i)$ is contained in the closure of T. As $T_i[V(T_i) \setminus V(\hat{F}_i)]$ is by construction a subgraph of T, the edges contained in each $\text{clos}(T_i[V(T_i) \setminus V(\hat{F}_i)])$ are contained in $\text{clos}(T)$. It remains to show that for $i \in \{1,2\}$, every edge xy that has one endpoint $x \in V(\hat{F}_i)$ and the other endpoint $y \in V(T_i) \setminus V(\hat{F}_i)$ will also be covered by the closure of T. Consider the x-y-path in T_i: this path contains a node $z \in \hat{F}_i$ whose successor is not contained in $\hat{F}_i$. Because $\hat{F}$ is a topological generalization of $\hat{F}_i$, the node x is an ancestor of z in $\hat{F}$ and thus in T. Furthermore, by construction of T, the node z is an ancestor of y in T. It follows by transitivity that $xy \in \text{clos}(T)$. Therefore T is a treedepth decomposition of $G[V(\mathcal{T}_X)]$ and the lemma follows for the introduce-case. □

Lemma 12. *Algorithm 1 decides the treedepth of the input graph G'.*

Proof. By Lemma 10 it follows that the set R contains all restrictions of any nice treedepth decomposition rooted at r of the rooted graph G after line 6 of Algorithm 1 is executed. We know that the height of the partial decomposition equals the height of the treedepth decomposition of which it is a restriction. From Lemma 11 we know that every partial decomposition in R is a restriction of a treedepth decomposition of G. From Lemma 3 and Lemma 6 we know that there is a nice treedepth decomposition rooted at r of minimal height of the rooted graph G. From Lemma 3 we know that G has a treedepth decomposition of height $d+1$ if and only if G' has one of height d. Thus the return statement at line 7 will give the correct answer. □

RUNNING TIME OF DYNAMIC PROGRAMMING ALGORITHM

We start by proving an upper bound on the size of the tables.

Lemma 13. *For a set X, the number of possible restrictions on X of height at most d is, up to equivalency, bounded by $2^{|X|d+|X|\log d+|X|\log|X|}$.*

Proof. For any restriction (F, X, h) of height at most d, we have that $|F| \leqslant |X| \cdot d$, since every leaf of F is contained in X and the height of F is $\leqslant d$.

First note that any monotone path P (i.e., a path on which every node is either an ancestor or a descendant of any other node on the path) inside the forest of a restriction can be labeled by h in at most 2^d ways: Since h will increase strictly while following P from top to bottom and $|P| \leqslant d$, the function $h|_P$ is already completely determined by the set $h(P)$.

Consider any ordering $x_1, \dots, x_{|X|}$ of the elements in X and denote by X_i the set $\{x_1, \dots, x_i\}$ for $1 \leqslant i \leqslant |X|$. We upper bound the number of restrictions by considering the following construction: Given a restriction (F, X_i, h_i), we have at most $i \cdot d \cdot 2^d$ ways of constructing a restriction on X_{i+1}: We choose one of $i \cdot d$ nodes of F and attach one of the possible 2^d labeled paths to it, with leaf-node x_{i+1}. We allow adding a path of length zero, such that this operation simply exchanges the initially chosen node with x_{i+1}. Clearly all restrictions on X_{i+1} can be generated in such a way from restrictions on X_i. Thus the number of restrictions is upper bounded by

$$\prod_{i=1}^{|X|} di2^d = 2^{|X|d+|X|\log d}|X|! \leqslant 2^{|X|d+|X|\log d+|X|\log|X|}$$

which is the desired bound. □

Next we upper the number of possible function to consider as a witness for a topological generalization.

Lemma 14. *Given restrictions $(F, X, h), (F', X', h')$ with $X' \subseteq X$ there are at most $2^{d \cdot |X'|/2}$ ways how F can topologically generalize F' and all candidate functions witnessing this fact can be generated in this time.*

Proof. We upper bound the number of possible functions f that witness that F is a topological generalization of F'. Consider a leaf node $v \in F'$, which is necessarily contained in $v \in X' \subseteq X$. Let P'_v be the path from the root of F' to v (in F') and P_v the path from the root of F to v (in F). In order for f to preserve the ancestor relationship of vertices in F', the vertices of P'_v must be mapped to vertices of P_v while preserving order, i.e., if x appears before y in P'_v then $f(x)$ must appear before $f(y)$ in P_v. It follows that there are exactly $\binom{|P_v|}{|P'_v|}$ ways of how f could map P'_v to P_v. We upper bound the number of functions by taking the product of all such paths by $\prod_{v \in X'} \binom{|P_v|}{|P'_v|} \leqslant 2^{d \cdot |X'|/2}$, using the fact that no rooted path in F and F' exceeds length d. This method can be used constructively (since we can check whether a function indeed witnesses a topological generalization in polynomial time) to enumerate all functions. □

Lemma 15. *Algorithm 2 called on G, d, $\mathcal{T}$ and X, where G is a graph rooted at r of size n, $\mathcal{T}$ is a nice tree decomposition of G of width w where every bag contains r and X is a bag of $\mathcal{T}$ runs in time $O(2^{4wd+3w \log wd} \cdot wd \cdot n)$.*

Proof. A nice tree decomposition has $O(n)$ bags (Proposition 4), therefore the linear dependence on n follows easily. By Lemma 13, the set R of restrictions at any given time cannot contain more then $2^{wd+w \log d+w \log w}$ elements. During the join case, we generate all possible restrictions (F, X, h) and for each we consider all pairs $(F_1, X, h_1), (F_2, X, h_2)$ from the respective tables R_1, R_2 of the child bags. For such a pair we need to compute all possible functions f_1, f_2 that might witness that F topologically generalizes both F_1 and F_2. To check if a function witnesses a topological generalization takes linear time in the size of the trees, i.e. $O(d \cdot |X|)$. The total amount of time needed for this operation, using the bound provided by Lemma 14, is at most $(2^{wd+w \log d+w \log w})^3 \cdot (2^{d/2 \cdot w})^2 \cdot O(wd) = O(2^{4wd+3w \log wd} \cdot wd)$. Both forget and introduce operation and checking if the result set already contains an equivalent partial decomposition have running times bounded by this function, thus $O(2^{4wd+3w \log wd} \cdot wd)$ is also an upper bound for the total running time of every operation and the lemma follows. □

We finally are able to sum up the results in the following theorem, a direct consequence of Lemma 19 and Lemma 15. To actually construct a solution, we keep the tables of all bags in memory and employ backtracking to reconstruct a minimal treedepth decomposition.

Theorem 1. *Let G be a graph of size n and d an integer. Given a tree decomposition of G of width w, one can decide in time and space $O(2^{4wd+3w \log wd} \cdot wd \cdot n)$ whether G has treedepth at most d and if so, output a treedepth decomposition of that height.*

7 SIMPLER DYNAMIC PROGRAMMING ALGORITHM

The algorithm we presented in Section 6 is the algorithm that was presented at ICALP 2014. We now present a simpler algorithm with a similar running time that exploits the same basic ideas. We will avoid the use of topological generalizations and a height function to make both the description of the algorithm and the proof of its correctness simpler. The idea here is, whenever we encounter a new node during the dynamic programming, to attempt to introduce it at the depth it would be in a treedepth decomposition for the whole graph. For this we introduce "future" nodes, i.e. nodes we expect to encounter in the part of the graph we have not yet considered. Since every node is immediately introduced at its final depth, it suffices to make sure that no node is too deep. This allows us to avoid the need for a height function. We replace it with a function which tells us if a node in the partial decomposition was already "used" by a node which was then forgotten or is still free for a vertex which will be introduced. More specifically, we replace the height function of partial decompositions by a function $h\colon V(F) \setminus X \to \{\circ, \bullet\}$. We can think of nodes labeled $\bullet$, as nodes that were forgotten and are thus in the "past", and of nodes labeled $\circ$ as nodes we expect to encounter later and are thus in the "future." We need to adapt the definitions for restrictions accordingly.

Definition 30 (Restriction of a partial decomposition)**.** The *restriction of a partial decomposition* (F, X, h) *to* $(\emptyset \neq X' \subseteq X, Y, Z)$, where $Y \cap Z = \emptyset$, is the partial decomposition (F', X', h'), where F' is obtained by iteratively deleting the leaves of the forest F that are *not* in X'. The function h' is defined for any $x \in V(F') \setminus X'$ to be

$$h'(x) = \begin{cases} \bullet & \text{if } x \in Y \\ \circ & \text{if } x \in Z \\ h(x) & \text{otherwise} \end{cases}$$

Accordingly we redefine the restriction of a tree.

Definition 31 (Restriction of a tree)**.** Given a tree T, a set Y and a set Z such that $Y \cap Z = \emptyset$, let $(T, V(T), \emptyset)$ be a partial decomposition, where $\emptyset$ is the null function. A partial decomposition (F', X, h') is a *restriction* of T to (X, Y, Z) if (F', X, h') is equivalent to the restriction (F, X, h) we get from restricting $(T, V(T), \emptyset)$ to (X, Y, Z). We call the function $\psi\colon V(F') \to V(F)$ that witnesses the equivalency as per Definition 23 of these two restrictions the *witness of the restriction*.

The following statement follows from the proof of Lemma 7.

Lemma 16. *Let (F, X, h) be a partial decomposition. For $X' \subseteq X$, and sets Y and Z, where $Y \cap Z = \emptyset$, let (F', X', h') be the restriction of (F, X, h) to (X', Y, Z). Then for any $X'' \subseteq X'$, the restrictions of (F', X', h') and (F, X, h) to (X'', Y, Z) are identical.*

We can redefine the forget, introduce and join operations.

Definition 32 (Forgetting a vertex from a partial decomposition)**.** Let G be a graph, let $X \subseteq V(G)$ and let R' be a set of partial decompositions on the set X. For a vertex $u \in X$, the forget operation on u denoted by *forget*(R', X, u) is defined to be a set A of partial decompositions obtained as follows: Initially set $A \leftarrow \varnothing$; for every partial decomposition $(F', X', h') \in R'$, consider its restriction to $(X \setminus \{u\}, \{u\}, \emptyset)$ and add it to the set A only if it is *not* equivalent to any member in A.

Definition 33 (Vertex introduction into a partial decomposition)**.** Let $G = (V, E, r)$ be a rooted graph, let $X' \subseteq V(G)$ and let R' be a set of partial decompositions of the form (F', X', h'). For a vertex $u \in V(G) \setminus X'$ and an integer d, the introduction operation on u, denoted by *intro*$_d(R', X', u, G)$, is defined to be a set A of partial decompositions constructed as follows: Let $X = X' \cup \{u\}$. Initialize $A \leftarrow \varnothing$. For every $(F', X', h') \in R'$ create new partial decompositions $(F, X' \cup \{u\}, h)$ in the following ways:

- For every node $x \in V(F') \setminus \{X'\}$ where $h'(x) = \circ$ create F by replacing x with u in F' and set $h = h'|_{V(F') \setminus (X' \cup \{u\})}$.
- For every node $x \in X'$ and every $i < d$ create F by adding a path with i nodes to x and u at the end of the path. The function h is an extension of h', such that $h(y) = \circ$ for every node of the added path besides u.

If $E(G[X]) \subseteq E(\mathrm{clos}(F)[X])$ and the depth of F is $\leqslant d$ add $(F, X' \cup \{u\}, h)$ to A.

Definition 34 (Joining Partial Decompositions)**.** Let $G = (V, E, r)$ be a rooted graph. Let R_1 and R_2 be two sets of partial decompositions on $X \subseteq V(G)$. Let d be an integer. Then the join operation *join*$_d$ is defined via *join*$_d(X, R_1, R_2, G) = A$, where A is a set of partial decompositions which is constructed as follows: Initialize $S \leftarrow \varnothing$. Take every pair of partial decompositions $(F_1, X, h_1) \in R_1$ and $(F_2, X, h_2) \in R_2$. If there is an isomorphism ϕ between F_1 and F_2, such that $\phi|_X$ is the identity and for every $x \in V(F_1) \setminus X$ it holds that $h_1(x) = \bullet \Rightarrow h_2(\phi(x)) = \circ$, add the partial decomposition (F_1, X, h) to the set A, where for all $y \in V(F_1) \setminus X$ it holds that $h(y) = \bullet$ if $h_1(y) = \bullet$ or $h_2(\phi(x)) = \bullet$ and $h(y) = \circ$ otherwise.

We can already see that the operations become simpler in this version of the algorithm. We will show now that the proof of its correctness also simplifies. We will prove its correctness via the following two lemmas, which parallel Lemmas 10 and 11. In the following assume that we replaced line 6 in Algorithm 2 by $R := \{(F, \{r\}, \emptyset)\}$.

Lemma 17. *Let Algorithm 2 be called on $(G, d, \mathcal{T}, X)$, where G is a graph rooted at r, the remaining parameters $d, \mathcal{T}, X$ are as described in the algorithm. Then for every nice treedepth* *decomposition T of depth at most d rooted at r of G, the set R returned by the algorithm contains a restriction of T to $(X, V(\mathcal{T}_X), V(G) \setminus V(\mathcal{T}_X))$.*

Proof. We show this by induction, starting at the leaves. Notice that by construction the leaves of $\mathcal{T}$ only contain the root node r. This is the only restriction of any nice treedepth decomposition rooted at r of G.

INTRODUCE CASE If X is an introduce bag whose single child in $\mathcal{T}$ is the bag X', then the if-clause at line 12 is entered. Fix a nice treedepth decomposition T rooted at r of G of depth at most d. We want to show that a partial decomposition (F, X, h) is contained in R, which is a restriction of T to $(X, V(\mathcal{T}_X), V(G) \setminus V(\mathcal{T}_X))$. By induction we assume that the set R' contains a restriction (F', X', h') of T to $(X', V(\mathcal{T}_{X'}), V(G) \setminus V(\mathcal{T}_{X'}))$. Since the introduced node u is not contained in $V(\mathcal{T}_{X'})$ but is contained in X it follows by Lemma 16 that we can compute a restriction equivalent to (F', X', h'), by first taking the restriction (F, X, h) of T to $(X, V(\mathcal{T}_X), V(G) \setminus V(\mathcal{T}_X))$ and then restricting (F, X, h) to $(X', V(\mathcal{T}_{X'}), V(G) \setminus V(\mathcal{T}_{X'}))$. In this last restriction, only two things can happen, depending on u being a leaf or not in F. If u is an internal node in F, its value in h' will be set to $\circ$ and no other changes are made. This change is made backwards on F' by the introduce operation from Definition 33, since it replaces all nodes for which the value of h' is $\circ$ by u. As such, this case is correct. In the other case the node u is a leaf in F and as such u and all its ancestors A that do not have a descendant $x \in X'$ in F are deleted. However long this deleted path is, it is reintroduced by the introduce operation. We just need to show that the value of $h(a) = \circ$ for all $a \in A$. Assume that there is at least one node for which this is not the case and let a be the deepest such node on the path. This implies that $a \in V(\mathcal{T}_{X'}) \setminus X'$. Since the subtree T_a does by construction not contain any nodes of X', but contains at least one of $V(\mathcal{T}_{X'}) \setminus X'$ (namely a) and at least one node of $V(G) \setminus (V(\mathcal{T}_{X'}) \cup X')$ (namely u) and by the properties of tree decompositions X' is a separator between these two sets, it follows that T_a has more than one component and T is not nice. Contradiction.

JOIN CASE Finally, if X is a join bag with two children X_1 and X_2 in $\mathcal{T}$, then the if-clause at line 17 is entered. Let again T be a nice treedepth decomposition rooted at r of the graph G. By inductive hypothesis there exists partial decompositions $(F_1, X, h_1) \in R_1$ and $(F_2, X, h_2) \in R_2$ such that (F_1, X, h_1) and (F_2, X, h_2) are restrictions of T to $(X, V(\mathcal{T}_{X_1}), V(G) \setminus V(\mathcal{T}_{X_1}))$ and $(X, V(\mathcal{T}_{X_2}), V(G) \setminus V(\mathcal{T}_{X_2}))$, respectively. We want to show that a restriction equivalent to the restriction (F, X, h) of the partial decomposition $(T, V(T), \emptyset)$ to $(X, V(\mathcal{T}_X), V(G) \setminus V(\mathcal{T}_X))$ is added to the result set. By induction, there is an isomorphism ϕ between F_1 and F_2, such that $\phi|_X$ is the identity and there is no node $x \in V(F_1) \setminus X$ such that $h_1(x) = \bullet$ and $h_2(\phi(x)) = \bullet$, since that would mean that there is a node of T that is both contained in $V(\mathcal{T}_{X_1}) \setminus X$ and $V(\mathcal{T}_{X_2}) \setminus X$, which is not possible by the properties of tree decompositions. Furthermore, there are functions ϕ_1 and ϕ_2 which witness that F_1 and F_2 are isomorphic to F, respectively, such that both $\phi_1|_X$ and $\phi_2|_X$ are the identity function. This means that $h(y)$ for any $y \in V(F) \setminus X$ must be $\circ$ iff $h_1(\phi_1^{-1}(y)) = \circ$ and $h_2(\phi_2^{-1}(y)) = \circ$, since otherwise $y \notin V(G) \setminus V(\mathcal{T}_X)$. If $h(y) = \bullet$ for any $y \in V(F) \setminus X$ then either $h_1(\phi_1^{-1}(y)) = \bullet$ or $h_2(\phi_2^{-1}(y)) = \bullet$, since $y \in V(\mathcal{T}_X) \setminus X$ and $V(\mathcal{T}_X) \setminus X = (V(\mathcal{T}_{X_1}) \setminus X_1) \cup (V(\mathcal{T}_{X_2}) \setminus X_2)$. It follows that the restriction (F_1, X, h') added to the result set, where h' is the function computed in the join operation, is equivalent to (F, X, h). □

Lemma 18. *Let Algorithm 2 be called on $(G, d, \mathcal{T}, X)$, where G is a graph rooted at r, the remaining parameters $d, \mathcal{T}, X$ are as described in the algorithm. Then every member of R returned by the algorithm is a restriction of a treedepth decomposition of depth $\leqslant d$ of $G[V(\mathcal{T}_X)]$ to (X, Y, Z), for some sets Y and Z, such that $Z \cap V(\mathcal{T}_X) = \emptyset$.*

Proof. We prove this by induction. This statement is clearly correct for the case of leaves, since then h is the null function, and as such it is irrelevant what Y and Z are.

INTRODUCE CASE If X is an introduce bag whose single child in $\mathcal{T}$ is the bag X', then the if-clause at line 12 is entered. By induction every $(F', X, h') \in R'$ is a restriction to (X', Y', Z') of a treedepth decomposition T' of $G[V(\mathcal{T}_{X'})]$ where $Z' \cap V(\mathcal{T}_{X'}) = \emptyset$. Let ϕ' be the function that witnesses this restriction. Consider the case when the introduce operation generates $(F, X' \cup \{u\}, h)$ after replacing a node $x \in V(F') \setminus X'$ for which $h'(x) = \circ$ with the introduced node u. Let T be a treedepth decomposition computed by replacing the node $\phi'(x)$ with u in T'. The partial decomposition $(F, X' \cup \{u\}, h)$ is a restriction of T to $(X, Y', Z = Z' \setminus \{\phi'(x)\})$. Since $Z \subseteq Z'$ it holds that $Z \cap V(\mathcal{T}_X) = \emptyset$. If $(F, X' \cup \{u\}, h)$ is added to the result set, it holds that $E(G[X]) \subseteq E(\text{clos}(F)[X])$. Thus all edges incident to u are covered. Furthermore, since $\phi'(x)$ was not a node in $G[V(\mathcal{T}_{X'})]$, we have not removed any graph edge from the closure of T' and thus T is a treedepth decomposition of $G[V(\mathcal{T}_X)]$ and its height equals the height of T'. Consider now the case where u is added via a path with the node set A. We add the same path to T' to create T. Clearly, the resulting partial decomposition from the introduce operation $(F, X' \cup \{u\}, h)$ in this case is a restriction of T to $(X, Y', Z' \cup \{A\})$ and by construction $(Z' \cup \{A\}) \cap V(\mathcal{T}_X) = \emptyset$. Since we check that all edges of u are covered and that the depth of u is $\leqslant d$ before adding it to the result set, this case is correct.

JOIN CASE Finally, if X is a join bag with two children X_1 and X_2 in $\mathcal{T}$, then the if-clause at line 17 is entered. By inductive hypothesis any partial decompositions $(F_1, X, h_1) \in R_1$ and $(F_2, X, h_2) \in R_2$ are restrictions of treedepth decompositions T_1 of $G[V(\mathcal{T}_{X_1})]$ to (X_1, Y_1, Z_1) and T_2 of $G[V(\mathcal{T}_{X_2})]$ to (X_2, Y_2, Z_2), respectively, such that $Z_1 \cap V(\mathcal{T}_{X_1}) = \emptyset$ and $Z_2 \cap V(\mathcal{T}_{X_2}) = \emptyset$. Let ϕ_1 and ϕ_2 be the witnesses of these restrictions, respectively, and let $\psi\colon V(F_1) \to V(F_2)$ be the function that witnesses the equivalency of (F_1, X, h_1) and (F_2, X, h_2). Assume that the join operation adds a partial decomposition (F, X, h) constructed from (F_1, X, h_1) and (F_2, X, h_2) to the result set.

Let T be the treedepth decomposition achieved by taking T_1, replacing for every node $x \in V(F_1)$ where $h_1(x) = \circ$ and $h_2(\psi(x)) = \bullet$ the node $\phi_1(x)$ with $\phi_2(\psi(x))$, adding the components of $T_2[V(T_2) \setminus \phi_2(V(F_2))]$ and adding to the resulting forest the edge set $\{\phi_1(\psi^{-1}(\phi_2^{-1}(u)))v \mid u \in \phi_2(V(F_2)), v \in V(T_2) \setminus \phi_2(V(F_2)), uv \in E(T_2)\}$. Since in T the set of ancestors for any node of T_1 or T_2 not in $Z_1 \cup Z_2$ is a strict superset to the set of ancestors in T_1 or T_2 it follows that T is a treedepth decomposition of $G[V(\mathcal{T}_X)]$ and its height is by construction the maximum over the heights of T_1 and T_2. Furthermore, by construction (F, X, h) is a restriction of T to $(X, Y_1 \cup Y_2, (Z_1 \cup Z_2) \setminus (Y_1 \cup Y_2))$ and by construction and the induction hypothesis $((Z_1 \cup Z_2) \setminus (Y_1 \cup Y_2)) \cap V(\mathcal{T}_X) = \emptyset$. □

Lemma 19. *Algorithm 1 decides the treedepth of the input graph G'.*

Proof. By Lemma 17 and it follows that the set R contains all restrictions of any nice treedepth decomposition rooted at r of depth $\leqslant d + 1$ of the rooted graph G after line 6 of Algorithm 1 is executed. From Lemma 11 we know that every partial decomposition in R is a restriction of a treedepth decomposition of depth $\leqslant d + 1$ of G. From Lemma 3 and Lemma 6 we know that there is a nice treedepth decomposition rooted at r of minimal height of the rooted graph G. From Lemma 3 we know that G has a treedepth decomposition of height $d + 1$ if and only if G' has one of height d. Thus the return statement at line 7 will give the correct answer. □

Since we changed the definition of restrictions we need to prove an upper bound for the number of possible restrictions on a certain set. This proof follows rather closely the proof of Lemma 13. The upper bound becomes slightly worse. Nevertheless, the final upper bound for the running time will be slightly better.

Lemma 20. *For a set X, the number of possible restrictions of the form (F, X, h) where F has height at most d is, up to equivalency, bounded by $2^{|X|d+2|X|\log d+|X|\log|X|}$.*

Proof. For any restriction (F, X, h) of height at most d, we have that $|F| \leqslant |X| \cdot d$, since every leaf of F is contained in X and the height of F is $\leqslant d$. Consider any ordering $x_1, \ldots, x_{|X|}$ of the elements in X and denote by X_i the set $\{x_1, \ldots, x_i\}$ for $1 \leqslant i \leqslant |X|$. We upper bound the number of trees by considering the following construction: Given a restriction $(F, X_i, \emptyset)$, we have at most $i \cdot d^2$ ways of constructing the tree F' of a restriction of the form $(F', X_{i+1}, \emptyset)$: We choose one of $i \cdot d$ nodes of F and attach one of the possible $d - 1$ paths of length at most $d - 1$ to it, with leaf-node x_{i+1}. We allow adding a path of length zero, such that this operation simply exchanges the initially chosen node with x_{i+1}. These are $i \cdot d^2$ possibilities in total. Clearly all restrictions on X_{i+1} can be generated in such a way from restrictions on X_i. Thus the number of such restrictions is upper bounded by

$$\prod_{i=1}^{|X|} d^2 i = 2^{2|X|\log d}|X|! \leqslant 2^{2|X|\log d+|X|\log|X|}.$$

There are at most $2^{|X|d}$ many h functions for a tree of size $|X| \cdot d$. Thus the desired bound follows. □

Lemma 21. *Algorithm 2 called on G, d, $\mathcal{T}$ and X, where G is a graph rooted at r of size n, $\mathcal{T}$ is a nice tree decomposition of G of width w where every bag contains r and X is a bag of $\mathcal{T}$ runs in time $O(2^{2wd+5w\log wd} \cdot wd \cdot n)$.*

Proof. A nice tree decomposition has $O(n)$ bags (Proposition 4), therefore the linear dependence on n follows easily. Checking if an appropriate isomorphism to perform a join exists, can be done in linear time in the size of trees, since the mapping of the leaves is fixed. To perform a join we to thus check this for every pair of partial decompositions. The total amount of time needed for this is at most $(2^{wd+2w\log d+w\log w})^2 \cdot O(wd) = O(2^{2wd+5w\log wd} \cdot wd)$. Both forget and introduce operation and checking if the result set already contains an equivalent partial decomposition have running times bounded by this function, thus the lemma follows. □

To compute a treedepth decomposition we can perform backtracking. This leads to the following theorem, which is very close to Theorem 1. We point out that the running time given here is slightly better.

Theorem 2. *Let G be a graph of size n and d an integer. Given a tree decomposition of G of width w, one can decide in time and space $O(2^{2wd+5w\log wd} \cdot wd \cdot n)$ whether G has treedepth at most d and if so, output a treedepth decomposition of that height.*

This means, that by introducing future nodes, we achieved an algorithm whose description is simpler, whose correctness is easier to prove and which is faster. This simplification comes from the following insight: In our computation we only need to really check edges between a node and its ancestors in the treedepth decompositions. Since the number of ancestors is always bounded, we can exploit this to keep placeholders for any node that is an ancestor in the final solution. In this way we never have to restructure any treedepth decompositions.

8 SIMPLE ALGORITHM

We can now use Theorem 1 to answer the problem posed by Ossona de Mendez and Nešetřil.

Problem ([193])**.** *Is there a simple linear time algorithm to check* $\mathbf{td}(G) \leqslant d$ *for fixed d? Is there a simple linear time algorithm to compute a rooted forest Y of height d such that* $G \subseteq \operatorname{clos}(Y)$ *(provided that such a rooted forest exists)?*

Treedepth—being a minor-closed property—can be expressed in monadic second order (MSO) logic and thus one can, as mentioned before, employ Courcelle's theorem to compute the treedepth of a graph in linear time. The above problem is motivated by the fact that the running time of this approach is unclear: To use this approach we need a different MSO formula for every d, which depends on the size of the set of forbidden minors for graphs of treedepth d. These families are unknown and it is unclear how to compute them. Thus, to compute the treedepth of a graph exploiting Courcelle's theorem is non-constructive. Furthermore, the size of these forbidden minor families grows at least like a double exponential (and at most like a triple exponential) of d [77] and thus the size of the formula for a specific treedepth grows accordingly. Even worse, the dependency of the running time on the size of the MSO formula of model-checking MSO on graphs of bounded treewidth cannot be bounded by any elementary function unless $\mathrm{P} = \mathrm{NP}$ [94]. Work has been done to simplify the machinery [150, 157], some of it has even been implemented [158, 159], but the tools necessary still remain quite complex. It is thus also not clear how bad the running time dependency on the size of MSO-formula is. We will show that we can use the algorithm presented in Section 6 to give a much more direct and simpler algorithm.

From Proposition 3 we know that a depth first search of a graph with treedepth d gives us a treedepth decomposition of depth at most 2^d. Furthermore, from Proposition 5 we know that from this treedepth decomposition we can easily compute a path decomposition of width $2^d - 1$. We can exploit this to construct an algorithm which only takes a graph as its input, cf., Algorithm 3. The following theorem now follows from Theorem 1 and Algorithm 3.

Theorem 3. *There is a simple algorithm to decide whether the treedepth of a graph is at most d in time and space* $2^{2^{O(d)}} \cdot n$ *and, in the positive case, output a treedepth decomposition witnessing this fact.*

We point out that Algorithm 3 can be made to run in logarithmic space.

Lemma 22. *The algorithm in Algorithm 3 can be made to run in logarithmic space for a fixed treedepth d.*

```
**Input**: A graph $G$, an integer $d$
**Output**: Is the treedepth of $G$ smaller or equal to $d$?
Start computing a tree $Y$ representing a depth first search in $G$;
while Computing $Y$ do
    if depth is greater than $2^d - 1$ then
        return No;
Compute a nice path decomposition $\mathcal{P}$ of $G$ from $Y$;
return treedepth($G, d, \mathcal{P}$);
```

Algorithm 3: treedepth-simple

Proof. A depth first search can be implemented in such a way that only the current path from the root of the search tree to the leaf must be kept in memory. Since the depth of our search is bounded by 2^d keeping such a path in memory requires at most $O(2^d \cdot \log n)$ bits. The bags of the path decomposition on which we want to run Algorithm 1 are by construction precisely these paths in the order they are generated in a depth first search of the graph. The dynamic programming procedure runs over the bags of the decomposition just once. For path decompositions we only need the forget and introduce operations. As such, we only need to keep two bags in memory at any point. If we run the depth first search only until it finds the next leaf, this can be done using logarithmic space. Since furthermore, the size of the tables is bounded by $2^{2^{O(d)}} \cdot \log n$, it follows that the algorithm in Algorithm 3 can be implemented in such a way that it only uses logarithmic space for a fixed depth d. □

In conclusion, we consider this algorithm to solve the problem stated by Ossona de Mendez and Nešetřil.

9 FAST ALGORITHM

The version we presented in the previous section might be simple, but it runs in double exponential time. This is because the way we compute the required tree decomposition for Algorithm 1, which is fast and simple, but we cannot guarantee that its width will be better than $2^d - 1$. If we could bound the width of the tree decomposition we compute before running Algorithm 1 then we could get a much better running time. Remember that $\mathbf{tw}(G) \leqslant \mathbf{pw}(G) \leqslant \mathbf{td}(G) - 1$ for any graph G (Proposition 5). It follows then that if we bound the width of the tree decomposition linearly on the treewidth of the graph we will also be bounding it on the treedepth of the graph. There is an algorithm which runs in time $2^{O(w)} \cdot n$ and calculates a 5-approximation tree decomposition for a graph of treewidth w [38]. The algorithm in Algorithm 4 shows an algorithm that uses these two facts to give a fast algorithm.

```
Input: A graph $G$, an integer $d$
Output: Is the treedepth of $G$ smaller or equal to $d$?
Compute a 5-approximated nice tree decomposition $\mathcal{T}$ of $G$ [38];
if no such tree decomposition is found then
    return No;
return treedepth-on-tree-decomposition($G, d, \mathcal{T}$);
```

Algorithm 4: treedepth-fast

Lemma 23. *Algorithm 4 decides the treedepth of the input graph G.*

Proof. Since $\mathbf{tw}(G) \leqslant \mathbf{td}(G) - 1$ if the graph G has treedepth d then there must exist a tree decomposition of width at most $5d$ which is a 5-approximation for the treewidth of the graph. Thus returning with a negative result on line 4 is correct. From Theorem 1 we know that the call to Algorithm 1 decides if G has treedepth d. □

Lemma 24. *Algorithm 4 runs in time $2^{O(d^2)} \cdot n$ given a graph G and an integer d.*

Proof. Since $\mathbf{tw}(G) \leqslant \mathbf{td}(G) - 1$ it follows that the width of the tree decomposition $\mathcal{T}$ is at most $5d$. The running time of computing the tree decomposition is $2^{O(d)} \cdot n$ and from Theorem 1 the the call to Algorithm 1 is $2^{O(dw)} \cdot n$, where w is the width of $\mathcal{T}$. Since $w \leqslant 5d$ it follows that the running time of the call to Algorithm 1 is $2^{O(d^2)} \cdot n$. □

We arrive at the main theorem of this section.

Theorem 4. *Let G be a graph with n nodes. Deciding if G has a treedepth decomposition of height d and constructing such a treedepth decomposition can be computed in time $2^{O(d^2)} \cdot n$.*

10 TREEDEPTH AND CHORDAL GRAPHS

As mentioned in the introduction, deciding treedepth remains NP-hard even on chordal graphs. Interestingly, the special structure of tree decompositions of chordal graphs can be used to reduce the running time of our algorithm significantly with only minor changes. To the best of our knowledge, no such algorithm was known so far (an algorithm with exponential dependence on the number of cliques in a chordal graph has been proposed before [14]). Since obtaining an optimal tree decomposition for chordal graphs is possible in linear time, we do not need the treewidth approximation here.

Theorem 5. *Given a chordal graph G and an integer d, one can decide in time and space $2^{O(d \log d)} \cdot n$ whether $\mathbf{td}(G) \leqslant d$ and in the positive case output a treedepth decomposition of that height.*

Proof. Since adding a universal vertex to a chordal graph does not violate the chordality, we tacitly assume in the following that such a vertex r exists. First, check whether $\omega(G) > d$ and if that is the case, output that the treedepth of G is greater than d. Otherwise, $\omega(G) \leqslant d$ which implies that $\mathbf{tw}(G) \leqslant d$. It is possible to compute a *clique tree* of G in linear time [29], i.e., an optimal tree decomposition of G in which every bag induces a clique.

If we now run Algorithm 2 on G we can show that only partial decompositions whose forest is a path are kept during each step of the dynamic programming: Consider a bag X and a set of restrictions R computed by the algorithm. For any partial decomposition $(F, X, h) \in R$, the condition $E(G[X]) \subseteq E(\text{clos}(F)[X])$ must be fulfilled (in the join- and introduce-case this is explicitly enforced and it is easy to see that the forget-case cannot create a non-path from a path). Therefore, all elements of X lie in a single path from the root to a leaf in F—but since in a restriction every leaf of F must be a member of X, this path is exactly F. The maximum number of restrictions of height at most d and whose forest is a single path is bounded by $2^{O(d \log d)}$, cf., proof of Lemma 13. If we modify the introduce- and join-procedure of Algorithm 2 to only generate restrictions whose forests are paths, which by the previous observation are the only restrictions that would be kept in any case, the running time reduces to the claimed bound. $\square$

11 CONCLUSION

We provide an explicit simple self contained algorithm, i.e. an algorithm which does not rely on any other complex results, which for a fixed d decides if a graph G has treedepth d and computes a treedepth decomposition of height d if one exists in linear time. This answers an open question posed by Ossona de Mendez and Nešetřil. We also provide an explicit algorithm to decide the treedepth and construct an optimal treedepth decomposition of a given graph in time $2^{O(d^2)}n$.

A natural question that arises is whether one can find a constant-factor approximation for treedepth in single-exponential time, similar to the algorithm for treewidth. Such an algorithm would be interesting in the sense that it would remove the dependency of the algorithm provided in this paper on the treewidth approximation.

Part III

BRANCHING VERSUS DYNAMIC PROGRAMMING

12 BRANCHING, DYNAMIC PROGRAMMING, TREEDEPTH AND TREEWIDTH

As previously discussed, treedepth is algorithmically interesting since it is structurally more restrictive than pathwidth. Remember that there are clear bounds between the treedepth, pathwidth and treewidth of a graph, i.e. by Proposition 5 it holds that $\mathbf{tw}(G) \leqslant \mathbf{pw}(G) \leqslant \mathbf{td}(G) - 1 \leqslant \mathbf{tw}(G) \cdot \log n$. Furthermore, a path decomposition can be easily computed from a treedepth decomposition. Not only are there, as previously mentioned, problems that are $W[1]$-hard or remain NP-hard when parameterized by treewidth or pathwidth, but fpt when parameterized by treedepth [20, 112, 130, 233]; low treedepth can also be exploited to count the number of appearances of different substructures, such as matchings and small subgraphs, much more efficiently [65, 95].

Lokshtanov, Marx and Saurabh showed—assuming SETH—that for 3-Coloring, Vertex Cover and Dominating Set algorithms on a tree decomposition of width w with running time $O(3^w \cdot n)$, $O(2^w \cdot n)$ and $O(3^w w^2 \cdot n)$, respectively, are basically optimal [172]. Their stated intent (as reflected in the title of the paper) was to substantiate the common belief that known DP algorithms that solve these problems were optimal. This is why we feel that a restriction to a certain type of algorithm is not necessarily inferior to a complexity-based approach. Indeed, most algorithms leveraging treewidth *are* dynamic programming algorithms or can be equivalently expressed as such [33, 35, 36, 42, 43, 217]. Even before dynamic programming on tree-decompositions became an important subject in algorithm design, similar concepts were already used implicitly [23, 27]. The sentiment that the table size is the crucial factor in the complexity of dynamic programming algorithms is certainly not new (see e.g. [212]), so it seems natural to provide lower bounds to formalize this intuition. Our tool of choice will be a family of boundaried graphs that are distinct under Myhill–Nerode equivalence. The perspective of viewing graph decompositions as an "algebraic" expression of boundaried graphs that allow such equivalences is well-established [36, 43].

It can be noted that there have been previous formalizations of common algorithmic paradigms in an attempt to investigate what different kinds of algorithms can and cannot achieve, including dynamic programming [7, 50, 118, 137]. This allowed to prove lower bounds for the number of operations required for certain specific problems when a certain algorithmic paradigm was applied. Other research shows that for certain problems such as Steiner Tree and Set Cover an improvement over the "naive" dynamic programming algorithm implies improving exhaustive k-SAT, which would have implications related to SETH [61, 185].

To formalize the notion of a dynamic programming algorithm on tree, path and treedepth decompositions, we consider algorithms that take as input a tree-, path-

or treedepth decomposition of width/depth s and size n and satisfy the following constraints:

1. They pass a single time over the decomposition in a bottom-up fashion;
2. they use $O(f(s) \cdot \log^{O(1)} n)$ space; and
3. they do not modify the decomposition, including re-arranging it.

While these three constraints might look stringent, they include pretty much all dynamic programming algorithm for hard optimization problems on tree or path decompositions. For that reason, we will refer to this type of algorithms simply as *DP algorithms* in the following.

In order to show the aforementioned space lower bounds, we introduce a simple machine model that models DP algorithms on treedepth decompositions and construct superexponentially large *Myhill–Nerode families* that imply lower bounds for DOMINATING SET, VERTEX COVER/INDEPENDENT SET and 3-COLORABILITY in this algorithmic model. These lower bounds hold as well for tree and path decompositions and align nicely with the space complexity of known DP algorithms: for every $\varepsilon > 0$, no DP algorithm on such decomposition of width/depth k can use space bounded by $O\big((3-\varepsilon)^k \cdot \log^{O(1)} n\big)$ for 3-COLORING or DOMINATING SET nor $O\big((2-\varepsilon)^k \cdot \log^{O(1)} n\big)$ for VERTEX COVER/INDEPENDENT SET. While probably not surprising, we consider a formal proof for what previously were just widely held assumptions valuable. The provided framework should easily extend to other problems.

Consequently, any algorithmic benefit of treedepth over pathwidth and treewidth must be obtained by non-DP means. We demonstrate that treedepth allows the design of branching algorithms whose space consumption grows only polynomially in the treedepth and logarithmic in the input size. Such space-efficient algorithms are quite easy to obtain for 3-COLORING and VERTEX COVER/INDEPENDENT SET with running time $O(3^d \cdot n)$ and $O(2^d \cdot n)$, respectively, and space complexity $O(d + \log n)$ and $O(d \cdot \log n)$. Our main contribution on the positive side here are two linear-fpt algorithms for DOMINATING SET which use more involved branching rules on treedepth decompositions. The first one runs in time $d^{O(d^2)} \cdot n$ and uses space $O(d^3 \log d + d \cdot \log n)$. Compared to simple dynamic programming, the space consumption is improved considerably, albeit at the cost of a much higher running time. For this reason, we design a second algorithm that uses a hybrid approach of branching and dynamic programming, resulting in a competitive running time of $O(3^d \log d \cdot n)$ and space consumption $O(2^d d \log d + d \log n)$. Both algorithms are amenable to heuristic improvements (see Section 16 for a discussion).

While applying branching to treedepth seems natural, it is unclear whether it could be applied to treewidth or pathwidth. Recent work by Drucker, Nederlof and Santhanam suggests that, relative to a collapse of the polynomial hierarchy, INDEPENDENT SET restricted to low-pathwidth graphs cannot be solved by a branching algorithm in fpt time [73].

The idea of using treedepth to improve space consumption is not novel. Fürer and Yu demonstrated that it is possible count matchings using polynomial space in the size of the input [95] and a parameter closely related to the treedepth of the input. Their algorithm achieves a small memory footprint by using the algebraization framework developed by Lokshtanov and Nederlof [173]. This technique was also used by Pilipczuk and Wrochna to develop an algorithm for Dominating Set which runs in time $3^d \cdot \text{poly}(n)$ (non-linear) and uses space $O(d \cdot \log n)$ [201]. Based on this last algorithm they showed that computations on treedepth decompositions correspond to a model of non-deterministic machines that work in polynomial time and logarithmic space, with access to an auxiliary stack of maximum height equal to the decomposition's depth.

In our opinion, algorithms based on algebraization have two disadvantages: On the theoretical side, the dependency of the running and space consumption on the input size is often at least $\Omega(n)$. On the practical side, using the *Discrete Fourier Transform* makes it hard to apply common algorithm engineering techniques, like *branch & bound*, which are available for branching algorithms.

13 MYHILL–NERODE FAMILIES

In this section we introduce the basic machinery to formalize the notion of dynamic programming algorithms and how we prove lower bounds based on this notion. To make things easier, we assume that the input graphs are connected, which allows us to presume that the treedepth decomposition is always a tree instead of a forest.

First of all, we need to establish what we mean by *dynamic programming (DP)*. DP algorithms on graph decompositions work by visiting the bags/nodes of the decomposition in a bottom-up fashion (a post-order depth-first traversal), maintaining tables to compute a solution. For decision problems, these algorithms only need to keep at most $\log n$ tables in memory at any given moment (achieved in the case of treewidth by always descending first into the part of the tree decomposition with the greatest number of leaves). We propose a machine model with a read-only tape for the input that can only be traversed once, which only accepts as input decompositions presented in a valid order. This model suffices to capture known dynamic programming algorithms on path, tree and treedepth decompositions. More specifically, given a decision problem on graphs Π and some well-formed instance (G, ξ) of Π (where ξ encodes the non-graph part of the input), let T be a tree, path or treedepth decomposition of G of width/depth k. We fix an encoding $\hat{T}$ of T that lists the separators provided by the decomposition in the order they are normally visited in a dynamic programming algorithm (post-order depth-first traversal of the bag/nodes of a tree/path/treedepth decomposition) and additionally encodes the edges of G contained in a separator using $O(k \log k)$ bits per bag or path. Then $(k, \hat{T}, \xi)$ is a well-formed instance of the *DP decision problem* Π_{DP}. Pairing DP decision problems with the following machine model provides us with a way to model DP computation over graph decompositions.

Definition 35 (Dynamic programming TM)**.** A *DPTM M* is a Turing machine with an input read-only tape, whose head moves only in one direction and a separate working tape. It accepts as inputs only well-formed instances of some DP decision problem.

Any single-pass dynamic programming algorithm that solves a DP decision problem on tree, path or treedepth decompositions of width/depth k using tables of size $f(k)$ that does not re-arrange the decomposition can be translated into a DPTM with a working tape of size $O(f(k) \cdot \log n)$. This model does not suffice to rule out algebraic techniques, since this technique, like branching, requires to visit every part of the decomposition many times [95]; or algorithms that preprocess the decomposition first to find a suitable traversal strategy.

An *s-boundaried graph* $^{\circ}G$ is a graph G with a set $bd(^{\circ}G) \subseteq V(G)$ of s distinguished vertices labeled 1 through s, called the *boundary* of $^{\circ}G$. We will call vertices that are not in $bd(^\circ G)$ *internal*. By $^\circ\mathcal{G}_s$ we denote the class of all s-boundaried graphs. For s-boundaried graphs $^\circ G_1$ and $^\circ G_2$, we let the *gluing* operation $^\circ G_1 \oplus {}^\circ G_2$ denote the s-boundaried graph obtained by first taking the disjoint union of G_1 and G_2 and then unifying the boundary vertices that share the same label.[1]

The following notion of a *Myhill–Nerode family* will provide us with the machinery to prove space lower-bounds for DPTMs where the input instance is an unlabeled graph and hence for common dynamic programming algorithms on such instances. Recall that $^\circ\mathcal{G}_s$ denotes the class of all s-boundaried graphs. The idea is to construct a sufficient number of bounded treedepth instances, such that any two of these instances can be extended such that only one of them is part of the DP-decision problem. Since a correct DPTM has to differentiate all these instances it needs to use at least one bit of memory for every instance in the family, since otherwise there is a pair of instances that could not be differentiated. For technical reasons that will become clear in the proof of Lemma 25 we also have to make sure that all these graphs are small enough.

Definition 36 (Myhill–Nerode family)**.** A set $\mathcal{H} \subseteq {}^\circ\mathcal{G}_s \times \mathbf{N}$ is an *s-Myhill–Nerode family* for a DP-decision problem Π_{DP} if the following holds:

1. For every $(^\circ H, q) \in \mathcal{H}$ it holds that $|^\circ H| = |\mathcal{H}| \cdot \log^{O(1)}|\mathcal{H}|$ and $q = 2^{|\mathcal{H}|\cdot\log^{O(1)}|\mathcal{H}|}$.
2. For every subset $\mathcal{I} \subseteq \mathcal{H}$ there exists an s-boundaried graph $^\circ G_{\mathcal{I}} \in {}^\circ\mathcal{G}_s$ with $|^\circ G_{\mathcal{I}}| = |\mathcal{H}| \cdot \log^{O(1)}|\mathcal{H}|$ and an integer $p_{\mathcal{I}}$ such that for every $(^\circ H, q) \in \mathcal{H}$ it holds that
$$(^\circ G_{\mathcal{I}} \oplus {}^\circ H, p_{\mathcal{I}} + q) \notin \Pi_{\mathrm{DP}} \iff (^\circ H, q) \in \mathcal{I}.$$

Let $^\circ\mathbf{td}(^\circ G)$ be the minimal depth over all treedepth decompositions of $^\circ G \in \mathcal{G}_s$ where the boundary appears as a path starting at the root. We define the *size* of a Myhill–Nerode family $\mathcal{H}$ as $|\mathcal{H}|$, its *treedepth* as

$$\mathbf{td}(\mathcal{H}) = \max_{(^\circ H,\cdot)\in\mathcal{H},\mathcal{I}\subseteq\mathcal{H}} {}^\circ\mathbf{td}(^\circ G_{\mathcal{I}} \oplus {}^\circ H)$$

and its *treewidth* and *pathwidth* as the maximum tree/path decomposition of lowest width of any $(^\circ H, \cdot) \in \mathcal{H}$ where the boundary is contained in every bag.

The following lemma still holds if we replace "treedepth" by "pathwidth" or "treewidth".

Lemma 25. *Let $\varepsilon > 0, c > 1$ and Π be a DP decision problem such that for every s there exists an s-Myhill–Nerode family $\mathcal{H}$ for Π of size $c^s/f(s)$ where $f(s) = s^{O(1)} \cap \Omega(1)$ and depth $\mathbf{td}(\mathcal{H}) = s + o(s)$. Then no DPTM can decide Π using space $O((c-\varepsilon)^k \cdot \log^{O(1)} n)$, where n is the size of the input instance and k the depth of the treedepth decomposition given as input.*

1 In the literature the result of gluing is often an unboundaried graph. Our definition of gluing will be more convenient in the following.

Proof. Assume to the contrary that such a DPTM M exists. Fix s and consider any subset $\mathcal{I} \subseteq \mathcal{H}$ of the s-Myhill–Nerode family $\mathcal{H}$ of Π. By definition, all graphs in $\mathcal{H}$ and the graph ${}^{\circ}G_{\mathcal{I}}$ have size at most

$$|\mathcal{H}| \cdot \log^{O(1)}|\mathcal{H}| = c^s \cdot s^{O(1)}.$$

By definition, for every s-boundaried graph ${}^{\circ}H$ contained in $\mathcal{H}$, there exist treedepth decompositions for ${}^{\circ}G_{\mathcal{I}} \oplus {}^{\circ}H$ of depth at most $s + o(s)$ such that the boundary vertices of ${}^{\circ}G_{\mathcal{I}}$ appear on a path of length s starting at the root of the decomposition. Hence, we can fix a treedepth decomposition $T_{\mathcal{I}}$ of $G_{\mathcal{I}}$ with exactly these properties and choose a treedepth decomposition T of ${}^{\circ}G_{\mathcal{I}} \oplus {}^{\circ}H$ such that $T_{\mathcal{I}}$ is a subgraph. Moreover, we choose an encoding of T that lists the separators of $T_{\mathcal{I}}$ first.

Notice that M only uses $(c-\varepsilon)^{s+o(s)} \cdot s^{O(1)}$ space. There are $2^{|\mathcal{H}|} = 2^{c^s/f(s)}$ choices for $\mathcal{I}$. For there to be a different content on the working tape of M for every choice of $\mathcal{I}$ we need at least $c^s/f(s)$ bits. We rewrite this as $(c-\varepsilon)^s \cdot \alpha^s/f(s)$, where $\alpha = c/(c-\varepsilon)$. Since $\alpha > 1$ it follows that $\alpha^s/f(s)$ grows exponentially faster than $(c-\varepsilon)^{o(s)} \cdot s^{O(1)}$ and thus $c^s/f(s) \in \omega((c-\varepsilon)^{s+o(s)} \cdot s^{O(1)})$. By pigeonhole principle it follows that there exist graphs ${}^{\circ}G_{\mathcal{I}}$, ${}^{\circ}G_{\mathcal{J}}$ for sets $\mathcal{I} \neq \mathcal{J} \subseteq \mathcal{H}$ for sufficiently large s for which M is in the same state and has the same working tape content after reading the separators of the respective decompositions $T_{\mathcal{I}}$ and $T_{\mathcal{J}}$. Choose $({}^{\circ}H, q) \in \mathcal{I} \mathbin{\triangle} \mathcal{J}$. By definition

$$({}^{\circ}G_{\mathcal{I}} \oplus {}^{\circ}H, p_{\mathcal{I}} + q) \notin \Pi \iff ({}^{\circ}G_{\mathcal{J}} \oplus {}^{\circ}H, p_{\mathcal{J}} + q) \in \Pi$$

but M will either reject or accept both inputs. Contradiction. $\square$

14 SPACE LOWER BOUNDS FOR DYNAMIC PROGRAMMING

In this section we prove space lower bounds for dynamic programming algorithms as defined in Section 12 for 3-Coloring, Vertex Cover and Dominating Set. These space lower bounds all follow the same basic construction. We define a problem-specific "state" for the vertices of a boundary set X and construct two boundaried graphs for it: one graph that enforces this state in any (optimal) solution of the respective problem and one graph that "tests" for this state by either rendering the instance unsolvable or increasing the costs of an optimal solution. We begin by proving a lower bound for 3-Coloring.

Theorem 6. *For every $\varepsilon > 0$, no DPTM solves 3-Coloring on a tree, path or treedepth decomposition of width/depth k with space bounded by $O((3-\varepsilon)^k \cdot \log^{O(1)} n)$.*

Proof. For any s we construct an s-Myhill–Nerode family $\mathcal{H}$. Let X be the s vertices in the boundary of all the boundaried graphs in the following. Then for every three-partition $\mathcal{X} = \{R, G, B\}$ of X we add a boundaried graph $^\circ H_\mathcal{X}$ to the family $\mathcal{H}$ by taking a single triangle v_R, v_G, v_B and connecting the vertices v_C to all vertices in $X \setminus C$ for $C \in \{R, G, B\}$. Notice that any 3-coloring of $^\circ H_\mathcal{X}$ induces the partition $\mathcal{X}$ on the nodes X. Since instances of three-coloring do not need any additional parameter, we ignore this part of the construction of $\mathcal{H}$ and implicitly assume that every graph in $\mathcal{H}$ is paired with zero.

To construct the graphs $G_\mathcal{I}$ for $\mathcal{I} \subset \mathcal{H}$, we will employ the *circuit gadget* v_1, v_2, u highlighted in Figure 14.1. Note that if v_1, v_2 receive the same color, then u must be necessarily colored the same. In every other case, the color of u is arbitrary. Now for every three-partition $\mathcal{X} = \{R, G, B\}$ of X we construct a *testing gadget* $^\circ\mathrm{T}_\mathcal{X}$ as follows: For every $C \in \{R, G, B\}$ we arbitrarily pair the vertices in C and connect them via the circuit gadget (as v_1, v_2). If $|C|$ is odd, we pair some vertex of C with itself. We

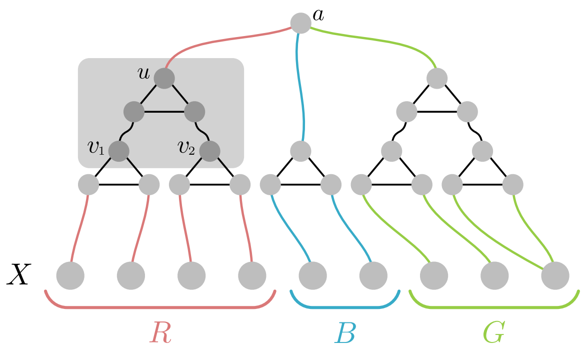

Figure 14.1: The gadget $^\circ\mathrm{T}_\mathcal{X}$ for $\mathcal{X} = \{R, G, B\}$.

then repeat the construction with all the u-vertices of those gadgets, resulting in a hierarchical structure of depth $\sim \log |B_i|$ (cf., Figure 14.1 for an example construction). Finally, we add a single vertex a and connect it to the top vertex of the three circuits. Note that by the properties of the circuit gadget, the graph $^\circ\mathrm{T}_{\mathcal{X}}$ is three-colorable iff the coloring of X does *not* induce the partition $\mathcal{X}$. In particular, the graph $^\circ\mathrm{T}_{\mathcal{X}} \oplus {}^\circ H_{\mathcal{X}'}$ is three-colorable iff $\mathcal{X} \neq \mathcal{X}'$.

Now for every subset $\mathcal{I} \subseteq \mathcal{H}$ of graphs from the family, we define the graph $^\circ G_{\mathcal{I}} = \bigoplus_{^\circ H_{\mathcal{X}} \in \mathcal{I}} {}^\circ\mathrm{T}_{\mathcal{X}}$. By our previous observation, it follows that for every $^\circ H_{\mathcal{X}} \in \mathcal{H}$ the graph $^\circ G_{\mathcal{I}} \oplus {}^\circ H_{\mathcal{X}}$ is three-colorable iff $^\circ H_{\mathcal{X}} \notin \mathcal{I}$. Furthermore, every composite graph has treedepth at most $s + 3\lceil \log s \rceil + 1$ as witnessed by a decomposition whose top s vertices are the boundary X and the rest has the structure of the graph itself after every triangle is made into a path. The graphs $^\circ G_{\mathcal{I}}$ for every $\mathcal{I} \subseteq \mathcal{H}$ have size at most $3^s \cdot 6s$. We conclude that $\mathcal{H}$ is an s-Myhill–Nerode family of size $3^s/6$ (the factor $1/6$ accounts for the 3! permutations of the partitions) and the claim follows from Lemma 25. □

Surprisingly, the construction to prove a lower bound for VERTEX COVER is very similar to the one for 3-COLORING.

Theorem 7. *For every $\varepsilon > 0$, no DPTM solves* VERTEX COVER *on a tree, path or treedepth decomposition of width/depth k with space bounded by $O((2-\varepsilon)^k \cdot \log^{O(1)} n)$.*

Proof. For every s we construct an s-Myhill–Nerode family $\mathcal{H}$. Let X be the s vertices in the boundary of all the boundaried graphs in the following. Assume for now that s is even. For every subset $A \subseteq X$ such that $|A| = |X|/2$ we construct a graph $^\circ H_A$ which consists of the boundary as an independent set and a matching to A and add $(^\circ H_A, s/2)$ to $\mathcal{H}$. Note that any optimal vertex cover of any $^\circ H_A$ has size $s/2$ and that A is such a vertex cover.

Consider $\mathcal{I} \subseteq \mathcal{H}$. We will again use the circuit gadget highlighted in Figure 14.1 to construct $^\circ G_{\mathcal{I}}$. Note that if either v_1 or v_2 is in the vertex cover we can cover the rest of gadget with two vertices, one of them being the top vertex u. Otherwise, u cannot be included in a vertex cover of size two. We still need two vertices, even if both v_1 and v_2 are already in the vertex cover. For a set $A \subseteq X$ such that $|A| = |X|/2$ we construct the testing gadget $^\circ\mathrm{T}_A$ by starting with the boundary X as an independent set, connecting the vertices of $X \setminus A$ pairwise via the circuit gadget (using an arbitrary pairing and potentially pairing a leftover vertex with itself). As in the proof of Theorem 6, we repeat this construction on the respective u-vertices of the circuits just added and iterate until we have added a single circuit at the very top. Let us denote the topmost u-vertex in this construction by u'. Let λ be the number of circuits added in this fashion. Any optimal vertex of $^\circ\mathrm{T}_A$ has size 2λ and does not include u'. Note that if a node of $X \setminus A$ is in the vertex cover, we can cover the rest of the gadget with 2λ many vertices, such that u' is part of the vertex cover.

We construct $^\circ G_{\mathcal{I}}$ by taking $\oplus_{^\circ H_A \in \mathcal{I}} {}^\circ\mathrm{T}_A$ and adding a single vertex a that connects to all u'-vertices of the gadgets $\{^\circ\mathrm{T}_A\}_{^\circ H_A \in \mathcal{I}}$. Notice that, by the same reason u' was not part of an optimal vertex cover of any gadget $^\circ T_A$, the node a must be part of any optimal vertex cover of $^\circ G_{\mathcal{I}}$ for $|\mathcal{I}| > 1$. For $|\mathcal{I}| = 1$ either the only u' or a must be contained besides the other vertices, but we will assume w.l.o.g. that it is a. Let ℓ be the biggest optimal vertex cover for any such $^\circ G_{\mathcal{I}}$. Let $\ell_{\mathcal{I}}$ be the size of an optimal vertex cover for a specific $^\circ G_{\mathcal{I}}$. For simplicity, we pad $^\circ G_{\mathcal{I}}$ with $\ell - \ell_{\mathcal{I}}$ isolated K_2 subgraphs to ensure that the size of an optimal vertex cover is ℓ.

We claim that $^\circ G_{\mathcal{I}} \oplus {}^\circ H_A$ has a vertex cover of size $\ell + s/2 - 1$ iff $^\circ H_A \notin \mathcal{I}$. If $H_A \notin \mathcal{I}$, then for every gadget $^\circ T_{A'}$ that comprises $^\circ G_{\mathcal{I}}$ it holds that $A' \neq A$. Since $|A'| = |A|$ it follows that $(X \setminus A') \cap A \neq \emptyset$. Since $^\circ G_{\mathcal{I}} \oplus {}^\circ H_A$ has $s/2$ vertices of degree one whose neighborhood is A, we can assume that an optimal vertex cover contains A. From the previous arguments about the possible vertex covers for the $^\circ T_{A'}$ gadgets it follows that the solution still needs two nodes for every circuit gadget of $^\circ T_{A'}$, but now this part of the vertex cover can include u'. Since this is true for every $^\circ T_{A'}$ it follows that a does not need to be part of the vertex cover. Thus the size of an optimal vertex cover is precisely $\ell + s/2 - 1$. If $^\circ H_A \in \mathcal{I}$ then the gadget $^\circ T_A$ that has no vertex cover using two nodes per circuit gadget that contains its node u'. It follows that any optimal vertex cover of $^\circ G_{\mathcal{I}} \oplus {}^\circ H_A$ must contain either a or its u'. Thus the size of the solution is at least $\ell + s/2$. We thus set $p_{\mathcal{I}}$ to be $\ell - 1$.

The size of the family $\mathcal{H}$ is, using Stirling's approximation, bounded from below by

$$\binom{s}{s/2} \geqslant \frac{2^{s-1}}{\sqrt{s/2}}$$

and is smaller than 2^s. All numbers involved describe subsets of graphs and thus must be smaller than the sizes of those graphs. All graphs in the family have size s. The graphs $^\circ G_{\mathcal{I}}$ as described, are constructed by adding a polylogarithmic number of nodes to the boundary per gadget $^\circ T_A$ and thus their size is bounded by $|\mathcal{H}| \cdot \log^{O(1)}|\mathcal{H}|$. For uneven s we take the next smaller even s' and use the s'-family as the s-family. The treedepth is $\mathbf{td}(\mathcal{H}) = s + o(s)$ by the same argument as for the construction for Theorem 6. We conclude that $\mathcal{H}$ is a s-Myhill–Nerode family of size $2^s/f(s)$ for $f(s) = s^{O(1)} \cap \Omega(1)$ and depth $\mathbf{td}(\mathcal{H}) = s + o(s)$ and thus by Lemma 25 the theorem follows. □

Theorem 8. *For every $\varepsilon > 0$, no DPTM solves* Dominating Set *on a tree, path or treedepth decomposition of width/depth k with space bounded by $O\big((3-\varepsilon)^k \cdot \log^{O(1)} n\big)$.*

Proof. For any s divisible by three we construct an s-Myhill–Nerode family $\mathcal{H}$ as follows. Let X be the s boundary vertices of all the boundaried graphs in the following. Then for every three-partition $\mathcal{X} = (B, D, W)$ of X into sets of size $s/3$, we construct a graph $H_{\mathcal{X}}$ by connecting two pendant[1] vertices to every vertex in B, connecting every vertex in D to a vertex which itself is connected to two pendant vertices and leaving

1 A pendant vertex is a node of degree one.

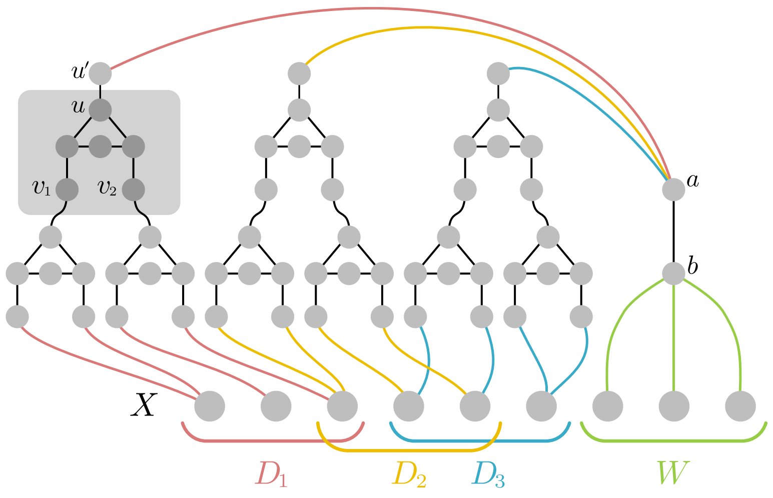

Figure 14.2: The gadget $^{\circ}\mathrm{T}_W$ for $\mathcal{D}_W = \{D_1, D_2, D_3\}$. Padding-vertices are not included.

W untouched. Intuitively, we want the vertices of B to be in any minimal dominating set, the vertices in D to be dominated from a vertex in $H_{\mathcal{X}}$ not in the boundary and the vertices in W to be dominated from elsewhere. We add every pair $(H_{\mathcal{X}}, 2s/3)$ to $\mathcal{H}$. Notice that the size of an optimal dominating set of $H_{\mathcal{X}}[B \cup D]$ is $2s/3$ and there is only one such optimal dominating set, namely $B \cup N(D)$.

For a subset $\mathcal{I} \subseteq \mathcal{H}$ let $\mathcal{D}_W = \{D \mid H_{(X \setminus (D \cup W)), D, W)} \in \mathcal{I}\}$ be a set defined for every $W \subset X$. We construct the graph $^{\circ}G_{\mathcal{I}}$ using the *circuit gadget* with nodes v_1, v_2, u highlighted in Figure 14.2: Note that if v_1, v_2 need to be dominated, then there is no dominating set of the gadget of size two that contains u. If one of v_1, v_2 does not need to be dominated (but is not in the dominating set) then a dominating set of size two of the circuit gadget containing u exists. For every $W \subset X$ with $|W| = s/3$ construct a *testing gadget* $^{\circ}\mathrm{T}_W$ as follows. Assume first that $\mathcal{D}_W$ is non-empty. For every set $D \in \mathcal{D}_W$ we construct the gadget $^{\circ}\Lambda_D$ by arbitrarily pairing the vertices in D and connecting them via the circuit gadget as exemplified in Figure 14.2. This closely parallels the constructions we have seen in the proofs for Theorem 6 and 7: If $|D|$ is odd, we pair some vertex of D with itself. We then repeat the construction with all the u-vertices of those gadgets, resulting in a hierarchical structure of depth $\sim \log |D|$. To finalize the construction of $^{\circ}\Lambda_D$ we take the u-vertex of the last layer and connect it to a new vertex u'. This concludes the construction of $^{\circ}\Lambda_D$. Let in the following λ be the number of circuits we used to construct such a $^{\circ}\Lambda_D$ gadget (this quantity only depends on s and is the same for any $^{\circ}\Lambda_D$). If $\mathcal{D}_W$ is empty, then $^{\circ}\mathrm{T}_W$ is the boundary and a K_2 with one of its vertices adjacent to all vertices in W plus $\binom{2s/3}{s/3} 2\lambda$ isolated padding-vertices. Otherwise we obtain $^{\circ}\mathrm{T}_W$ by taking the graph $\bigoplus_{D \in \mathcal{D}} {^{\circ}\Lambda_D}$ and adding two additional vertices a, b as well as $(\binom{2s/3}{s/3} - |\mathcal{D}_W|) 2\lambda$ isolated vertices for padding. The vertex a is adjacent to all u' vertices of all the gadgets $\{^{\circ}\Lambda_D\}_{D \in \mathcal{D}_W}$ and the vertex b

is adjacent to $\{a\} \cup W$ (cf., again Figure 14.2 for an example). Finally we define for every $\mathcal{I} \subseteq \mathcal{H}$ the graph ${}^{\circ}G_{\mathcal{I}} = \bigoplus_{W \subset X, |W| = s/3} {}^{\circ}\mathrm{T}_W$.

Let $\alpha = \binom{2s/3}{s/3} 2\lambda + 1$. Consider some ${}^{\circ}\mathrm{T}_W$, for a W such that $\mathcal{D}_W \neq \varnothing$. Assume we start with a dominating set S such that $S \cap D = \emptyset$ for at least one $D \in D_W$. We want to show that extending S to dominate $V(({}^{\circ}\mathrm{T}_W) \setminus X) \cup W$ requires at least $\alpha + 1$ many vertices. We can assume that b must be added to the dominating set. All $(\binom{2s/3}{s/3} - |\mathcal{D}_W|)2\lambda$ padding vertices must also be added. Since we need at least two vertices per circuit gadget, at least 2λ vertices will always be necessary to dominate each ${}^{\circ}\Lambda_D$ subgraph of ${}^{\circ}\mathrm{T}_W$ (of which there are $|\mathcal{D}_W|$ many). For ${}^{\circ}\Lambda_D$ where $S \cap D = \emptyset$ no dominating set of the circuit gadgets of size 2λ can also dominate u'. Thus we also need to take a or u' into the dominating set and we need at least $\alpha + 1$ many vertices.

Now assume that we start with a dominating set S that contains at least one node of every $D \in \mathcal{D}_W \neq \varnothing$. In this case we can dominate all the circuit gadgets and u' with 2λ many nodes in every ${}^{\circ}\Lambda_D$. Thus, there is a set in ${}^{\circ}\mathrm{T}_W$ that dominates $V(({}^{\circ}\mathrm{T}_W) \setminus X) \cup W$ of size α, since neither a nor any u' needs to be in the dominating set.

Let us now show that our boundaried graphs work as intended and calculate the appropriate parameters $p_{\mathcal{I}}$. Consider any graph ${}^{\circ}H_{(B_0, D_0, W_0)} \in \mathcal{H}$ and the graph ${}^{\circ}G_{\mathcal{I}}$ for any $\mathcal{I} \subseteq \mathcal{H}$. We show that ${}^{\circ}H_{(B_0, D_0, W_0)} \oplus G_{\mathcal{I}}$ has an optimal dominating set of size at most $\binom{s}{s/3}\alpha + 2s/3$ iff ${}^{\circ}H_{(B_0, D_0, W_0)} \notin \mathcal{I}$. We need to include the $s/3$ vertices of B_0 and the $s/3$ vertices of $N(D) \cap V({}^{\circ}H_{(B_0, D_0, W_0)})$. We use the sets $\mathcal{D}_W$ as defined previously. First, assume that $\mathcal{D}_{W_0} = \varnothing$, that is, for every set B', D' we have that $H_{(B', D', W_0)} \notin \mathcal{I}$ and in particular $H_{(B_0, D_0, W_0)} \notin \mathcal{I}$. It is easy to see that the simple version of the gadget ${}^{\circ}\mathrm{T}_{W_0}$ for the case where $\mathcal{D}_{W_0} = \varnothing$ can dominate its non-boundary nodes and W_0 with α nodes. All other gadgets ${}^{\circ}\mathrm{T}_{W'}$ for $W' \neq W_0$ do not need to dominate their respective W'-sets and can therefore include their a-vertices and not include their b-vertices. Accordingly, they can all dominate their internal vertices with α many nodes. This all adds up to a dominating set of size $\binom{s}{s/3}\alpha + 2s/3$. Next, assume that $\mathcal{D}_{W_0} \neq \varnothing$ and $D_0 \notin \mathcal{D}_{W_0}$, i.e., again $H_{(B_0, D_0, W_0)} \notin \mathcal{I}$. Therefore, for every set $D' \in D_{W_0}$ we have that $D' \cap B_0 \neq \varnothing$. Since we can assume B_0 is part of our dominating set, we only need to add α vertices to the dominating from ${}^{\circ}\mathrm{T}_{W_0}$ to dominate $V(({}^{\circ}\mathrm{T}_{W_0}) \setminus X) \cup W_0$. All gadgets ${}^{\circ}\mathrm{T}_{W'}, W' \neq W_0$ also need α vertices, as observed above. We obtain in total a dominating set of size $\binom{s}{s/3}\alpha + 2s/3$. Finally, consider the case that $D_0 \in \mathcal{D}_{W_0}$, i.e. $H_{(B_0, D_0, W_0)} \in \mathcal{I}$. Since $B_0 \cap D_0 = \emptyset$ the gadget ${}^{\circ}\Lambda_{D_0}$ needs $\alpha + 1$ vertices to dominate $V(({}^{\circ}\mathrm{T}_{W_0}) \setminus X) \cup W_0$. Dominating W_0 with nodes different from the b-vertex of ${}^{\circ}\mathrm{T}_{W_0}$ does not help. Thus we need at least $\binom{s}{s/3}\alpha + 2s/3 + 1$ vertices to dominate ${}^{\circ}H_{(B_0, D_0, W_0)} \oplus G_{\mathcal{I}}$.

Choosing $p_{\mathcal{I}} = \binom{s}{s/3}\alpha$ completes the construction of $({}^{\circ}G_{\mathcal{I}}, p_{\mathcal{I}})$. The size of all these graphs is bounded by $O(\binom{s}{s/3}\binom{2s/3}{s/3} \log s) = O(3^s \log s)$. We conclude that $\mathcal{H}$ is an s-Myhill–Nerode family of size $\binom{s}{s/3}\binom{2s/3}{s/3}$ which is $\Omega(3^s/s)$ and $O(3^s)$. For s indivisible by three we take the next smaller integer s' divisible by three and use s'-family as the s-family. It is easy to confirm that the treedepth of $\mathcal{H}$ is $\mathbf{td}(\mathcal{H}) = s + o(s)$ and the theorem follows from Lemma 25. □

15 DOMINATING SET WITH $O(d^3 \log d + d \log n)$ SPACE

That branching might be a viable algorithmic design strategy for low-treedepth graphs can easily be demonstrated for problems like 3-Coloring and Vertex Cover: We simply branch on the topmost vertex of the decomposition and recurse into (annotated) subinstances. For q-Coloring, this leads to an algorithm with running time $O(q^d \cdot n)$ and space complexity $O(d \log n)$. Since it is possible to perform a depth-first traversal of a given tree using only $O(\log n)$ space [165], the space consumption of this algorithm can be easily improved to $O(d + \log n)$. Similarly, branching solves Vertex Cover in time $O(2^d \cdot n)$ and space $O(d \log n)$.

The task of designing a similar algorithm for Dominating Set is much more involved. Imagine branching on the topmost vertex of the decomposition: while the branch that includes the vertex into the dominating set produces a straightforward recurrence into annotated instances, the branch that excludes it from the dominating set needs to decide *how* that vertex should be dominated. The algorithm we present here proceeds as follows: We first guess whether the current node x is in the dominating set or not. Recall that P_x denotes the nodes of the decomposition that lie on the unique path from x to the root of the decomposition (and $x \notin P_x$). We iterate over every possible partition $S_1 \uplus \cdots \uplus S_\ell = P_x \cup \{x\}$ into $\ell \leqslant d$ sets of $P_x \cup \{x\}$. The semantic of a block S_i is that we want every element S_i to be dominated exclusively by nodes from a specific subtree of x. A recursive call on a child y of x, together with an element of the partition S_i, will return the size of a dominating set which dominates $V(T_y) \cup S_i$. The remaining issue is how these specific solutions for the subtrees of x can be combined into a solution in a space-efficient manner. To that end, we first compute the size of a dominating set for T_y itself and use this as baseline cost for a subtree T_y. For a block S_i of a partition of P_x, we can now compare the cost of dominating $V(T_y) \cup S_i$ against this baseline to obtain overhead cost of dominating S_i using vertices from T_y. Collecting these overhead costs in a table for subtrees of x and the current partition, we are able to apply certain reduction rules on these tables to reduce their size to at most d^2 entries. Recursively choosing the best partition then yields the solution size using only polynomial space in d and logarithmic in n. Formally, we prove the following:

Theorem 9. *Given a graph G and a treedepth decomposition T of G, Algorithm 5 finds the size of a minimum dominating set of G in time $d^{O(d^2)} \cdot n$ using $O(d^3 \log d + d \log n)$ bits.*

We split the proof of Theorem 9 into lemmas for correctness, running time and used space.

Lemma 26. *Algorithm 5 called on a graph G, a treedepth decomposition T of G, the root r of T and $P = D = \varnothing$ returns the size of a minimum dominating set of G.*

Proof. If we look at a minimal dominating set S of G we can charge every node in $V(G) \setminus S$ to a node from S that dominates it. We are thus allowed to treat any node in G as if it was dominated by a single node of S. We will prove this lemma by induction, the inductive hypothesis being that a call on a node x with arguments $D = S \cap P_x$ and $P \subseteq P_x$ being the set of nodes dominated from nodes in T_x by S returns $|S \cap V(T_x)|$.

It is clear that the algorithm will call itself until a leaf is reached. Let x be a leaf of T on which the function was called. We first check the condition at line 2, which is true if either x is not dominated by a node in D or if some node in P is not yet dominated. In this case we have no choice but to add x to the dominating set. Three things can happen: P is not fully dominated, which means that it was not possible under these conditions to dominate P, in which case we correctly return ∞, signifying that there is no valid solution. Otherwise we can assume P is dominated and we return 1 if we had to take x and 0 if we did not need to do so. Thus the leaf case is correct.

We assume now x is not a leaf and thus we reach line 6. We first add x to P, since it can only be dominated either from a node in D or a node in T_x. Nodes in T_x can only be dominated by nodes from $V(T_x) \cup P_x$. We assume by induction that $D = S \cap P_x$ and that P only contains nodes which are either in S or dominated from nodes in T_x. Algorithm 5 executes the same computations for D and $D \cup \{x\}$, representing not taking and taking x into the dominating set respectively. We must show that the set P for the recursive calls is correct. There exists a partition of the nodes of P not dominated by D (respectively $D \cup \{x\}$) such that the nodes of every element of the partition are dominated from a single subtree T_y where y is a child of x. The algorithm will eventually find this partition on line 8. The baseline value, i.e. the size of a dominating set of T_y given that the nodes in D (respectively $D \cup \{x\}$) are in the dominating set, gives a lower bound for any solution. In the lists in L and L' we keep the extra cost incurred by a subtree T_y if it has to dominate an element of the partition. We only need to keep the best d values for every S_i: Assume that it is optimal to dominate S_i from T_y and there are $d+1$ subtrees induced on children $y' \neq y$ of x whose extra cost over the baseline to dominate S_i is strictly smaller than the extra cost for T_y. At least one of these subtrees $T_{y'}$ is not being used to dominate an element of the partition. This means we could improve the solution by letting T_y dominate itself and taking the solution of $T_{y'}$ that also dominates S_i. Keeping d values for every element in the partition suffices to find a minimal solution, which is what *find_min_solution*(L) does as follows: Create a bipartite graph $G = (A \uplus B, E)$ such that A contains a node for every S_i and B contains a node for every y for which there is an entry $(\cdot, y)$ in L. For every node a representing S_i we add an edge with weight $d - c$ to a node b representing y if $(c, y) \in L[i]$. Notice that the minimal number of nodes above the baseline needed to dominate an element of the partition is always less than d. A maximal matching in this bipartite graph tells us how many nodes above the baseline are required to dominate the elements of the partition from subtrees rooted at children of x. Since L contains at most d^2 entries this can be computed in polynomial time in d.

Since with lines 23 and 24 we take the minimum over all possible partitions and taking x into the dominating set or not, we get that by inductive assumption the algorithm returns the correct value. The lemma follows since the first call to the algorithm with $D = P = \varnothing$ is obviously correct. □

```
Input: A graph $G$, a treedepth decomposition $T$ of $G$, a node $x$ of $T$ and sets $P, D \subseteq V(G)$.
Output: The size of a minimum Dominating Set.
if $x$ is a leaf in $T$ then
    if $x \notin N_G[D]$ or $P \not\subseteq N_G[D]$ then $D := D \cup \{x\}$ ;
    if $P \not\subseteq N_G[D]$ then return $\infty$ ;
    else if $x \in D$ then return $1$ ;
    else return $0$ ;
$result := \infty$;
$P := P \cup \{x\}$;
foreach partition $S_1 \uplus \cdots \uplus S_\ell$ of $P$ do
    $L := |P|$-element array of ordered lists;
    $L' := |P|$-element array of ordered lists;
    $baseline := 0$;
    $baseline' := 0$;
    foreach child $y$ of $x$ in $T$ do
        $b := domset(G, T, y, \varnothing, D)$;
        $baseline := baseline + b$;
        $b' := domset(G, T, y, \varnothing, D \cup \{x\})$;
        $baseline' := baseline + b'$;
        for $S_i \in \{S_1, \dots, S_\ell\}$ do
            $c :=$ domset$(G, T, y, S_i, D) - b$;
            $c' :=$ domset$(G, T, y, S_i, D \cup \{x\}) - b'$;
            Insert $(c, y)$ into ordered list $L[i]$ and keep only smallest $\ell$ elements;
            Insert $(c', y)$ into ordered list $L'[i]$ and keep only smallest $\ell$ elements;
    /* Find minimal cost of dominating $\{S_1,\dots,S_\ell\}$ from $L$ and $L'$ by solving appropriate
       matching problems (see proof of Lemma 26 for details).                                */
    $result := min(result,$ find_min_solution$(L) + baseline)$;
    $result := min(result,$ find_min_solution$(L') + baseline' + 1)$;
return $result$;
```

Algorithm 5: domset

Lemma 27. *Algorithm 5 runs in time $d^{O(d^2)} \cdot n$.*

Proof. The running time when x is a leaf is bounded by $O(d^2)$, since all operations exclusively involve some subset of the d nodes in $P_x \cup \{x\}$. Since $|P| \leqslant d$ the number of partitions of P is bounded by d^d. When x is not a leaf the only time spent on computations which are not recursive calls of the algorithm are all trivially bounded by $O(d)$, except the time spent on find_min_solution, which can be solved via a matching problem in polynomial time in d (see proof of Lemma 26). The number of recursive calls that a single call on a node x makes on a child y is $O(d \cdot d^d)$ which bounds total number of calls on a single node by $d^{O(d^2)}$. This proves the claim. □

Lemma 28. *Algorithm 5 uses $O(d^3 \log d + d \log n)$ bits of space.*

Proof. There are at most d recursive calls on the stack at any point. We will show that the space used by one is bounded by $O(d^2 \log d + \log n)$. Each call uses $O(d)$ sets, all of which have size at most d. The elements contained in these sets can be represented by their position in the path to the root of T, thus they use at most $O(d^2 \log d)$ space. The arrays of ordered lists L, L' contain at most d^2 elements and all entries are $\leqslant d$ or ∞: If the additional cost (compared to the baseline cost) of dominating a block S_i of the current partition from some subtree T_y exceeds d, we disregard this possibility—it would be cheaper to just take all vertices in S_i, a possibility explored in a different branch. To find a minimal solution from the table we need to avoid using the same subtree to dominate more than one element of the partition; however, at any given moment we only need to distinguish at most d^2 subtrees. Thus the size of the arrays L and L' is bounded by $O(d^2 \log d)$. The only other space consumption is caused by a constant number of variables (*result*, *baseline*, *baseline'*, b, b' and x) all of them $\leqslant n$. Thus the space consumption of a single call is bounded by $O(d^2 \log d + \log n)$ and the lemma follows. □

FAST DOMINATING SET USING $O(2^d d \log d + d \log n)$ SPACE

We have seen that it is possible to solve Dominating Set on low-treedepth graphs in a space-efficient manner. However, we traded exponential space against superexponential running time and it is natural to ask whether there is some middle ground. We present Algorithm 6 to answer this question: its running time $O(3^d \log d \cdot n)$ is competitive with the default dynamic programming but its space complexity $O(2^d \log d + d \log n)$ is exponentially better. The basic idea is to again branch from the top deciding if the current node x is in the dominating set or not. Intertwined in this branching we compute a function which for a subtree T_x and a set $S \subseteq P_x$ gives the cost of dominating $V(T_x) \cup S$ from T_x. For each recursive call on a node we only need this function for subsets of P_x which are not dominated. If d' is the number of nodes of P_x that are currently contained in D, the function only needs to be computed for $2^{d-d'}$ sets. This allows us to keep the running time of $O^*(3^d)$, since $\sum_{i=0}^{d} \binom{d}{i} \cdot 2^{d-i} = 3^d$, while only creating tables with at most $O(2^d)$ entries. By representing all values in these tables as $\leqslant d$ offsets from a base value, the space bound $O(2^d d \log d + d \log n)$ follows. Part of the algorithm will be *convolution* operations.

Definition 37 (Convolution)**.** For two functions M_1, M_2 with domain 2^U for some ground-set U we use the notation $M_1 * M_2$ to denote the *convolution* $(M_1 * M_2)[X] := \min_{A \uplus B = X} M_1[A] + M_2[B]$, for all $X \subseteq U$.

Theorem 10. *For a graph G with treedepth decomposition T, Algorithm 6 finds the size of a minimum dominating set in time $O(3^d \log d \cdot n)$ using $O(2^d d \log d + d \log n)$ bits of space.*

```
Input: A graph $G$, a treedepth decomposition $T$ of $G$, a node $x$ of $T$ and a set $D \subseteq V(G)$.
Output: The size of a minimum Dominating Set.
1  $M, M_1, M_2$ := are empty associative arrays. If a set is not in the array its value is $\infty$;
2  if $x$ is a leaf in $T$ then
3      $M[N_G[x] \setminus D] := 1$;
4      if $x \in N_G[D]$ then $M[\emptyset] := 0$ ;
5      return $M$;
   /* Assume the children of $x$ are $\{y_1, \dots, y_\ell\}$.                                */
6  for $i \in \{1, \dots, \ell\}$ do
7      $M' :=$ domset$(G, T, y_i, D)$;
8      $M_1 := M_1 * M'$;
   /* $x$ is not in the dominating set.  Discard entries where $x$ is undominated.   */
9  if $x \notin N_G[D]$ then delete all entries $S$ from $M_1$ where $x \notin S$ ;
10 for $i \in \{1, \dots, \ell\}$ do
11     $M' :=$ domset$(G, T, y_i, D \cup \{x\})$;
12     $M_2 := M_2 * M'$;
13 foreach $S \in M_2$ do $M[S] := M[S] + 1$ ;
14 $M := M_1 * M_2$;
   /* Forget $x$.                                                                     */
15 foreach $S \in M$ where $x \notin S$ do $M[S] := min(M[S], M[S \cup \{x\}])$ ;
16 Delete all entries $S$ from $M$ where $x \in S$;
17 if $x$ is the root of $T$ then return $M[\emptyset]$ ;
18 else return $M$ ;
```

Algorithm 6: domset

We divide the proof into lemmas as before.

Lemma 29. *Algorithm 6 called on $G, T, r, \emptyset$, where T is a treedepth decomposition of G with root r, returns the size of a minimum dominating set of G.*

Proof. Notice that the associative array M represents a function which maps subsets of $P_x \setminus D$ to integers and ∞. At the end of any recursive call, $M[S]$ for $S \subseteq P_x \setminus D$ should be the size of a minimal dominating set in T_x which dominates T_x and S assuming that the nodes in D are part of the dominating set. We will prove this inductively.

Assume x is a leaf. We can always take x into the dominating set at cost one. In case x is already dominated we have the option of not taking it, dominating nothing at zero cost. This is exactly what is computed in lines 2–5.

Assume now that x is an internal, non-root node of T. First, in lines 6–9 we assume that x is not in the dominating set. By inductive assumption calling *domset* on a child y of x returns a table which contains the cost of dominating T_y and some set $S \subseteq P_y \setminus D$. By convoluting them all together M_1 represents a function which gives the cost of dominating some set $S \subseteq (P_x \cup \{x\}) \setminus D$ and all subtrees rooted at children of x. We just need to take care that x is dominated. If x is not dominated by a node in D, then it must be dominated from one of the subtrees. Thus we are only allowed to retain solutions which dominate x from the subtrees. We take care of this on line 9. After this M_1

represents a function which gives the cost of dominating some set $S \subseteq (P_x \cup \{x\}) \setminus D$ and T_x assuming x is not in the dominating set. Then we compute a solution assuming x is in the dominating set in lines 10–13. We first merge the results on calls to the children of x via convolution. Since we took x into the dominating set we increase the cost of all entries by one. After this M_2 represents the function which gives the cost of dominating some set $S \subseteq P_x \setminus D$ and T_x assuming x is in the dominating set. We finally merge M_1 and M_2 together with the min-sum convolution. Since we have taken care that all solutions represented by entries in M dominate x we can remove all information about x. We do this in lines 15–16. Finally, M represents the desired function and we return it. When x is the root, instead of returning the table we return the value for the only entry in M, which is precisely the size of a minimum dominating set of G. □

To prove the running time of Algorithm 6 we will need the values M to be all smaller or equal to the depth of T. Thus we first prove the space upper bound. In the following we treat the associative arrays M, M_1 and M_2 as if the entries were values between 0 and n. We will show that we can represent all values as an offset $\leqslant d$ of a single value between 0 and n.

Lemma 30. *Algorithm 6 uses $O(2^d d \log d + d \log n)$ bits of space.*

Proof. Let d be the depth of the provided treedepth decomposition. It is clear that the depth of the recursion is at most d. Any call to the function keeps a constant number of associative arrays and nodes of the graph in memory. By construction these associative arrays have at most 2^d entries. For any of the computed arrays M the value of $M[\varnothing]$ and $M[S]$ for any $S \neq \varnothing$ can only differ by at most d. We can thus represent every entry for such a set S as an offset from $M[\varnothing]$ and use $O(2^d \log d + \log n)$ space for the tables. This together with the bound on the recursion depth gives the bound $O(2^d d \log d + d \log n)$. □

Lemma 31. *Algorithm 6 runs in time $O(3^d \log d \cdot n)$.*

Proof. On a call on which d' nodes of P_x are in the dominating set the associative arrays have at most 2^s entries for $s = d - d'$. As shown above the entries in the arrays are $\leqslant s$ (except one). Hence, we can use fast subset convolution to merge the arrays in time $O(2^s \log s)$ [28]. It follows that the total running time is bounded by

$$O\big(n \cdot \sum_{i=0}^{d} \binom{d}{i} \cdot 2^{d-i} \log(d-i)\big) = O(3^d \log d \cdot n)$$

and thus the lemma follows. □

16 CONCLUSION

We have shown that single-pass dynamic programming algorithms on treedepth, tree or path decompositions without preprocessing of the input must use space exponential in the width/depth, confirming a common suspicion and proving it rigorously for the first time. This complements previous SETH-based arguments about the running time of arbitrary algorithms on low treewidth graphs. We further demonstrate that treedepth allows non-DP linear-time algorithms that only use polynomial space in the depth of the provided decomposition. Both our lower bounds and the provided algorithm for Dominating Set appear as if they could be special cases of a general theory to be developed in future work and we further ask whether our result can be extended to less stringent definitions of "dynamic programming algorithms."

It would be great to be able to characterize exactly which problems can be solved in linear-fpt time using $\text{poly}(d) \cdot \log n$ space. Tobias Oelschlägel proved as part of his master thesis [199] that the ideas presented here can be extended to the framework of Telle and Proskurowski for graph partitioning problems [225]. Mimicking the development for treewidth would point to extending this result to MSO. Sadly, a proven double exponential dependency on the run-time of model-checking MSO parameterized by the size of a vertex cover implies that no such result is possible [156]. Is there a characterization that better captures for which problems this is possible? Previous research that might be relevant to this endeavor has investigated the height of the tower in the running time for MSO model-checking on graphs of bounded treedepth [96].

Despite the less-than-ideal theoretical bounds of the presented Dominating Set algorithms, the opportunities for heuristic improvements are not to be slighted. Take the pure branching algorithm presented in Section 15. During the branching procedure, we generate all partitions from the root-path starting at the current vertex. However, we actually only have to partition those vertices that are not dominated yet (by virtue of being themselves in the dominating set or being dominated by another vertex on the root-path). A sensible heuristic as to which branch—including the current vertex in the dominating set or not—to explore first, together with a *branch & bound* routine should keep us from generating partitions of very large sets. A similar logic applies to the mixed dynamic programming/branching algorithm since the tables only have to contain information about sets that are not yet dominated. It might thus be possible to keep the tables a lot smaller than their theoretical bounds indicate.

Furthermore, it seems reasonable that in practical settings, the nodes near the root of treedepth decompositions are more likely to be part of a minimal dominating set. If this is true, computing a treedepth decomposition would serve as a form of smart preprocessing for the branching, a rough "plan of attack", if you will. How much such a *guided branching* improves upon known branching algorithms in practice is an interesting avenue for further research.

It is still an open question, proposed by Michał Pilipczuk during GROW 2015, whether Dominating Set can be solved in time $(3-\varepsilon)^d \cdot \mathrm{poly}(n)$ when parameterized by treedepth. Our lower bound result implies that if such an algorithm exists, it cannot be a straightforward dynamic programming algorithm.

Part IV

MOTIF COUNTING ON RANDOM INTERSECTION GRAPHS

17 SPARSITY OF COMPLEX NETWORKS

There has been a recent surge of interest in analyzing large graphs, stemming from the rise in popularity (and scale) of social networks and significant growth of relational data in science and engineering fields (e.g. gene expressions, cybersecurity logs and neural connectomes). One significant challenge in the field is the lack of deep understanding of the underlying structure of various classes of real-world networks.

Although it is widely accepted that complex networks tend to be sparse (in terms of edge density), this property usually is not sufficient to improve algorithmic tractability: many NP-hard problems on graphs, for instance, remain NP-hard when restricted to graphs with bounded average degree. In contrast, graph classes that are *structurally* sparse (bounded treewidth, planar, etc.) often admit more efficient algorithms—in particular when viewed through the lens of parameterized complexity. Consequently, we are interested whether random graph models and, by extension, real-world networks exhibit any form of structural sparseness that might be exploitable algorithmically.

As a first step, we would like that a graph is not only sparse on average, but that this property extends to all its subgraphs. This motivates a very general class of structurally sparse graphs—those of bounded *degeneracy*, a property that has been previously studied in the context of both graph theory and complex networks [8, 9, 10, 52, 104, 148].

Definition 38 (k-core)**.** The *k-core* of G, denoted C_k, is the maximum induced subgraph of G in which all vertices have degree at least k. The *degeneracy* of G is the maximum k so that C_k is nonempty (equivalently, the least positive integer k such that every induced subgraph of G contains a vertex with at most k neighbors).

Some classes of graphs where the members have bounded degeneracy have stronger structural properties—here we focus on graphs of bounded expansion (see Definition 14). In the context of networks, bounded expansion captures the idea that networks decompose into small dense structures (e.g. communities) connected by a sparse global structure.

Intuitively, a graph class has bounded expansion if for every member G, one cannot form arbitrarily dense graphs by contracting subgraphs of small radius. Formally, the degeneracy of every minor of G is bounded by a function of the *depth* of that minor (the maximum radius of its branch sets). Bounded expansion offers a structural generalization of both bounded-degree and graphs excluding a (topological) minor.

This property presents challenges for empirical evaluation, since bounded expansion is only defined with respect to graph classes (not for single instances). As is typical in the study of network structure, we instead ask how the properties behave

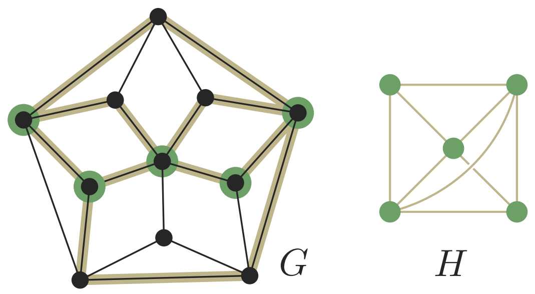

Figure 17.1: The graph H on the right is a 1-shallow topological minor of G, as witnessed by the $\leqslant$ 2-subdivision highlighted inside G. Further, H is the densest among all 1-shallow topological minor of G: hence $\widetilde{\nabla}_1(G) = |E(H)|/|V(H)| = 9/5$.

with respect to randomized models which are designed to mimic aspects of network formation and structure. In related work several previously proposed models for random networks have been analyzed and shown to produce graphs belonging to a class of bounded expansion w.h.p. [65]:

- Graphs sampled with the *Molloy-Reed configuration model* (including a variation of the model which achieves high clustering) or the *Chung-Lu model* with a prescribed sparse degree sequence (including heavy-tailed degree distributions)
- *Perturbed bounded-degree graphs*
- *Stochastic block models* with small probabilities

Furthermore, experimental evidence for the claim that many complex networks have bounded expansion was given, by measuring the "low treedepth coloring number" on a corpus of real-world data.

We expand on this previous work by considering the *random intersection graph model* introduced by Karoński, Scheinerman and Singer-Cohen [136, 220] which has recently attracted significant attention in the literature [31, 64, 105, 131, 214]. *Random intersection graphs* are based on the premise that network edges often represent underlying shared interests or attributes. The model first creates a bipartite object-attribute graph $B = (V, A, E)$ by adding edges uniformly at random with a fixed per-edge probability $p(\alpha)$, then considers the *intersection graph*: $G := (V, E')$ where $xy \in E'$ if the neighborhoods of the vertices x, y in B have a non-empty intersection. The parameter α controls both the ratio of attributes to objects and the probability p: for n objects the number of attributes m is proportional to n^α and the probability p to $n^{-(1+\alpha)/2}$.

This model is attractive because they meet three important criteria: (1) the generative process makes sense in many real-world contexts, for example collaboration networks of scientists [195, 232]; (2) they are able to generate graphs which match key empirically established properties of real data—namely sparsity, (tunable) clustering and assortativity [30, 31, 64]; and (3) they are relatively mathematically tractable due to significant amounts of independence in the underlying edge creation process.

We will show that the random intersection graphs model generates graphs that belong w.h.p. to a graph class of bounded expansion precisely when it generates degenerate graphs. More specifically, we present the following results on the structure of random intersection graphs.

(i) For $\alpha \leqslant 1$, random intersection graphs are w.h.p. somewhere dense (and thus do not have bounded expansion) and have unbounded degeneracy.

(ii) For $\alpha > 1$, random intersection graphs have w.h.p. bounded expansion (and thus constant degeneracy).

While in general a graph class with unbounded degeneracy is not necessarily somewhere dense, the negative proofs presented here show that members of the graph class contain w.h.p. large cliques. This simultaneously implies unbounded degeneracy and that the class is somewhere dense (as a clique is a 0-subdivision of itself). Consequently, we prove a clear dichotomy: random intersection graphs are either structurally sparse or somewhere dense.

In particular, the second result strengthens the original claim that the model generates sparse graphs for $\alpha > 1$, by establishing they are in fact *structurally* sparse in a robust sense. It is of interest to note that random intersection graphs only exhibit tunable clustering when $\alpha = 1$ [64], when our results indicate they are not structurally sparse (in any reasonable sense).[1]

It is easy to see that the degeneracy is lower-bounded by the size of the largest clique. Thus, the degeneracy of intersection graphs is bounded below by the maximum attribute degree in the associated bipartite graph since each attribute contributes a complete subgraph of size equal to its degree to the intersection graph. For certain parameter values, this lower bound will, w.h.p., give the correct order of magnitude of the degeneracy of the graph.

Algorithmically, this property is extremely useful: every first-order-definable problem is decidable in linear fpt-time in these classes [78]. In the following we highlight domain-specific applications of computing the frequency of small fixed pattern graphs inside a network. In particular, the concept of *network motifs* and *graphlets* has proven very useful in the area of computational biology.

A network motif is a (labeled) subgraph that appears more often in a real-world network than one would expect by pure chance. The hypothesis here is that such a structure is likely to have some particular significance [184]. By now, motifs have been found in a wide range of domains, such as protein-protein-interaction networks [6], brain networks [221] and electronic circuits [124]. For an extensive overview see the surveys of Kaiser, Ribeiro and Silva [209] and Masoudi-Nejad, Schreiber and Kashani [179].

1 This is not tautological—a previous result shows that constant clustering and bounded expansion are not orthogonal [65].

Graphlets are a related concept, which is used to "fingerprint" networks instead of identifying interesting local structures. Pržulj introduced the *graphlet degree distribution* as a way of measuring network similarity [206]. The idea is to enumerate all connected graphs of small size (originally up to size five) and count for every node in the network how often they appear as part of such graphs (taking automorphisms into account). The degree distribution is then how many vertices are part of $0, 1, 2, \ldots$ subgraphs isomorphic to G_i for every G_i—more precisely, in how many orbits of the automorphism group it appears in. Notice that if we only take the edge as a graphlet this becomes the classical degree distribution.

This distribution can be used to measure the similarity of multiple networks, especially biological networks [117]. Furthermore, the local structure around a vertex can reveal a domain-specific function, such as in protein-protein interaction networks, where local structure correlates with biological activity [183]. This has been used to identify cancer genes [182] and construct phylogenetic trees [154]. Graphlets have also been used in aiding the analysis of workplace dynamics [226], photo cropping [48] and DoS attack detection [203].

Ugander *et al.* [227] showed with their empirical analysis via subgraph counting and subsequent modeling of social networks that there is a bias towards the occurrence of certain subgraphs. This indicates that the frequencies of small subgraphs are a good indicator for the social domain, similar to the role of graphlet frequencies in biological networks.

For graph classes of bounded expansion counting the number of satisfying assignments of a fixed Boolean query is possible in linear time on a labeled graph (Theorem 18.9 [193]), which immediately implies that (labeled) graphlet and motif counting can be computed in linear time on a graph class of bounded expansion. This result is achieved by counting on graphs of bounded treedepth (Lemma 17.3 [193]) with a running time of $O(2^{hd} \cdot hd \cdot n)$, where h is the number of nodes in the graphlet or motif and d is the depth of a treedepth decomposition. We provide an algorithm with a running time of $O(6^h \cdot d^h \cdot h^2 \cdot n)$. This achieves a better running time when used to count on graphs of bounded expansion, since then d will equal h, i.e., a constant. With a small modification, this algorithm can count how many times a node appears as a specific node of a specific graphlet or motif.

18 RANDOM INTERSECTION GRAPHS AND BOUNDED EXPANSION

We formalize the model and introduce another characterization of bounded expansion and the concept of stable-r subdivisions which are used to simplify later proofs.

RANDOM INTERSECTION GRAPHS

A wide variety of random intersection graph models have been defined in the literature; in this paper, we restrict our attention to the most well-studied of these, $G(n, m, p)$, which is defined as follows:

Definition 39 (Random Intersection Graph Model)**.** Fix positive constants α, β and γ. Let B be a random bipartite graph on parts of size n and $m = \beta n^{\alpha}$ with each edge present independently with probability $p = \gamma n^{-(1+\alpha)/2}$. Let V (the nodes) denote the part of size n and A (the attributes) the part of size m. The associated *random intersection graph* $G = G(n, m, p)$ is defined on the nodes V: two nodes are adjacent in G if they share (are both adjacent to in B) at least one attribute in A.

We note that $G(n, m, p)$ defines a distribution $\mathcal{G}_n$ on graphs with n vertices. The notation $G = G(n, m, p)$ denotes a graph G that is randomly sampled from the distribution $\mathcal{G}_n$. Throughout the manuscript, given a random intersection graph $G(n, m, p)$ we will often refer to B, the associated bipartite graph on n nodes and m attributes from which G is formed.

DEGENERACY & EXPANSION

We now state a characterization of bounded expansion which is often helpful in establishing the property for classes formed by random graph models.

Proposition 7 ([193, 194])**.** *A class $\mathcal{C}$ of graphs has bounded expansion if and only if there exists real-valued functions $f_1, f_2, f_3, f_4 \colon \mathbb{R} \to \mathbb{R}^+$ such that the following two conditions hold:*

(i) For all positive ε and for all graphs $G \in \mathcal{C}$ with $|V(G)| > f_1(\varepsilon)$, it holds that

$$\frac{1}{|V(G)|} \cdot |\{v \in V(G) \colon \deg(v) \geqslant f_2(\varepsilon)\}| \leqslant \varepsilon.$$

(ii) For all $r \in \mathbb{N}$ and for all $H \subseteq G \in \mathcal{C}$ with $\widetilde{\nabla}_r(H) > f_3(r)$, it follows that

$$|V(H)| \geqslant f_4(r) \cdot |V(G)|.$$

Intuitively, this states that any class of graphs with bounded expansion is characterized by two properties:

(i) All sufficiently large members of the class have a small fraction of vertices of large degree.

(ii) All subgraphs of $G \in \mathcal{C}$ whose shallow topological minors are sufficiently dense must necessarily span a large fraction of the vertices of G.

STABLE r-SUBDIVISIONS

In order to disprove the existence of an r-shallow topological minor of a certain density δ, we introduce a stronger topological structure.

Definition 40 (Stable r-subdivision)**.** Given graphs G, H we say that G *contains* H *as a stable r-subdivision* if G contains H as a $\frac{r}{2}$-shallow topological minor with model G' such that every path in G' corresponding to an edge in H has exactly length $r+1$ and is an induced path in G.

A stable r-subdivision is by definition a shallow topological minor, thus the existence of an r-subdivision of density δ implies that $\widetilde{\nabla}_{\frac{r}{2}}(G) \geqslant \delta$. We prove that the densities are also related in the other direction.

Lemma 32. *A graph G with $\widetilde{\nabla}_{\frac{r}{2}}(G) \geqslant \delta$ contains a stable i-subdivision of density at least $\delta/(r+1)$ for some $i \in \{0, \ldots, r\}$.*

Proof. Consider a $\frac{r}{2}$-shallow topological minor H of G with density at least δ. Let $H' \subseteq G$ be the model of H and let $\lambda \colon V(H') \to V(H) \cup E(H)$ be a mapping that maps nails of the model to vertices of the minor and subdivision vertices of the model to their respective edge in the model. Consider the preimage λ^{-1}. As a slight abuse of notation, we can consider λ^{-1} as a map to (possibly empty) paths of H: indeed, we can assume that every edge of H is mapped by λ^{-1} to an *induced* path in H'. If H' uses any non-induced paths, we can replace each such path a by a (shorter) induced path and obtain a (different) model of H with the desired property.

We partition the edges of H by the length of their respective paths in the model: Define $E_\ell = \{e \in H \mid |\lambda^{-1}(e)| = \ell\}$ for $0 \leqslant \ell \leqslant r+1$. Since $|E(H)| = \bigcup_{0 \leqslant \ell \leqslant r+1} |E_\ell| \geqslant \delta|V(H)|$, there exists at least one set E_ℓ such that its size $|E_\ell| \geqslant \delta|V(H)|/(r+1)$. Then the subgraph $(V(H), E_\ell)$ is a stable ℓ-subdivision of G. $\square$

Thus, to show that a graph has no r-shallow minor of density δ, it suffices to prove that no stable i-subdivision of density $\delta/(2r+1)$ exists for any $i \in \{0, \ldots, 2r\}$. We note that the other direction would not work, since the existence of a stable i-subdivision for some $i \in \{0, \ldots, 2r\}$ of density $\delta/(2r+1)$ does not imply the existence of an r-shallow topological minor of density δ.

19 STRUCTURAL SPARSITY

In this section we will characterize a clear break in the sparsity of graphs generated by $G(n, m, p)$, depending on whether α is strictly greater than one. In each case, we analyze (probabilistically) the degeneracy and expansion of the generated class.

Theorem 11. *Fix constants α, β and γ. Let $m = \beta n^{\alpha}$ and $p = \gamma n^{-(1+\alpha)/2}$. Let $G = G(n, m, p)$. Then the following hold w.h.p.*

(i) If $\alpha < 1$, $G(n, m, p)$ is somewhere dense and G has degeneracy $\Omega(\gamma n^{(1-\alpha)/2})$.

(ii) If $\alpha = 1$, $G(n, m, p)$ is somewhere dense and G has degeneracy $\Omega(\frac{\log n}{\log \log n})$.

(iii) If $\alpha > 1$, $G(n, m, p)$ has bounded expansion and thus G has degeneracy $O(1)$.

We prove each of the three cases of Theorem 11 separately.

PROOF OF MAIN THEOREM WHEN $\alpha \leqslant 1$

When $\alpha \leqslant 1$, we prove that w.h.p. the random intersection graph model generates graph classes with unbounded degeneracy by establishing the existence of a high-degree attribute in the associated bipartite graph (thus lower-bounding the clique number). The proof is divided into two lemmas, one for $\alpha < 1$ and one for $\alpha = 1$, for which we prove different lower bounds.

Lemma 33. *Fix constants $\alpha < 1, \beta$ and γ. If $m = \beta n^{\alpha}$ and $p = \gamma n^{-(1+\alpha)/2}$, then w.h.p. $G = G(n, m, p)$ has degeneracy $\Omega(\gamma n^{(1-\alpha)/2})$.*

Proof. Let $G = G(n, m, p)$ and $B = (V, A, E)$ be the bipartite graph associated with G. Define the random variable X_i to be the number of nodes in V connected to a particular attribute a_i. Then $X_i \sim \text{Binom}(n, p)$ and $\mathbb{P}[X_i < np - 1] \leqslant 1/2$, since the average of X_i lies between $\lfloor np \rfloor$ and $\lceil np \rceil$. Let $\mathcal{S}$ be the event that $|X_i| < np - 1$ for all $i \in [1, m]$. Since the number of vertices attached to each attribute is independent,

$$\mathbb{P}[\mathcal{S}] = \prod_{i=1}^{m} (1 - \mathbb{P}[X_i \geqslant np - 1]) \leqslant [1 - (1 - 1/2)]^m = 2^{-m}.$$

Now, it follows that $\lim_{n\to\infty} \mathbb{P}[\mathcal{S}] = 0$ and w.h.p. the graph G contains a clique of size $np - 1 = \gamma n^{(1-\alpha)/2} - 1$, and thus has degeneracy at least $\gamma n^{(1-\alpha)/2} - 1$. □

Corollary 5. *Fix constants $\alpha < 1, \beta$ and γ. If $m = \beta n^{\alpha}$ and $p = \gamma n^{-(1+\alpha)/2}$, then w.h.p. $G(n, m, p)$ is somewhere dense.*

Proof. The proof of Lemma 33 shows that w.h.p. a clique of size $\gamma n^{(1-\alpha)/2}$ exists already as a subgraph (i.e., a 0-subdivision) in every $G \in G(n,m,p)$. □

The following lemma addresses the case when the attributes grow at the same rate as the number of nodes. We note that Bloznelis and Kurauskas independently proved a similar result (using a slightly different random intersection graph model) [32]; we include a slightly more direct proof here for completeness.

Lemma 34. *Fix constants $\alpha = 1, \beta$ and γ. Then a random graph $G = G(n,m,p)$ has degeneracy $\Omega(\frac{\log n}{\log\log n})$ w.h.p.*

Proof. Let c be any constant greater than one. We will show that for every $k \leqslant \frac{\log n}{\log\log n}$, a random graph $G \in G(n,m,p)$ contains a clique of size k with probability $\Omega(1 - n^{-c})$. Fix an attribute a. The probability that a has degree at least k in the bipartite graph is at least the probability that it is exactly k, hence

$$\binom{n}{k} p^k (1-p)^{n-k} \geqslant \binom{n}{k} p^k (1-p)^n \geqslant \frac{\gamma^k}{e^\gamma k^k}.$$

We will show that this converges fast enough for $\gamma < 1$; the case for $\gamma \geqslant 1$ works analogously. Therefore the probability that none of the $m = \beta n$ attributes has degree at least k is at most

$$\left(1 - \frac{\gamma^k}{e^\gamma k^k}\right)^{\beta n} \leqslant e^{-\beta n\left(\frac{\gamma^k}{e^\gamma k^k}\right)}.$$

We prove that this probability is smaller than n^{-c} by showing that

$$\frac{\beta}{e^\gamma}\frac{n\gamma^k}{k^k} \geqslant c \cdot \log n, \tag{19.1}$$

when $k = \frac{\log n}{\log\log n}$. Let $c' = ce^\gamma/\beta$. Then to show Inequality 19.1 holds, it is enough to show

$$\begin{aligned} n\frac{(\gamma\log\log n)^{\log n/\log\log n}}{(\log n)^{\log n/\log\log n}} &= n\frac{(\gamma\log\log n)^{\log n/\log\log n}}{2^{\log\log n(\log n/\log\log n)}} \\ &= (\gamma\log\log n)^{\frac{\log n}{\log\log n}} \geqslant c' \cdot \log n. \end{aligned}$$

Comparing the functions $e^{x/\log x}$ and xc', we see that for large enough positive x,

$$x > \log c' \log x + \log^2 x$$

and equivalently

$$e^{x/\log x} > c' \cdot x.$$

Therefore for large enough n, $e^{\log n / \log\log n} > c' \cdot \log n$, and in particular for $n > e^{e^{e/\gamma}}$,

$$(\gamma \log\log n)^{\frac{\log n}{\log\log n}} \geqslant c' \cdot \log n,$$

as previously claimed. This shows the probability that no attribute has degree at least $\log n / \log\log n$ is at most $O(n^{-c})$ and the claim follows. □

Corollary 6. *Fix constants $\alpha = 1, \beta$ and γ. If $m = \beta n^{\alpha}$ and $p = \gamma n^{-(1+\alpha)/2}$, then w.h.p. $G(n, m, p)$ is somewhere dense.*

Proof. Lemma 34 is proven by showing that for a clique of size $\Omega(\log n/(\log\log n))$ it holds w.h.p. that there exists as a subgraph (i.e., a 0-subdivision) in every graph in $G(n, m, p)$. □

PROOF OF MAIN THEOREM WHEN $\alpha > 1$

In this section, we focus on the case when $\alpha > 1$. This is the parameter range in which the model generates sparse graphs. Before beginning, we note that if $G(n, m, p)$ has bounded expansion w.h.p., then for any $p' \leqslant p$ and $m' \leqslant m$ it follows that w.h.p. $G(n, m', p')$ also has bounded expansion by a simple coupling argument. Thus we can assume without loss of generality that both γ and β are greater than one. For the remainder of this section, we fix the parameters $\gamma, \beta, \alpha > 1$, the resulting number of attributes $m = \beta n^{\alpha}$ and the per-edge probability $p = \gamma n^{-(1+\alpha)/2}$.

Bounded Attribute-Degrees

As mentioned before, for a random intersection graph to be degenerate, the attributes of the associated bipartite graph must have bounded degree. We prove that w.h.p., this necessary condition is satisfied.

Lemma 35. *Let $c \geqslant 1$ be a constant such that $2\frac{\alpha+c}{\alpha-1} > \beta\gamma e$. Then the probability that there exists an attribute in the bipartite graph associated with $G(n, m, p)$ of degree higher than $2\frac{\alpha+c}{\alpha-1}$ is $O(n^{-c})$.*

Proof. Taking the union bound, the probability that some attribute has degree larger than d is upper bounded by

$$m\binom{n}{d}p^d \leqslant \frac{\beta e^d \gamma^d}{d^d} \cdot \frac{n^{\alpha+d}}{n^{\frac{\alpha+1}{2}d}},$$

where the first fraction is bounded by a constant as soon as $d > e\beta\gamma$. Then we achieve an upper bound of $O(n^{-c})$ as soon as $\frac{\alpha+1}{2}d - d - \alpha > c$, or equivalently, $d > 2\frac{\alpha+c}{\alpha-1}$, proving the claim. □

This allows us to assume for the remainder of the proof that the maximum attribute degree is bounded.

Stable-r subdivisions

We now establish the probability of having this structure in the random intersection graph model, noting that the following structural result is surprisingly useful and appears to have promising applications beyond this work. We will argue that a dense subdivision in G implies the existence of a dense subgraph in the associated bipartite graph. We show this by considering the existence of a stable r-subdivision where all paths are induced which is generated by a minimal number of attributes. Notice that if a model of some graph H exists so does a model with these properties. This allows us to only consider attributes with minimum degree two, since every edge in the path is generated by a different attribute. This is key to prove the following theorem.

Theorem 12. *Let $c \geqslant 1$ be a constant and let $\phi = (6eg\beta\gamma\delta r)^{5\delta 2r/(\alpha-1)}$. The probability that $G(n, m, p)$ contains a stable r-subdivision with k nails for $r \geqslant 1$ and of density $\delta > 1$ is at most*

$$r\delta k \cdot \left(\frac{\phi}{n}\right)^{\frac{\alpha-1}{2}k}$$

Proof. Let us first bound the probability that the bipartite graph associated with $G = G(n, m, p)$ contains a dense subgraph. We will then argue that a dense subdivision in G implies the existence of such a dense bipartite subgraph.

Let $\mathbb{P}_{\text{dense}}(\kappa, \nu, \lambda)$ be the probability that there exists sets $V' \subseteq V$, $A' \subseteq A$, of size κ and ν respectively, such that there exist at least λ edges between nodes of V' and A'. It is easy to see that this probability is bounded by

$$\mathbb{P}_{\text{dense}}(\kappa, \nu, \lambda) \leqslant \binom{n}{\kappa}\binom{m}{\nu} \sum_{d_1,\ldots,d_\nu} \prod_{i=1}^{\nu} \binom{\kappa}{d_i} p^{d_i}, \tag{19.2}$$

where $d_1, \ldots, d_\nu$ represent all possible choices of the degrees of ν attributes such that $\sum_{i=1}^{\nu} d_i = \lambda$. By Lemma 35, w.h.p. $d_i \leqslant g$ and thus w.h.p. there are at most g^ν terms in the sum of Equation 19.2. Using this together with Stirling's approximation allows us to simplify the bound as follows:

$$\mathbb{P}_{\text{dense}}(\kappa, \nu, \lambda) \leqslant \left(\frac{ne}{\kappa}\right)^{\kappa} \left(\frac{gme}{\nu}\right)^{\nu} (\kappa e p)^{\lambda} = \frac{e^{\kappa+\nu+\lambda} g^\nu \beta^\nu \gamma^\lambda}{\nu^\nu} \frac{\kappa^\lambda n^{\alpha\nu+\kappa}}{\kappa^\kappa n^{\frac{\alpha+1}{2}\lambda}} \tag{19.3}$$

Consider a stable r-subdivision H in G with k nails and density δ. The model of H uses exactly $k + r\delta k$ vertices of G. Let A_H be a minimal set of attributes that generates the edges of the model of H in G. There is at least one edge between every nail and an attribute in A_H. Furthermore, since the paths connecting the nails in the model are induced, every subdivision vertex has at least two edges to the attributes A_H. We conclude that there exists a bipartite subgraph with $\kappa = k + r\delta k$ and $\lambda = 2r\delta k + k$. Since A_H is minimal, every attribute of A_H generates at least one edge in the model of

H and therefore $|A_H| \leqslant (r+1)\delta k$. Let $\delta_1 = (r\delta + 1)$ and $\delta_2 = (2r\delta + 1)$. By the bound in Equation 19.3, the probability of such a structure is at most

$$\begin{aligned}
&\sum_{\nu=r\delta k/g}^{r\delta k} \mathbb{P}_{\text{dense}}(\delta_1 k, \nu, \delta_2 k) \\
\leqslant &\sum_{\nu=r\delta k/g}^{r\delta k} \frac{e^{\delta_1 k+\nu+\delta_2 k} g^{\nu} \beta^{\nu} \gamma^{\delta_2 k}}{\nu^{\nu}(\delta_2 k)^{\delta_2 k}} \frac{(\delta_1 k)^{\delta_2 k} n^{\alpha\nu+\delta_1 k}}{(\delta_1 k)^{\delta_1 k} n^{\frac{\alpha+1}{2}\delta_2 k}}
\end{aligned}$$

Let ψ be the exponent of $1/n$ in a term of this sum. Then we have

$$\begin{aligned}
\psi &= \left(\left(\frac{\alpha+1}{2}\right)\delta_2 k - (\alpha\nu + \delta_1 k)\right) \\
&= \left(\left(\frac{\alpha+1}{2}\right)(2r\delta+1)k - (\alpha\nu + (r\delta+1)k)\right).
\end{aligned}$$

Simplifying, we see that

$$\psi = (\alpha+1)r\delta k + \frac{\alpha+2}{2}k - \alpha\nu - (r\delta+1)k = \frac{\alpha-1}{2}k + \alpha(r\delta k - \nu).$$

Thus we can rewrite the previous inequality as

$$\begin{aligned}
&\sum_{\nu=r\delta k/g}^{r\delta k} \mathbb{P}_{\text{dense}}(\delta_1 k, \nu, \delta_2 k) \\
\leqslant &\sum_{\nu=r\delta k/g}^{r\delta k} \frac{e^{\delta_1 k+\nu+\delta_2 k} g^{\nu} \beta^{\nu} \gamma^{\delta_2 k} {\delta_1}^{\delta_2 k}}{{\delta_1}^{\delta_1 k}} \frac{k^{\delta_2 k}}{\nu^{\nu} k^{\delta_1 k} n^{\alpha(r\delta k-\nu)}} \frac{1}{n^{\frac{\alpha-1}{2}k}} \\
\leqslant &\left(\frac{e^{\delta_1+r\delta+\delta_2} g^{r\delta} \beta^{r\delta} \gamma^{\delta_2} {\delta_1}^{\delta_2}}{{\delta_1}^{\delta_1}}\right)^k \sum_{\nu=r\delta k/g}^{r\delta k} \frac{k^{\delta_2 k}}{\nu^{\nu} k^{\delta_1 k} n^{\alpha(r\delta k-\nu)}} \frac{1}{n^{\frac{\alpha-1}{2}k}} \\
\leqslant &\left(\frac{e^{\delta_1+r\delta+\delta_2} g^{2r\delta} \beta^{r\delta} \gamma^{\delta_2} {\delta_1}^{\delta_2}}{{\delta_1}^{\delta_1}}\right)^k \sum_{\nu=r\delta k/g}^{r\delta k} \frac{k^{\delta_2 k}}{(r\delta k)^{\nu} k^{\delta_1 k} k^{\alpha(r\delta k-\nu)}} \frac{1}{n^{\frac{\alpha-1}{2}k}} \\
\leqslant &\left(e^{\delta_1+r\delta+\delta_2} g^{2r\delta} \beta^{r\delta} \gamma^{\delta_2} {\delta_1}^{\delta_2}\right)^k \sum_{\nu=r\delta k/g}^{r\delta k} \frac{k^{\delta_2 k}}{k^{\nu} k^{\delta_1 k} k^{\alpha(r\delta k-\nu)}} \frac{1}{n^{\frac{\alpha-1}{2}k}}
\end{aligned}$$

Let ψ' be the exponent of k in a term of this sum. Then we have

$$\begin{aligned}
\psi' &= \delta_2 k - \nu - \delta_1 k - \alpha(r\delta k) + \alpha\nu \\
&= (1-\alpha)(r\delta k) + (\alpha-1)\nu \\
&\leqslant 0.
\end{aligned}$$

Using ϕ as defined, we arrive at the following inequality.

$$\sum_{\nu=r\delta k/g}^{r\delta k} \mathbb{P}_{\text{dense}}(\delta_1 k, \nu, \delta_2 k) \leqslant \phi^{\frac{(\alpha-1)}{2}k} \sum_{\nu=r\delta k/g}^{r\delta k} \frac{1}{n^{\frac{\alpha-1}{2}k}} \leqslant r\delta k \cdot \left(\frac{\phi}{n}\right)^{\frac{\alpha-1}{2}k}$$

This completes our proof. □

Density

Before turning to our main result, we need two more lemmas that establish the probability of graphs generated using $G(n, m, p)$ have special types of dense subgraphs.

We note that it is perhaps surprising that ρ disappears in the upper bound given in the following theorem. Since we are assuming that the degree of the attributes is bounded by g, the number of attributes u must be at least $\rho/\binom{g}{2}$. Thus the ρ reappears upon expansion. Since we can bound the degree of the attributes w.h.p. when $\alpha > 1$ this theorem is generally applicable to sparse random intersection graphs.

Theorem 13. *Let $c \geqslant 1$ be a constant and let $g = 2\frac{\alpha+c}{\alpha-1}$. For $u \leqslant m, k \leqslant n$, the probability that the bipartite graph associated with $G(n, m, p)$ contains u attributes of degree $\leqslant g$ that generate at least $\rho \geqslant u$ edges between k fixed vertices is at most*

$$\left(\frac{e^{g+1}\gamma^g g\beta}{u/k}\right)^u \left(\frac{k}{n}\right)^u.$$

Proof. The probability that u attributes of maximal degree g generate at least $\rho \geqslant u$ edges between k fixed vertices can be upper bounded by

$$\binom{m}{u} \sum_{d_1,\dots,d_u} \prod_{i=1}^{u} \binom{k}{d_i} p^{d_i},$$

where $d_1, \dots, d_u$ represent all possible choices of the degrees of u attributes such that $\sum_{i=1}^{u} \binom{d_i}{2} \geqslant \rho$, i.e., the degrees of the chosen attributes can generate enough edges. Let $D = \sum_{i=1}^{u} d_i$. The following bound follows from Stirling's inequality:

$$\begin{aligned}
\binom{m}{u} \sum_{d_1,\dots,d_u} \prod_{i=1}^{u} \binom{k}{d_i} p^{d_i} &\leqslant \frac{(e\beta n^\alpha)^u}{u^u} \sum_{d_1,\dots,d_u} \prod_{i=1}^{u} \frac{(ek)^{d_i}}{d_i^{d_i}} \left(\frac{\gamma}{n^{(\alpha+1)/2}}\right)^{d_i} \\
&= \frac{(e\beta)^u n^{\alpha u}}{u^u} \sum_{d_1,\dots,d_u} \frac{e^D k^D}{\prod_{i=1}^{u} d_i^{d_i}} \frac{\gamma^D}{n^{\frac{\alpha+1}{2}D}} \\
&\leqslant \frac{(e\beta)^u (e\gamma)^{gu} n^{\alpha u}}{u^u} \sum_{d_1,\dots,d_u} \frac{k^D}{n^{\frac{\alpha+1}{2}D}}
\end{aligned}$$

Since each d_i is smaller or equal to g, we can upper bound this term by

$$\binom{m}{u} \sum_{d_1,\ldots,d_u} \prod_{i=1}^{u} \binom{k}{d_i} p^{d_i} \leqslant \frac{(e^{g+1}\gamma^g\beta)^u n^{\alpha u}}{u^u} \cdot \sum_{d_1,\ldots,d_u} \frac{k^D}{n^{\frac{\alpha+1}{2}D}}$$
$$= \left(\frac{e^{g+1}\gamma^g\beta}{u/k}\right)^u \sum_{d_1,\ldots,d_u} \frac{n^{\alpha u} k^{D-u}}{n^{\frac{\alpha+1}{2}D}}.$$

We want to show that $(n^{\alpha u}k^{D-u})/(n^{\frac{\alpha+1}{2}D})$ is bounded by $(k/n)^x$ for some $x \geqslant u$. We first look at the following inequality:

$$\left(\frac{\alpha+1}{2}\right) D - \alpha u \geqslant D - u \Leftrightarrow D \geqslant 2u$$

Notice that an attribute of degree one generates no edges, thus we can assume that all $d_i \geqslant 2$. It follows that $D \geqslant 2u$ and thus the inequality holds. It follows that

$$(n^{\alpha u}k^{D-u})/(n^{\frac{\alpha+1}{2}D}) \leqslant \left(\frac{k}{n}\right)^{D-u} \leqslant \left(\frac{k}{n}\right)^u$$

The probability of u attributes generating at least ρ edges between k vertices is then at most

$$\binom{m}{u} \sum_{d_1,\ldots,d_u} \binom{k}{d_i} p^{d_i} \leqslant \left(\frac{e^{g+1}\gamma^g\beta}{u/k}\right)^u \sum_{d_1,\ldots,d_u} \left(\frac{k}{n}\right)^u.$$

Finally, since any d_i can be at most g we can get rid of the sum by multiplying with a g^u factor.

$$\left(\frac{e^{g+1}\gamma^g\beta}{u/k}\right)^u \sum_{d_1,\ldots,d_u} \left(\frac{k}{n}\right)^u \leqslant \left(\frac{e^{g+1}\gamma^g g\beta}{u/k}\right)^u \left(\frac{k}{n}\right)^u.$$

□

The following lemma is a rather straightforward consequence of Theorem 13.

Lemma 36. *Let $c \geqslant 1$ be a constant, $g = 2\frac{\alpha+c}{\alpha-1}$, $g' = \binom{g}{2}$ and $\delta > e^{g+1}\gamma^g g g'\beta$. Then the probability that $G(n,m,p)$ contains a subgraph of density δ on k nodes is at most*

$$\delta k \left(\frac{k}{n}\right)^{\frac{\delta k}{g'}}.$$

Proof. By Lemma 35 we can disregard all graphs whose associated bipartite graph have an attribute of degree greater than g. We can bound the probability as follows:

$$\sum_{u=\frac{\delta k}{g'}}^{\delta k} \binom{m}{u} \sum_{d_1,\ldots,d_u} \binom{k}{d_i} p^{d_i} \qquad (19.4)$$

where $d_1, \ldots, d_u$ represent the degrees of the u attributes such $\sum_{i=1}^{u} \binom{d_i}{2} \geqslant \delta k$ (i.e., the degrees of the u attributes that generate all direct edges).

Using Theorem 13, the right hand side of Equation (19.4) is bounded by

$$\sum_{u=\frac{\delta k}{g'}}^{\delta k} \left(\frac{e^{g+1}\gamma^g g \beta}{u/k} \right)^u \left(\frac{k}{n} \right)^u \leqslant \sum_{u=\frac{\delta k}{g'}}^{\delta k} \left(\frac{e^{g+1}\gamma^g g g' \beta}{\delta} \right)^u \left(\frac{k}{n} \right)^u,$$

using the fact that $u/k \geqslant \delta/g'$. Since we set up $\delta \geqslant e^{g+1}\gamma^g g g' \beta$, we can cancel these terms and simplify the above to

$$\mathbb{P}_{\text{direct}} \leqslant \sum_{u=\frac{\delta k}{g'}}^{\delta k} \left(\frac{k}{n} \right)^u \leqslant \delta k \left(\frac{k}{n} \right)^{\frac{\delta k}{g'}}$$

using the fact that k/n is smaller than one. □

Main Result

We finally have all the necessary tools to prove the main theorem of this section.

Theorem 14. *Fix positive constants $\alpha > 1$, β and γ. Then w.h.p. the class of random intersection graphs $G(n, m, p)$ defined by these constants has bounded expansion.*

Proof. We show the two conditions of Proposition 7 are satisfied in Lemma 37 and Lemma 38, respectively. □

Lemma 37. *Let $c \geqslant 1$ be a constant, $g = 2\frac{\alpha+c}{\alpha-1}$ $g' = \binom{g}{2}$ and λ be a constant bigger than $\max\{2e^{g+2}\gamma^g g\beta,\ c\}$. For $G = \mathcal{G}(n, m, p)$ and for all $\varepsilon > 0$ it holds with probability $O(n^{-c})$ that*

$$\frac{1}{|V(G)|} \cdot \left| \left\{ v \in V(G) \colon \deg(v) \geqslant \frac{2\lambda g'}{\varepsilon} \right\} \right| \leqslant \varepsilon.$$

Proof. By Lemma 35 we can disregard all bipartite graphs that have an attribute of degree greater than g. Suppose that for some ε there exists a vertex set S of size greater than εn in which all vertices have degree at least $2\lambda g'/\varepsilon$. This implies that there exists a set F of edges of size at least $\frac{\varepsilon n}{2} \frac{2\lambda g'}{\varepsilon} = \lambda g' \cdot n$ whose members each have at least one endpoint in S. Further, since every attribute has degree at most g and thus it generates at most g' edges, there exists a set $F' \subseteq F$ such that

(i) $|F'| \geqslant |F|/g' = \lambda n$,

(ii) and every $e \in F'$ is generated by at least one attribute that generates no other edge in F'.

The existence of F' follows from a simple greedy procedure: Pick any edge from F and a corresponding attribute, then discard at most g' edges generated by this attribute. Repeat.

We now bound the probability that there exists such a set F': Since F' is generated by exactly $|F'| = \lambda n$ attributes, we can apply Theorem 13 to obtain the following bound:

$$\begin{aligned}\sum_{k=1}^{n} \binom{n}{k} \left(\frac{e^{g+1}\gamma^g g\beta \cdot k}{\lambda n}\right)^{\lambda n} \left(\frac{k}{n}\right)^{\lambda n} &\leqslant \sum_{k=1}^{n} \left(\frac{e^{g+1}\gamma^g g\beta}{\lambda}\right)^{\lambda n} \frac{n^k e^k}{k^k} \frac{k^{2\lambda n}}{n^{2\lambda n}} \\ &\leqslant \left(\frac{e^{g+2}\gamma^g g\beta}{\lambda}\right)^{\lambda n} \sum_{k=1}^{n} \left(\frac{k}{n}\right)^{2\lambda n - k}\end{aligned}$$

By the choice of λ, this expression is bounded by

$$\frac{1}{2^{\lambda n}} \sum_{k=1}^{n} \left(\frac{k}{n}\right)^{2\lambda n - k} \leqslant \frac{n}{2^{\lambda n}}$$

since every element of the sum is smaller than one and the statement follows. Note that $n/2^{\lambda n} < 1/n^c$ since $\lambda > c$, i.e., this probability converges faster than the one proven in Lemma 35. □

Lemma 38. *Let $c \geqslant 1$ be a constant, $g = 2\frac{\alpha+c}{\alpha-1}$, $g' = \binom{g}{2}$, ϕ be defined as in Theorem 12 and $\delta_r > (2r+1) \cdot \max\{e^{g+1}\gamma^g g g'\beta,\ (c+1)g'\}$. Then for every $r \in \mathbb{N}^+$, for every $0 < \varepsilon < e^{-2}$ and for every $H \subseteq G = G(n,m,p)$ with $|V(H)| < \varepsilon n$ it holds with probability $O(n^{-c})$ that $\widetilde{\nabla}_r(H) \geqslant \delta_r$.*

Proof. By Lemma 32 if G contains an r-shallow topological minor of density δ_r then for some $i \in \{0, \ldots, 2r\}$ there exists a stable i-subdivision of density $\delta_r/(2r+1)$. We can then bound the probability of a r-shallow topological minor by bounding the probability of a stable i-subdivision of density $\delta_r/(2r+1)$.

From Lemma 36 we know that the probability of an 0-shallow topological minor on k nails is bounded by

$$\binom{n}{k} \delta k \left(\frac{k}{n}\right)^{\frac{\delta k}{g'}}.$$

By Theorem 12, the density for an i-subdivision of density $\delta_r/(2r+1)$ for $i \in \{1, \ldots, 2r\}$ is bounded by

$$r\delta k \cdot \left(\frac{\phi}{n}\right)^{\frac{\alpha-1}{2}k}.$$

Taking the union bound of these two events gives us a total bound of

$$\binom{n}{k} \delta k \left(\frac{k}{n}\right)^{\frac{\delta k}{g'}} + (2r+1) r\delta k \cdot \left(\frac{\phi}{n}\right)^{\frac{\alpha-1}{2}k} \tag{19.5}$$

for the probability of a dense subgraph or subdivision on k vertices to appear. Taking the union bound over all k we obtain for the first summand that

$$\sum_{k=1}^{\varepsilon n} \binom{n}{k} \delta k \left(\frac{k}{n}\right)^{\frac{\delta k}{g'}} \leqslant \delta_r \sum_{k=1}^{\varepsilon n} \frac{n^k e^k}{k^k} \frac{k^{(c+1)k+1}}{n^{(c+1)k}}.$$

Since δ_r is a constant, it suffices that the term

$$\sum_{k=1}^{\varepsilon n} \frac{n^k e^k}{k^k} \frac{k^{(c+1)k+1}}{n^{(c+1)k}}$$

is in $O(n^{-c})$. We will show this is bounded by a geometric sum by considering the ratio of two consecutive summands:

$$\frac{e^{k+1}(k+1)^{c(k+1)+1}}{n^{c(k+1)}} \cdot \frac{n^{ck}}{e^k k^{ck+1}} = e\frac{(k(1+1/k))^{c(k+1)+1}}{n^c k^{ck+1}} \leqslant e^2 \frac{k^c}{n^c} \leqslant e^2 \varepsilon^c.$$

Since this is smaller than one when $\varepsilon < e^{-2}$ and $c \geqslant 1$, the summands decrease geometrically. Hence its largest element (i.e., the summand for $k = 1$) dominates the total value of the sum, more precisely, there exists a constant ξ (depending on α and c) such that

$$\sum_{k=1}^{\varepsilon n} \frac{e^k k^{ck+1}}{n^{ck}} \leqslant \xi \frac{e}{n^c} = O(n^{-c}). \tag{19.6}$$

We now turn to the second summand. It is easy to see by the same methods as before that this sum is also geometric for $n > \phi^{(\alpha+1)/2}$ and as such there exists a constant ξ' which multiplied with the first element bounds the sum. An r-shallow topological minor of density δ_r has at least $2\delta_r$ nails, thus we can assume $k \geqslant 2\delta_r$. Since $\delta_r > (c+1)g' \geqslant c/(\alpha-1)$, we have:

$$\sum_{k=2\delta_r}^{\varepsilon n} (2r+1)r\delta k \cdot \left(\frac{\phi}{n}\right)^{\frac{\alpha-1}{2}k} \leqslant \frac{\xi'(2r+1)\phi^{\delta_r}}{n^{(\alpha-1)\delta_r}} \leqslant \frac{\xi'(2r+1)\phi^{\delta_r}}{n^c} = O(n^{-c}). \tag{19.7}$$

Combining Equations 19.6 and 19.7, Equation 19.5 is bounded by $O(n^{-c})$. □

20 EXPERIMENTAL EVALUATION

Our theoretical results provide insight into asymptotic properties of the grad and degeneracy of random intersection graphs. To sharpen our understanding of how these statistics behave in realistic parameter ranges, we designed four experiments to relate our theoretical predictions to concrete measurements.

We used the NetworkX python package [114] to generate our random intersection graphs (using the `uniform_random_intersection_graph` method) and the SageMath software system [69] to compute the degeneracy [21, 202] and diameter [60, 59, 174, 224] of the generated graphs. The measurements of the p-centered coloring number (presented below) were executed using the implementation which is a part of CONCUSS [197]. In the first three experiments, we generated random intersection graphs using parameters $\alpha \in \{0.3, 0.5, 0.7, 0.9, 1.0, 1.2\}$ and fixed $\beta = \gamma = 1.2$. Each data point represents an average over 20 random instances of a given size n (increasing from a few thousand to several hundred thousand, with finer granularity at smaller sizes to capture boundary effects). The last experiment, which concerns the structural sparseness of the model $G(n, m, p)$ when $\alpha > 1$, fixes parameters $\alpha = 1.5$, $\beta = 0.1$ and $\gamma = 5$ (due to computational constraints) and averages over ten instances of each size.

Our first experiment is designed to estimate the constants involved in the asymptotic bounds provided by Theorem 11. To that end, we fit the three functions for the respective regimes of α by computing a multiplicative scaling τ using least-square fitting via the `scipy` [135] implementation of the Levenberg–Marquardt algorithm [163, 177].

Both the data and the fitted functions are plotted in Figure 20.1, the function parameters and scaling factors can be found in Table 20.1. Already for graphs of moderate size we see that the degeneracy closely follows the predicted functions. We further note that for the series $\alpha = 1.2$, the observed degeneracy is around 5, which is very far from the massive upper bound given by setting $r = 0$ in Lemma 38 (value not shown in plot). It would be interesting to see whether bounds with tighter constants can be obtained by different proof techniques. For the value $\alpha = 1.0$ we see that the asymptotic lower bound $\Omega(\log n / \log\log n)$ fits the observed degeneracy very well with only a small scaling factor of 1.57. We put forward the conjecture that the degeneracy actually follows $\Theta(\log n / \log\log n)$ in this regime. Finally, for $\alpha < 1$ we see some increase of the scaling factor τ as α tends to one. The lower bound $\gamma n^{(1-\alpha)/2}$ therefore seems to miss some slight dependency on α, but otherwise matches the degeneracy observed very well.

A second experiment measures the structural sparseness of $G(n, m, p)$ in the regime $\alpha > 1$. Since our bounds on the degeneracy—which can be understood as the most "local" grad $\widetilde{\nabla}_0$—are far away from what we observed in the first experiment, it is reasonably to presume that the bounds on higher grads are even worse. Since bounded

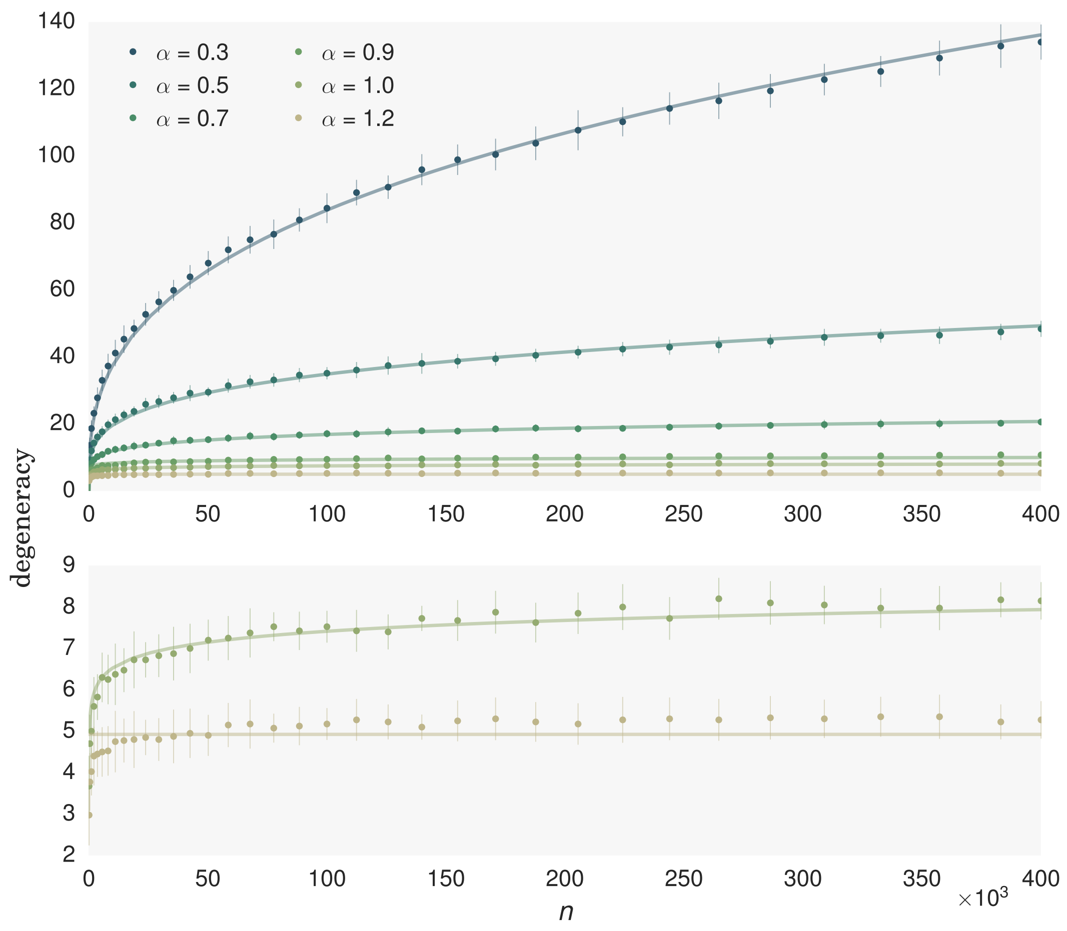

Figure 20.1: Degeneracy of $G(n, m, p)$ for different values of α and increasing n. The parameters $\beta = \gamma = 1.2$ were fixed; all data points are an average over 20 graphs. Error bars show one standard deviation. The lower figure contains the same plots for $\alpha \geqslant 1$ in a different scale. The continuous lines are functions listed in Table 20.1 fitted to the data.

Table 20.1: Functions corresponding to the degeneracy upper- and lower bounds from Theorem 11 fitted to the degeneracy data displayed in Figure 20.1. The coefficients τ were determined by least-square fitting.

α	Function	τ
0.3	$\tau \cdot 1.2n^{0.35}$	1.24
0.5	$\tau \cdot 1.2n^{0.25}$	1.63
0.7	$\tau \cdot 1.2n^{0.15}$	2.49
0.9	$\tau \cdot 1.2n^{0.05}$	4.34
1.0	$\tau \cdot \frac{\log n}{\log\log n}$	1.57
1.2	τ	4.92

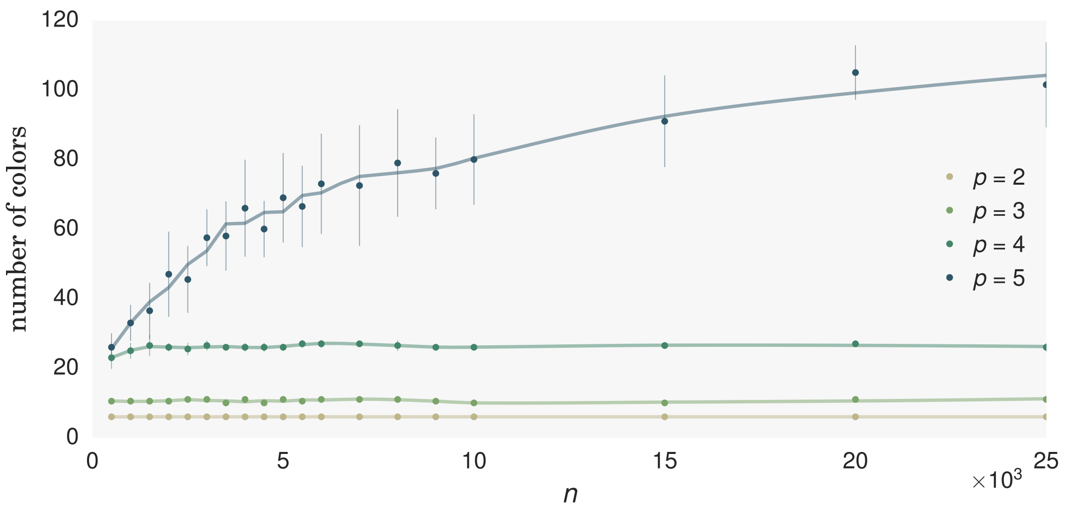

Figure 20.2: Median number of colors in a p-low treedepth coloring for $G(n, m, p)$ with parameters $\alpha = 1.5$, $\beta = 0.1$ and $\gamma = 5$ (taken over ten random instances). Error bars denote one standard deviation (for $p \leqslant 4$ hardly visible). Lines represent a smoothed versions of the series and are included as a visual guide.

expansion has large potential to be exploited algorithmically in practice, we want to obtain a better understanding of the orders of magnitudes involved.

The asymptotic bounds provided by Lemma 38 are incredibly pessimistic: For parameters $\alpha = 1.5$, $\gamma = 5$ and $\beta = 0.1$ (selected to be relatively realistic and enable easy generation) the bound on $\widetilde{\nabla}_r$ provided by this lemma is at least 10^{13} (independent of r) even if we only insist on an error probability of $O(n^{-1})$. Since all tools for classes of bounded expansion depend heavily on the behavior of the expansion function and the expansion function given by Nešetřil and Ossona de Mendez's framework [189] will depend on δ_r, this upper bound is not enough to show practical applicability. Our experiment provides empirical evidence that the upper bound is not tight, improving the prospects for these associated tools. Specifically, we calculate p-low treedepth colorings, which can be used to characterize classes of bounded expansion (see Proposition 6 in Section 3) and have immediate algorithmic applications.

We implemented a simple version of the linear time coloring algorithm and ran it on ten random intersection graphs for each ($n \in \{500, 1000, \ldots, 6000, 7000, \ldots, 10000, 15000, 20000, 25000\}$) with parameters $\alpha = 1.5$, $\gamma = 5$ and $\beta = 0.1$ for each $p \in \{2, 3, 4, 5\}$. Figure 20.2 shows the median number of colors used by the algorithm. Our theoretical results predict a horizontal asymptote for every p. We can see a surprisingly small bound for $p \in \{2, 3, 4\}$. Even for $p = 5$ the plot starts flattening within the experimental range. It should be noted that the colorings given by this simple approximation algorithm are very likely to be far from optimal (that is, they may use many unnecessary colors).

This result indicates that the graphs modeled by random intersection are amenable to algorithms based on low treedepth colorings (which usually perform dynamic programming computations that depend exponentially on the number of colors). Further, by the known relation between p-low treedepth colorings and the expansion function, this indicates this graphs have much more reasonable expansion bounds than Lemma 38 would suggest.

21
COUNTING GRAPHLETS AND SUBGRAPHS

The tool of choice for applying a counting algorithm designed for bounded-treedepth graphs to a class of bounded expansion is low treedepth colorings: to compute the frequency of a given pattern of size k, we compute a $(k+1)$-low treedepth coloring of the input graph in linear time as per Proposition 6. We can then enumerate all possible choices of $i < k$ colors and count the frequency of the pattern graph in the graph induced by those color classes. As this induced subgraph has bounded treedepth, we can focus on counting a fixed subgraph inside a target graph of treedepth at most k. We can then compute the frequency in the original graph using inclusion-exclusion on the color classes.

Central to the dynamic programming we will use to count isomorphisms is the following notion of a *k-pattern* which is very similar to the well-known notion of boundaried graphs. In the following we let $[i] = \{1, \ldots, i\}$ for any $i \geqslant 1$.

Definition 41 (k-pattern)**.** A *k-pattern* of a graph H is a triple $M = (W, X, \pi)$ where $X \subseteq W \subseteq V(H)$, $|X| \leqslant k$, such that $W \setminus X$ has no edge into $V(H) \setminus W$, and $\pi\colon X \to [k]$ is an injective function. We will call the set X the *boundary* of M. For a given k-pattern M we denote the underlying graph by $H[M] = H[W]$, the vertex set by $V(M) = W$, the boundary by $bd(M) = X$ and the mapping by π^M.

We denote by $\mathcal{P}_k(H)$ the set of all k-patterns of H. Note that every k-pattern (W, X, π) is also a $(k+1)$-pattern. In the following we denote by $|H| = |V(H)|$.

Lemma 39. *Let H be a graph. Then $|\mathcal{P}_k(H)| \leqslant 3^{|H|} \cdot k^{|H|}$.*

Proof. The vertices of H can be partitioned in $3^{|H|}$ possible ways into boundary vertices, pattern vertices and remainder. The number of ways an injective mapping for a boundary of size $b \leqslant |H|$ into $[k]$ can be chosen is bounded by $k^{|H|}$. In total the size of $\mathcal{P}_k(H)$ is always less than $3^{|H|} \cdot k^{|H|}$. $\square$

The following definition show how k-patterns will be used structurally, namely by gluing them together or by demoting a boundary-vertex to a simple vertex. These operations will later be used in a dynamic programming algorithm.

Definition 42 (k-pattern join)**.** Let H be a graph and let both $M_1 = (W_1, X_1, \pi_1)$ and $M_2 = (W_2, X_2, \pi_2)$ be k-patterns of H. Then the two patterns are *compatible* if $W_1 \cap W_2 = X_1 = X_2$ and for all $v \in X_1$ it holds that $\pi_1(v) = \pi_2(v)$. Their *join* is defined as the k-pattern $M_1 \oplus M_2 = (W_1 \cup W_2, X_1, \pi_1)$.

Definition 43 (k-pattern forget)**.** Let H be a graph, let $M = (W, X, \pi)$ be a k-pattern of H and $i \in [k]$. Then the *forget operation* is the k-pattern

$$M \ominus i = \begin{cases} (W, X \setminus \pi^{-1}(i), \pi|_{X \setminus \pi^{-1}(i)}) & \text{if } \pi^{-1}(i) \neq \emptyset \text{ and } N_H(\pi^{-1}(i)) \subseteq W \\ \perp & \text{if } \pi^{-1}(i) \neq \emptyset \text{ and } N_H(\pi^{-1}(i)) \not\subseteq W \\ (W, X, \pi) & \text{otherwise} \end{cases}$$

Structurally, the k-pattern's boundaries will represent vertices from the path of the root vertex to the currently considered vertex in the treedepth decomposition, while the remaining vertices of the pattern represent vertices somewhere below it. The following notation helps expressing these properties.

Definition 44 (Root path)**.** The *root path* of x is the unique path Q_x from the root r to x in T. We let $Q_x[i]$ denote the i^{th} vertex of the path (starting at the root), so that $Q_x[1] = r$ and $Q_x[\,|Q_x|\,] = x$, where $|Q_x|$ is the number of nodes on the path.

We can now state the main lemma. The proof contains the description of the dynamic programming which works bottom-up on the vertices of the given treedepth decomposition (i.e., starting at the leaves and working towards the root of the decomposition).

Lemma 40. *Let H be a fixed graph on h vertices. Given a graph G on n vertices and a treedepth decomposition T of height d, one can compute the number of isomorphisms from H to induced subgraphs of G in time $O(6^h \cdot d^h \cdot h^2 \cdot n)$ and space $O(3^h \cdot d^h \cdot hd \cdot \log n)$.*

Proof. We provide the following induction that easily lends itself to dynamic programming over T. Denote by $M_H = (V(H), \emptyset, \varepsilon)$ the trivial d-pattern of H, where $\varepsilon \colon \emptyset \to \emptyset$ denotes the null function. Consider a set of vertices $v_1, v_2, \dots, v_\ell \in G$ with a common parent v in T with respective subtrees T_{v_i} and root paths Q_{v_i} for $1 \leqslant i \leqslant \ell$. Note that the root paths $Q_{v_1}, \dots, Q_{v_\ell}$ all have the same length k and share the path Q_v as a common prefix.

Let M be a fixed k-pattern of H. We define the mapping $\psi_v^M \colon bd(M) \to V(Q_v)$ via $\psi_v^M(v') = Q_v[\pi^M(v')]$ for $v' \in bd(M)$, which takes the pattern's boundary and maps it to the vertices of the root-path. We denote by $f[v_1, \dots, v_\ell][M]$ the number of isomorphisms $\phi \colon V(M) \to V(G)$ such that the following properties hold:

1. $\phi|_{bd(M)} = \psi_v^M$
2. $\phi(V(M) \setminus bd(M)) \subseteq G[V(T_{v_1}) \cup \dots \cup V(T_{v_\ell})]$

In other words we charge subgraphs to patterns whose boundaries lie on the shared root-path Q_v such that the labeling of the boundary coincides with the numbering induced by Q_v while the rest of the pattern is contained entirely in the subtrees $T_{v_1}, \dots, T_{v_\ell}$. Note that v cannot be part of the boundary. If r is the root of the treedepth decomposition, $f[r][M_H]$ counts exactly the number of isomorphisms of H into subgraphs of G.

We will show now how we can compute $f[r][M_H]$ recursively. For a leaf $v \in T$ and a d-pattern $M_1 = (W_1, X_1, \pi_1) \in \mathcal{P}_d(H)$ we compute $f[v][M_1]$ as follows: Define the function $p_v(M)$ to be 1 if the function $\psi \colon V(M) \to V(Q_v)$ defined as $\psi(w) = Q_v[\pi_1[w]]$ is an isomorphism from $H[V(M)]$ to $G[\psi(V(M))]$ and 0 otherwise. In particular, $p_v(M)$ will be zero if $V(M) \neq bd(M)$ or $|V(M)| > |Q_v|$. Then for the leaf v we set $f[v][M_1] = p_v(M_1)$.

The following recursive definitions show how $f[\cdot][M_1]$ can be computed for all inner vertices of T. Let $v_1, \ldots, v_\ell$ be the children of an internal vertex v such that $f[v_i][M']$ are correctly set for all $v_i \in \{v_1, \ldots, v_\ell\}$ and $M \in \mathcal{P}_d(H)$. We define the following operations:

$$f[v_1, \ldots, v_{j-1}, v_j][M_1] = \sum_{M_2 \oplus M_3 = M_1} f[v_1, \ldots, v_{j-1}][M_2] \cdot f[v_j][M_3] \qquad \text{(join)}$$

$$f[v][M_1] = \sum_{M_2 \ominus |Q_v| = M_1} f[v_1, \ldots, v_\ell][M_2] \qquad \text{(forget)}$$

where $M_2, M_3 \in \mathcal{P}_d(H)$. It is clear then that we can iteratively, for increasing values of j, compute $f[v_1, \ldots, v_{j-1}, v_j][M_1]$ until $j = \ell$ and then compute the forget. Since we can start at the leaves, we can compute the value $f[r][M_H]$ this way.

We need to prove that the table f correctly reflects the number of isomorphisms to subgraphs satisfying Properties 1 and 2. We will prove this by induction.

Consider the *join*-case first: Fix a pattern $M_1 \in \mathcal{P}_d(H)$. By induction, the entries $f[v_1, \ldots, v_{j-1}][\cdot]$ and $f[v_j][\cdot]$ correspond to the number of isomorphisms to subgraphs that satisfy Properties 1 and 2 on these node sets, respectively. We will show that $f[v_1, \ldots, v_j][M_1]$ as defined gives the number of isomorphisms from $H[M_1]$ to subgraphs of G where $\phi_1|_{bd(M_1)} = \psi_v^{M_1}$ and $\phi_1(V(M_1) \setminus bd(M_1)) \subseteq G[V(T_{v_1} \cup \cdots \cup T_{v_j})]$.

Consider the set Φ_1 of all isomorphisms from $H[M_1]$ to subgraphs of G satisfying Properties 1 and 2 for the vertex tuple $v_1, \ldots, v_j$. For any vertex subset $R \subseteq V(M_1) \setminus bd(M_1)$, define the slice $\Phi_1(R) \subseteq \Phi_1$ as those isomorphisms ϕ that satisfy $\phi^{-1}(\phi(V(H)) \cap T_{v_j}) = R$. Let $L = (V(M_1) \setminus bd(M_1)) \setminus R$ and define the patterns $M_L = (L \cup bd(M_1), bd(M_1), \pi^{M_1})$ and $M_R = (R \cup bd(M_1), bd(M_1), \pi^{M_1})$. Then by induction $|\Phi_1(R)| = f[v_1, \ldots, v_{j-1}][M_L] \cdot f[v_j][M_R]$, since $M_1 = M_L \oplus M_R$ and clearly $M_L, M_R \in \mathcal{P}_d(H)$, the sum computes exactly $\sum_{R \subseteq V(M_1) \setminus bd(M_1)} |\phi_1(R)| = |\phi_1|$.

Next, consider the *forget*-case. Again, fix $M_1 \in \mathcal{P}_d(H)$ and let u be the parent of v in T. Let Φ_1 be the set of those isomorphisms ϕ_1 from $H[M_1]$ to subgraphs of G for which $\phi_1|_{bd(M_1)} = \psi_u^{M_1}$ and $\phi_1(V(M_1) \setminus bd(M_1)) \subseteq G[V(T_v)]$. We partition Φ_1 into $\Phi_1 = \Phi_{1,v} \cup \Phi_{1,\bar{v}}$ where $\Phi_{1,v}$ contains those isomorphisms ϕ for which $\phi^{-1}(v) \neq \emptyset$ and $\Phi_{1,\bar{v}}$ the rest. Since $|\Phi_{1,\bar{v}}| = f[v_1, \ldots, v_\ell][M_1]$ we focus on $\Phi_{1,v}$ in the following. For $w \in V(M_1) \setminus bd(M_1)$, define $\Phi_{1,v}(w)$ as the set of those isomorphisms ϕ for which $\phi(w) = v$. Clearly, $\{\Phi_{1,v}(w) \mid w \in V(M_1) \setminus bd(M_1)\}$ is a partition of $\Phi_{1,v}$. Define the pattern $M_w = (V(M_1), bd(M_1) \cup \{w\}, \pi_w^{M_1})$ where $\pi_w^{M_1}$ is π^{M_1} augmented with the value $\pi_w^{M_1}(v) = |Q_v|$. Note that by construction $M_1 = M_w \ominus |Q_v|$. By induction, $|\Phi_{1,v}(w)| = f[v_1, \ldots, v_\ell][M_w]$ and therefore

$$|\Phi_1| = |\Phi_{1,\bar{v}}| + \sum_{w \in V(M_1) \setminus bd(M_1)} |\Phi_{1,v}(w)| = \sum_{M_2 \ominus |Q_v|} f[v_1, \ldots, v_\ell][M_2]$$

It remains to be proven that this can be done in the claimed running time. Initialization of f for a leaf takes time $O(|\mathcal{P}_d(H)|h^2)$ since we need to test whether the function ψ defined above is an isomorphism for each pattern in $\mathcal{P}_d(H)$.

For the other vertices, a forget operation can be achieved in time $O(|\mathcal{P}_d(H)|)$ per vertex by enumerating all d-patterns, performing the forget operation and looking up the count of the resulting pattern in the previous table.

A join operation needs time $O(|\mathcal{P}_d(H)| \cdot h \cdot 2^h)$ per vertex, since for a given pattern M_1 those patterns M_2, M_3 with $M_1 = M_2 \oplus M_3$ are uniquely determined by partitions of the set $V(M_1) \setminus bd(M_1)$.

In total the running time of the whole algorithm is $O(|\mathcal{P}_d(H)| 2^h h^2 \cdot n)$ and thus by Lemma 39 $O(6^h \cdot d^h \cdot h^2 \cdot n)$. Note that we only have to keep at most $O(d)$ tables in memory, each of which contains the occurrence of up to $|P_d(H)|$ patterns stored in numbers up to n^h. Thus in total the space complexity is $O(|\mathcal{P}_d(H)| \cdot d \cdot \log(n^h)) = O(|\mathcal{P}_d(H)| \cdot hd \cdot \log n)$. □

To count the occurrences of H as an induced subgraph instead the number of subgraph isomorphisms, one can simply determine the number of automorphisms of H in time two to the power of $O(\sqrt{h \log h})$ [17, 180] and divide the total count by this value (since this preprocessing time is dominated by our running time we will not mention it in the following). Counting isomorphism to non-induced subgraphs can be done in the same time and space by changing the initialization on the leaves, such that it checks for an subgraph instead of an induced subgraph. Dividing again by the number of automorphisms gives the number of subgraphs. By allowing the mapping of the patterns to map several nodes to the same value, we can use them to represent homomorphisms. Testing the leaves accordingly, the same algorithm can be used to count the number of homomorphisms from H to subgraphs of G. By keeping all tables in memory, thus sacrificing the logarithmic space complexity, and using backtracking we can label every node with the number of times it appears as a certain vertex of H. From these observations and Lemma 39 we arrive at the following theorem:

Theorem 15. *Given a graph H on h vertices, a graph G on n vertices and a treedepth decomposition of G of height d, one can compute the number of isomorphisms from H to subgraphs of G, homomorphisms from H to subgraphs of G, or (induced) subgraphs of G isomorphic to H in time $O(6^h \cdot d^h \cdot h^2 \cdot n)$ and space $O(3^h \cdot d^h \cdot hd \cdot \log n)$.*

Note that for graphs of unbounded treedepth the running time of the algorithm degenerates to $O(6^h \cdot h^2 \cdot n^{h+1})$, which is comparable to the running time of $2^{O(\sqrt{h \log h})} \cdot n^h$ of the trivial counting algorithm.

By Proposition 6, we can immediately use this result to achieve the following theorem about graph classes of bounded expansion.

Theorem 16. *Given a graph H and a a graph G belonging to a class of bounded expansion, there exists an algorithm to count the appearances of H as a subgraph of G in time*

$$O\left(\binom{f(h)}{h} \cdot 6^h \cdot h^{h+2} \cdot n\right)$$

where f is a function depending only on the graph class.

This immediately extends to nowhere dense classes, which for any $\varepsilon > 0$ have low treedepth-colorings with at most n^ε colors (for sufficiently large graphs) [193]. Choosing the graphs large enough and setting $\varepsilon' = \varepsilon/h$, we can bound the term $\binom{f(h)}{h}$ by $n^{\varepsilon' \cdot h} = n^\varepsilon$.

Theorem 17. *Let $\mathcal{G}$ be a nowhere-dense class and let H be a graph. For every $\varepsilon > 0$ there exists $N_\varepsilon \in \mathbb{N}$, such that for any graph $G \in \mathcal{G}, |G| > N_\varepsilon$ there exists an algorithm to count the appearances of H as a subgraph of G in time*

$$O\left(6^h \cdot h^{h+2} n^{1+\varepsilon}\right).$$

22 CONCLUSION

In this part we have determined the conditions under which random intersection graphs exhibit a type of algorithmically useful structure. More specifically, we proved graphs in $G(n, m, p)$ are structurally sparse (have bounded expansion) precisely when the number of attributes in the associated bipartite graph grows faster than the number of nodes ($\alpha > 1$). Moreover, we showed that when the generated graphs are not structurally sparse, they fail to achieve even much weaker notions of sparsity (in fact, w.h.p. they contain large cliques). We furthermore showed how assuming a graph has bounded expansion is exploitable for motif counting and computing the graphlet degree distribution.

A question that naturally arises from these results is if structural sparsity should be an expected characteristic of practically relevant random graph models. Our contribution solidifies this idea and supports previous results for different random graph models [65, 207]. We further ask whether the grad is small enough to enable practical algorithmic application—our empirical evaluation using p-centered colorings of random intersection graphs with $\alpha > 1$ indicate the answer is affirmative.

Part V

TREEWIDTH FROM TREEDEPTH

23

STARTING FROM TREEDEPTH

In this part we will present a new idea to develop heuristics for treewidth, which is based on computing a treedepth decomposition of the graph to then manipulate it. This idea is related to the the notion of the elimination height of a chordal graph, which we discussed in Section 1. Here we will further draw a connection between a treedepth decomposition of a graph and an *elimination order* of a graph, which is the key concept in almost every treewidth heuristic. We will show how computing a good treedepth decomposition to then manipulate can lead to reasonably good heuristics for treewidth. In some cases, such a scheme even outperforms the thirteen other heuristics to which we compare our approach. We will not only present this idea as leading to further well-performing heuristic for treewidth, but also discuss how the general scheme on which it is based could be exploited in future work to implement meta-heuristics that apply different heuristics to different parts of the input graph.

In Section 1, while discussing elimination trees, we introduced the notion of the perfect elimination ordering of a chordal graph: Every chordal graph has at least one ordering of the nodes called a perfect elimination, such that for every node all neighbors that come later in the ordering form a clique. We also mentioned that computing the treewidth of a graph is equivalent to finding a triangulation with smallest clique size [12]. Almost every treewidth heuristic is based on attempting to find a so called *elimination order* of the graph, which is an order of the nodes which is assumed to be a perfect elimination of the optimal triangulation of the graph. The triangulation is then derived by adding all the missing edges from this order to be a perfect elimination. We will refer to the edges that are added as the *fill-in*.

The fundamental idea behind the heuristic is that, given a treedepth decomposition T of a graph, the maximal distance in T between any two nodes connected by an edge of the graph is an upper bound for the treewidth of the graph. We will call this distance the *stretch* of T. The heuristic thus computes a treedepth decomposition of the graph and then manipulates it in an attempt to minimize the stretch. Then an elimination order is derived by recursively taking leaves of the treedepth decomposition.

We will compute the treedepth decomposition in three ways: First by just taking a the tree with results from a depth first visit of all nodes in the graph, which is, as discussed in Section 3.1, a valid treedepth decomposition of the graph. This will cover the cases where the structure of the graph is rather simple. We also construct treedepth decompositions by recursively finding separators which either attempt to maximize the number of remaining components when removed, or minimize the size of the biggest remaining component. The separators are found by starting with a trivial separator and then greedily exchanging some nodes of the separator with neighbors of the node as long as it improves the measure. Finally, we create a treedepth decomposition by finding separators via eigenvectors [205]. The nodes of the treedepth decomposition are weighted by the stretch of its incident edges using a "spring-like" function. We then try to move the node on which the strongest resulting force acts via simple restructuring of the treedepth decomposition in an attempt to decrease the acting force. We do this for some small amount of time, since this process tends to quickly stop decreasing the stretch. The details of the heuristic will be provided in Section 24.

We will run these four heuristics and compare them to thirteen other heuristics on 371 graphs which have been used as test-beds for treewidth heuristics previously. When looking at the results in Section 25, we will see that in 6.5% of these graphs, one of these four heuristics achieves a better result than the other thirteen and that in 41.0% one of these achieves a result which equals or is better than the best result of the other thirteen heuristics.

Since there is a clear way to derive an elimination tree from an elimination order, we also let the second step of the heuristics, which attempts to minimize the stretch, run on treedepth decompositions generated from the elimination orders given by the other heuristics. It manages to improve 7.01% of these elimination orders beyond the previous best result and in 26.68% of the cases it takes a previously suboptimal one and derives a width equal or better than the previous best one between all heuristics.

24 HEURISTIC

The proposed heuristics are divided in two steps. The first one computes a treedepth decomposition. The second one takes this treedepth decomposition and manipulates it trying to create a treedepth decomposition whose corresponding elimination order has low treewidth.

STRETCH

We start by giving a formal definition of the stretch of a treedepth decomposition.

Definition 45 (Stretch)**.** Given a treedepth decomposition T of a graph G, the *stretch* of T is $\max_{uv \in E(G)} d_T(u,v)$. The stretch of an edge $uv \in E(G)$ is the distance of u and v in T.

As mentioned before, the second step of the heuristics will attempt to minimize the stretch of a given treedepth decomposition. The theoretical foundation for this is given by the following lemma.

Lemma 41. *Given a treedepth decomposition T of a graph G with stretch s it follows that* $\mathbf{tw}(G) \leqslant s$.

Proof. Let $\pi = v_1 \dots v_n$ be an elimination order of G achieved by recursively removing leaves of T. We can compute the fill edges of this elimination order by recursively taking the nodes of G in the order given by π, making the neighborhood of the currently selected node v_i a clique and deleting v_i. Notice that the cliques we create in this manner will be the maximal cliques of the triangulation. Thus the size of the biggest neighborhood of a node v_i during this elimination process will be the treewidth given by the elimination order π. The theorem follows from the fact that the stretch in T of any edge added by this process cannot be greater than the stretch of T and that the maximal stretch of edges incident to a node is an upper bound on the size of its neighborhood. □

The treedepth decomposition of minimum depth and the one of minimum stretch can look very different. A simple example would be a graph which is just a path. The treedepth decomposition with the lowest depth has depth and stretch $\log n$. The one with the lowest stretch is the path itself, having depth n but stretch 1, which is precisely its treewidth.

COMPUTING A TREEDEPTH DECOMPOSITION

Tobias Oelschlägel implemented as part of his bachelor thesis four different ways to compute a treedepth decomposition. The simplest one of these works by taking the tree given by a depth first search of the graph as the decomposition. This is valid treedepth decomposition as mentioned in Section 3.1. This cannot not be expected to compute a treedepth decomposition of low depth in general, but, as we will see later, it helps exploiting the structure of the graph when it has low treewidth.

The other three ways are all based on recursively finding minimal separators of the graph.

Definition 46 (Minimal separator)**.** A set S is a *separator* of G if there exist two nodes $u, v \notin S$ which lie in different components of $G[V(G) \setminus S]$. A set S' is a *minimal separator* of G if it is a separator and no proper subset of it is a separator.

In Section 1 we already discussed how a natural way of interpreting a treedepth decomposition is as a process of iteratively removing separators. When a separator is removed it becomes a path of nodes in the decomposition, such that all nodes except the deepest one have only one child. We can actually assume that all these separators are minimal: Let the *separator path* of a treedepth decomposition T be the nodes in a path from a root of T to the nearest node u with degree greater than two or $V(G)$ if no such node u exists. Note u can be the root itself. The *separator set* S of T is then the set of sets of nodes of T we get by recursively adding all separator paths of T to S. It is easy to see that given just the separator set S and G, we could reconstruct T. We now state a result by Manne rephrased for our current context.

Proposition 8 ([175])**.** *Let G be a graph with treedepth d. There exists a treedepth decomposition T of G of depth d such that every set A in the separator set of T is either a minimal separator of G or $G[A]$ does not have any separators.*

It follows that we can attempt to construct a treedepth decomposition of minimal depth by finding the correct minimal separators. Three ways to find separators were implemented, two based on enumerating separators, the third one based on finding separators via eigenvectors.

It is possible to enumerate all separators of a graph using $O(n^3)$ operations per separator [25]. The basis of this algorithm is the following proposition.

Proposition 9 ([25])**.** *If $S \subseteq V(G)$ is a minimal separator of a graph G and $x \in S$, then $N(C)$ is also a minimal separator for each component C of $G[V(G) \setminus (S \cup N(x))]$.*

We start the process by finding a *close separator*.

Definition 47 (Close separator)**.** A minimal separator S of graph G is called *close* to vertex x if $S \subseteq N_G(x)$.

A separator S close to x can be found easily by computing the graph $G' = G[V(G) \setminus (\{x\} \cup N(x))]$. Then, for each component of G', the set $N(C)$ is a minimal separator close to x. Since most graphs will have too many separators to enumerate them all, the enumeration algorithm was transformed into a kind of local search which greedily attempts to minimize some function over the separator: Let $N_S(G)$ be the set of minimal separators that can be constructed from a minimal separator S by applying Proposition 9. First the algorithm finds some minimal separator S which is close to some node. Then it enumerates every element of $N_S(G)$, takes the set $S' \in N_S(G)$ which minimizes a given function c and repeats this process on S' as long as $c(S') < c(S)$. The following functions were used in the experiments:

max	$c(S) =$ size of the greatest component of $G[V(G) \setminus S]$
num	$c(S) = \lvert V(G) \rvert -$ number of components of $G[V(G) \setminus S]$

Finally the last version of the heuristic finds separators using the method proposed by Pothen, Simon and Liu [205]:

1. Compute the second eigenvalue λ_2 of the *laplacian matrix* corresponding to G (see Chung [57]) and its corresponding eigenvector $\overline{y}$.
2. Partition the graph into $A = \{v_i \in V(G) \mid \overline{y}_i > 0\}$ and $B = V(G) \setminus A$.
3. Compute a minimal separator S by finding a minimum vertex cover of the graph induced on the edges incident to one node in A and one node in B. This can be done in polynomial time since this graph is bipartite [222].

Improving the treedepth decomposition

The methods discussed above to compute a treedepth decomposition do not attempt to minimize the stretch directly. The ones that are based in finding separators actually attempt to minimize the height of the decomposition. We will describe now a way to try to minimize the stretch of a given treedepth decomposition.

We start by calculating the acting force on every node v, choosing the force induced by an edge to grow quadratically with its stretch.

$$f_\Delta(v) = \sum_{\substack{uv \in E(G) \\ h_T(u) > h_T(v)}} (h_T(u) - h_T(v))^2 - \sum_{\substack{uv \in E(G) \\ h_T(u) < h_T(v)}} (h_T(v) - h_T(u))^2$$

Over all nodes incident to an edge with maximum stretch we choose the one with the greatest acting force. Let this node be u which is incident to an edge uv. If the acting force is positive we want to move the node up, down otherwise. To move the node u down towards v we will actually move the nodes between them up, which also has the effect of decreasing the stretch of the edge. Thus we actually only need to know how to move nodes upward for both operations. We do this by exchanging the position of the node x to be moved up with its parent p and then changing the parent of all children of x to be p. We then make the treedepth decomposition nice (see Definition 18) since this can only decrease the stretch of edges. This can be done thanks to a union-find structure in $O(n + m \cdot \alpha(m))$ amortized time, where α is the inverse of the Ackermann function [93, 98, 198]. All other operations described can be implemented in linear time. This operation is repeated until some given time limit is reached.

25 EXPERIMENTS

We compare our heuristics against thirteen other heuristics which are part of the INDDGO software package [16, 108]. A short description of these heuristics follows.

mind	*Min-degree* generates an elimination ordering by always choosing the next node to be one of minimum degree in the remaining graph [100].
mult	*Multiple min-degree* generates an elimination ordering by always choosing the next node to be one of minimum degree in the remaining graph and finding a set of nodes which can be safely eliminated simultaneously [166].
amd	*Approximate minimum degree* generates an elimination ordering by using approximate bounds for the minimum degree instead of the exact bounds, for the sake of efficiency [11].
minf	*Min-fill* generates an elimination ordering by iteratively choosing a node whose elimination introduces the least number of edges [18].
beta	In the *beta* heuristic all nodes with minimum fill-in are added to the ordering simultaneously at every step [16, 108].
bmf	The *batched min-fill* heuristic generates an elimination ordering by attempting to find a set of nodes which have together a small fill-in [16, 108].
mmd	The *minimum maximum degree* heuristic generates the elimination ordering by choosing nodes such that the maximum degree of the triangulated graph remains as small as possible after every step [16, 108].
lexm	*LEX-M* is a heuristic derived from the lexicographic breadth-first search (LEX-BFS) algorithm, which can recognize chordal graphs efficiently [213].
mcs	A heuristic derived from *maximum cardinality search*, which is itself an improvement of LEX-BFS for faster chordal graph recognition [24].
mcsm	*MCS-M* is another heuristic derived from maximum cardinality search [24].
metm	The METIS implementation of the multiple min-degree heuristic [140, 139].
metn	Uses the METIS package to compute an elimination ordering via *nested dissection* [140, 139].
parm	Uses the ParMETIS package to compute an elimination ordering via nested dissection, with an implementation that can run in parallel [140, 139].

The nested dissection algorithm mentioned for metn and parm also works by constructing a treedepth decomposition. The key difference between these heuristics and the ones proposed here is that these implementation attempt to minimize the fill produced by the elimination order from the treedepth decomposition *while* constructing it. The heuristics presented here start by constructing a treedepth decomposition without regard for the fill they would produce. In other words, the metn and parm heuristics are trying to find a good elimination order directly, while the new heuristics start by first representing the structure of the graph by a treedepth decomposition and only then trying to find a good elimination order.

To test the heuristic we chose the dimacs data set for graph coloring [1, 134], graphs from Bodlaender's `libtw` library [229] and the graphs from the PACE16 challenge [2]. These sets have all been previously used to test treewidth heuristics [2, 152, 229]. Some graphs are repeated between data sets. We considered two graphs to be the same graph if it had the same name (when cleaned up from prefixes like "dimacs_") and had the same number of nodes and edges.

In 24/371 (6.5%) instances one of the treedepth based heuristics is better than all other heuristics and for 152/371 (41.0%) of them the result is better or equal. For statistics divided by heuristic refer to Table 25.1. For a complete listing of all results see Table B.1 in Section B of the Appendix. As is to be expected, starting with a dfs gives an optimal results on all graphs which are trees, i.e. graphs of treewidth one. It is also rather effective on graphs of low treewidth. Unexpectedly, it provides the best result for some graphs that do not seem to have low treewidth, such as some of the "le450", "mulsol" and "queen" graphs; and the "Cosette", "HallJanko" and "Heawood" graphs. Clearly, the worst performing version of the heuristic is the one based on finding separators via eigenvectors. It only manages to achieve the best result over all heuristics and beat all other treedepth based heuristic once (the "Markstroem" graph). The other two heuristics sometimes achieve rather similar results most of the times, but are also often quite far apart. How we compute the treedepth decomposition has for many instances a big influence on the result. This is not surprising, since the improvement step does not restructure the treedepth decomposition too heavily.

As can be seen by the time statistics in Table 25.1, finding a treedepth via separators is quite slow. Nevertheless, the metn heuristic is also based on finding separators and it is the fastest of them all, so it should be possible to implement this more efficiently. The improvement step was run for three seconds on all instances except the ones with more than 500 nodes, for which it was run for 30 seconds. In almost every case the improvements stopped after less than five seconds.

As mentioned before there is a sensible way to derive a treedepth decomposition from an elimination order: Start taking nodes from the end of the elimination order until the nodes taken become a separator. Set this set as a path starting from the root in the decomposition, then recurse into the remaining components. Thus we can take

Heuristic	Best	Worst	Avg. rank	Worst time	Avg. time
mind	58.49%	3.23%	3.71	50.11	0.47
mult	58.49%	2.43%	3.58	65.30	0.51
amd	44.47%	3.23%	5.99	1.69	0.05
minf	72.51%	4.04%	2.35	> 300	2.79
beta	22.91%	29.38%	10.51	9.39	0.11
bmf	72.78%	4.04%	2.37	> 300	3.57
mmd	59.30%	2.96%	3.81	> 300	4.76
lexm	29.92%	14.56%	8.82	> 300	1.02
mcs	19.41%	24.80%	11.03	> 300	0.41
mcsm	28.03%	21.83%	9.30	> 300	0.66
metm	47.71%	5.66%	4.49	1.86	0.05
metn	26.15%	11.59%	7.38	1.85	0.05
parm	23.72%	11.59%	7.49	2.03	0.05
dfs	21.02%	13.75%	9.18	75.60	2.60
max	17.25%	18.06%	9.77	> 300	1.69
num	23.99%	8.09%	7.32	> 300	1.67
ev	7.55%	32.35%	12.52	47.32	0.89
td-all	40.97%	3.23%	4.98	—	—

Table 25.1: Overall results of all heuristics. The "td-all" row represents taking the best result over all treedepth based heuristics. Percentages are over all instances. All times are given in seconds. The average time was only computed over the instances for which the heuristics terminated in less than 5 minutes. For a complete table of all results see Table B.1 in Section B of the Appendix.

the results of the INDDGO heuristics, compute a treedepth decomposition for each one and run the improvement step on it.

In 26/371 (7.01%) instances, starting from the elimination tree given by the elimination order of one of the INDDGO heuristics, one of the treedepth based heuristics manages to get a better result than all other heuristics and for 99/371 (26.68%) of them an equally good result is achieved by improving a previously non-optimal one. For a complete listing of all these results see Table B.2 in Section B of the appendix.

26 CONCLUSION

We have seen that we can derive a competitive heuristic for treewidth by starting with the computation of a treedepth decomposition which is then manipulated to minimize its stretch. We implemented simple algorithms to both construct treedepth decompositions in different ways and to minimize the stretch. In our experimental results we have seen that despite the simplicity of the ideas behind the implementations, the results are good. What we have presented here is furthermore quite a flexible idea. Further work could attempt to improve how the treedepth decomposition is constructed and how it is manipulated.

A lot of effort has been put into figuring out how to find small separators efficiently and in parallel for the purpose of computing elimination trees for Cholesky factorization [55, 119, 140, 141, 161]. In this context though, the algorithms do not try to decrease the height of the treedepth decomposition but immediately try to find one whose corresponding elimination order has a small fill-in. It should be nevertheless possible to adapt these techniques to find treedepth decomposition of low height. Since many of these programs are open-source, it might even be possible to start with one of these implementations.

The manipulation presented here, making the treedepth decomposition nice after pushing nodes upwards to decrease distances, is not very involved. There are some known non-trivial manipulations of treedepth decompositions [113, 167, 169, 176] from which it might be possible to derive some further manipulations that decrease the stretch of a treedepth decomposition.

Finally, reordering or manipulating the treedepth decomposition is not the only way to get a better treewidth. It would be possible to throw away a part of the decomposition and compute a new treedepth decomposition for this part of the graph. This can be done also by using heuristics which compute an elimination order and then taking the corresponding treedepth decomposition. This suggests a straightforward way of using different heuristics on different parts of the graph.

Part VI

CONCLUSION

27 CONSIDER TREEDEPTH

At the beginning of this thesis we pointed out that treedepth or equivalent concepts have been (re)discovered again and again in different contexts. We also pointed out how in some cases it is the right tool to characterize some fundamental dichotomies. If one thinks of it as a measure of how easy it is to decompose a graph via separators, it might not seem too surprising that this turns out to be an important property for certain analyses.

We have presented here the asymptotically fastest exact parameterized algorithm to compute the treedepth of a graph to date, with a running time of $2^{O(d^2)} \cdot n$. An obvious open question is if a faster algorithm exists, especially if there is a single-exponential algorithm. If such an algorithm would use the same basic approach of the one presented here, i.e. doing dynamic programming on a tree decomposition, we would seemingly need to figure out how to compute an exact solution without our tables consisting of partially labeled trees. Recent work has parameterized calculating the treedepth of a graph by the vertex cover number [151], which allowed for a polynomial kernel. Maybe further sensible parameter besides the natural one exist.

A fundamental part of achieving this result was using a single-exponential linear-time fpt constant factor approximation for treewidth [38]. No such approximation is yet known for treedepth. It seems like the approach for treedepth would have to be fundamentally different than the approach for treewidth, since the treewidth approximation heavily relies on the last property of treewidth as an S-function (See Section 1 Definition 3), which as mentioned does not hold for treedepth.

In the next part, we investigated the treewidth and treedepth with relation to dynamic programming and space consumption. We proved that the space consumption of the current best algorithms for 3-Coloring, Vertex Cover and Dominating Set parameterized by treewidth cannot be beaten by any standard dynamic programming algorithm, by which we mean an algorithm whose running time does not depend on any other property of the tree decomposition besides its width and size, is not allowed to manipulate the provided tree decomposition and can only read every bag once in the normal order. As mentioned in Part iii, there exist very few algorithms exploiting tree decompositions that do any of these things and to the best of our knowledge most of these exceptions are for problems which are (assumed to be) not NP-hard. It might be nevertheless interesting to strengthen these results such that they apply for algorithms that are allowed to compute their own tree decomposition or manipulate a decomposition given as input. A way to achieve this would be to develop gadgets that allow the construction of Myhill–Nerode families such that the treedepth/pathwidth/treewidth decomposition of the elements of the family is basically unique.

We propose that investigating algorithmic paradigms in general is interesting and might be a fruitful endeavor. Several other abstractions of common algorithm design pattern have been proposed in the past, including for dynamic programming and branching [7, 118]. More recently Drucker, Nederlof and Santhanam showed among other things that INDEPENDENT SET is unlikely to have a fast fpt branching algorithm on graphs of bounded pathwidth [73].

A fundamental property of treedepth seems to be to permit algorithms with low space consumption. This is not only indicated by our results but also by the previously mentioned work by Pilipczuk and Wrochna [201]. We also mentioned that the techniques used to branch on a treedepth decomposition for DOMINATING SET can be extended [199] to the framework of Telle and Proskurowski for graph partitioning problems [225]. As a consequence of the work of Lampis this cannot be extended to MSO [156]. Is there a better way to characterize which problems are solvable with space polynomial in the treedepth and logarithmic in the input size?

An open question proposed by Michał Pilipczuk during GROW 2015 that, to the best of our knowledge, has not been resolved yet is if DOMINATING SET can be solved in time $(3-\varepsilon)^d \cdot \mathrm{poly}(n)$ for an $\varepsilon > 0$ or if this would contradict the (strong) exponential time hypothesis. By our lower bounds, if such an algorithm exists it cannot be a straightforward dynamic programming algorithm.

We have also connected treedepth via bounded expansion to random graph models designed to mimic real-world networks. We then provided an algorithm for motif counting which exploits low-treedepth colorings. It would be helpful to develop further algorithms for problems arising from practical applications for complex networks. For an in-depth elucidation of this program see the conclusion to Reidl's thesis [207].

We presented a treewidth heuristic which starts by attempting to compute a treedepth decomposition of low height or by taking a depth first search tree as a treedepth decomposition. It then takes the treedepth decomposition and tries to minimize its stretch, i.e. the maximal distance in the decomposition between nodes connected by an edge of the graph. In Section 26 we proposed several ways in which this heuristic could be improved and used to derive a meta-heuristic that applies different heuristics for different parts of the graph.

That this heuristic works as well as it does might be an indication that a treedepth decomposition says something fundamental about the graph. A decomposition of the graph might be of use as a preprocessing step outside of the framework of parameterized complexity, i.e. it might be possible to improve the running time of some processes by first representing the graph as a treedepth decomposition. In the field of routing it is common to build hierarchies of separators to improve the running time of shortest path searches [120, 231]. It could be argued that by doing this they are constructing something resembling a treedepth decomposition.

Recently there has been an increased interest in parameterizing by structural parameters problems which are in P [3, 22, 89, 92, 103, 181]. For an overview of previous

Graph	Nodes	Edges	**td**
karate	34	78	8
dolphins	62	159	20
lesmiserables	77	254	15
polbooks	105	441	28
word_adjacencies	112	425	40
football	115	613	67
airlines	235	1297	56
sp_data_school_day_2	238	5539	162
celegans	306	2148	118
hex	331	930	75
codeminer	724	1017	27
cpan-authors	840	2222	45
diseasome	1419	2738	23
polblogs	1490	16718	433
netscience	1589	2742	22
yeast	2361	7182	343
cpan-distributions	2724	5018	58
twittercrawl	3656	154824	1564
power	4941	6594	112
as20000102	6474	13895	173
hep-th	8361	15751	736
p2p-Gnutella04	10876	39994	3377
cond-mat	16726	47594	1696
CA-CondMat	23133	93497	3469

Table 27.1: Upper bounds on the treedepth of some real-world networks. For an explanation of the networks please refer to previous work [65].

results in this direction please refer to Fomin *et al.* [92]. We can see by the very preliminary results in Table 27.1 that the treedepth of real-world networks, while being too big for an exponential dependency might be small enough in sufficient cases for a polynomial one. Since it could be argued that treedepth captures something more fundamental about the graph than other measures it might be useful as a parameter for polynomial-time algorithms.

Furthermore, in the literature about elimination trees, elimination height and Cholesky factorization the elimination tree is not only measured by its depth, but also by how small its fill-in is. Since, as we can see by the results in Table 25.1 of Section 25, heuristically minimizing the fill-in is also a good heuristic to find a tree decomposition with low width, this could be seen as attempting to find a balance between the depth of the elimination tree and the treewidth give by the elimination tree's elimination order. We have furthermore presented an algorithm in Section 15.1 that interleaves branching and dynamic programming. It is not difficult to see that the dynamic part of the algorithm could be parameterized by the stretch of the treedepth decomposition instead of its depth. To minimize the stretch in our heuristic we might increase the height of the treedepth decomposition. It is nevertheless so that the stretch is close in many instances to the treewidth given the elimination order of the treedepth decomposition. It might be sensible to design algorithms that work on a treedepth decompositions parameterized by both its depth and stretch. Recently, Roden introduced the concept of *spanheight* [211]. Taken as a parameter, it is equivalent to parameterizing just by the stretch of a treedepth decomposition.

A further avenue of research are which problems that are hard when parameterized by treewidth admit fpt algorithms when parameterized by treedepth. We already mentioned how this proved to be possible for the Mixed Chinese Postman Problem and the Firefighter Problem. It has been stated as an open question whether the Minimum Shared Edges Problem is fpt parameterized by treedepth [90]. Other such candidate problems are problems that are NP-hard on paths, such as Rainbow Matching or MinCC Graph Motif, problems that are NP-hard on trees such as Bandwidth, Empire Coloring, The Traveling Repairman Problem, Achromatic Number, Integral k-Multicommodity flow, Capacitated Vehicle Routing, Minimum Latency, Call scheduling, and Connected Motifs in Vertex-Colored Graphs; and problems that are W[1]-hard when parameterized by treewidth, such as k-Capacitated Dominating Set, $[\sigma, \rho]$-Dominating Set, Equitable Coloring, General Factor, Minimum Maximum Outdegree, List Hamilton Path and Bounded-Degree Vertex Deletion. All these problems would be candidates to investigate whether treedepth is a suitable parameter. It should be noted that some problems such as List Coloring and Precoloring Extension remain W[1]-hard when parameterized by treedepth [49]. It might also be of interest to check if other reconfiguration problems besides the one investigated by Wrochna [233] are fpt when parameterized by treedepth. Double- and triple-exponential lower bounds have been proven for certain choosability problems when parameterized by treewidth [178]. Do these bounds persist when parameterizing by treedepth?

In Cholesky factorization the possibility of parallelization is one of the main uses of a treedepth decomposition. Nevertheless, we have not touched the topic of parallelization in this work, nor have we discussed any work on it besides Cholesky factorization. Recent work by Bannach and Tantau has investigated the parallelization of MSO on treewidth and treedepth via circuit complexity [19]. Further work into treedepth and parallelization might be of interest.

In the context of studying the isomorphism problem the concept of *generalized treedepth* has been proposed [46]. This characterization works by relaxing the cops-and-robbers games that is equivalent to treedepth. Bulian points out that treedepth and this generalization of treedepth are both special cases of the concept of *elimination distance to class $\mathcal{C}$* which he introduces [49]. In this characterization the class for treedepth is the edgeless graphs and for generalized treedepth the class contains graphs of degree at most two. A relaxation of vertex ranking, which only forces paths of length at most ℓ to contain a node of higher rank (cf. Definition 5), was recently introduced [138]. A restriction of treedepth called *starwidth* was also recently proposed [230]. How do this generalizations/restrictions of treedepth fit into the landscape?

Previous work has computed tree decompositions of a real-world networks not to necessarily exploit them algorithmically, but because they provided information about the deep structure of the network [4]. Since a treedepth decomposition of a network might be understood to represent a certain hierarchy of the network one might wonder if it works as a centrality measure. An indication that this might work out is that centrality measures are sometimes used to find good separators [120, 231].

Part VII

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Graph Coloring Instances. `http://mat.gsia.cmu.edu/COLOR/instances.html`. Accessed: 2017-05-10.

[2] PACE 2016 — Track A: Tree Width — The Parameterized Algorithms and Computational Experiments Challenge. `https://pacechallenge.wordpress.com/pace-2016/track-a-treewidth/`. Accessed: 2017-05-10.

[3] A. Abboud, V. V. Williams, and J. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete Algorithms*, pages 377–391, 2016.

[4] A. B. Adcock, B. D. Sullivan, and M. W. Mahoney. Tree decompositions and social graphs. *Internet Mathematics*, 12(5):315–361, 2016.

[5] A. Adiga, R. Chitnis, and S. Saurabh. Parameterized algorithms for boxicity. *Algorithms and Computation*, pages 366–377, 2010.

[6] I. Albert and R. Albert. Conserved network motifs allow protein-protein interaction prediction. *Bioinformatics*, 20(18):3346–3352, 2004.

[7] M. Alekhnovich, A. Borodin, J. Buresh-Oppenheim, R. Impagliazzo, A. Magen, and T. Pitassi. Toward a model for backtracking and dynamic programming. *computational complexity*, 20(4):679–740, 2011.

[8] N. Alon and S. Gutner. Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. *Algorithmica*, 54(4):544, 2009.

[9] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. *k*-core decomposition of internet graphs: hierarchies, self-similarity and measurement biases. *arXiv e-prints*, 2005, arXiv:cs/0511007.

[10] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. Large scale networks fingerprinting and visualization using the *k*-core decomposition. *Advances in neural information processing systems*, 18:41, 2006.

[11] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.

[12] E. Amir. Efficient approximation for triangulation of minimum treewidth. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 7–15, 2001.

[13] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

[14] B. Aspvall and P. Heggernes. Finding minimum height elimination trees for interval graphs in polynomial time. *BIT Numerical Mathematics*, 34(4):484–509, 1994.

[15] L. F. Avila, A. Garcıa, M. J. Serna, and D. M. Thilikos. Parameterized problems in bioinformatics. Technical report, Departament de Lenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 2006.

[16] B. D. Sullivan et al. Integrated network decompositions and dynamic programming for graph optimization (INDDGO). `http://github.com/bdsullivan/inddgo`.

[17] L. Babai, W. M. Kantor, and E. M. Luks. Computational complexity and the classification of finite simple groups. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*, pages 162–171, 1983.

[18] E. H. Bachoore and H. L. Bodlaender. New upper bound heuristics for treewidth. In *Experimental and Efficient Algorithms: 4th International Workshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005. Proceedings*, pages 216–227, 2005.

[19] M. Bannach and T. Tantau. Parallel multivariate meta-theorems. In *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63, 2017.

[20] M. J. Bannister, S. Cabello, and D. Eppstein. Parameterized complexity of 1-planarity. In *Algorithms and Data Structures: 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings*, pages 97–108, 2013.

[21] V. Batagelj and M. Zaversnik. An $O(m)$ algorithm for cores decomposition of networks. *arXiv e-prints*, 2003, arXiv:cs/0310049.

[22] M. Bentert, T. Fluschnik, A. Nichterlein, and R. Niedermeier. Parameterized aspects of triangle enumeration. *arXiv e-prints*, 2017, arXiv:1702.06548.

[23] M. W. Bern, E. L. Lawler, and A. L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *Journal of Algorithms*, 8(2):216–235, 1987.

[24] A. Berry, J. R. Blair, P. Heggernes, and B. W. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39(4):287–298, 2004.

[25] A. Berry, J.-P. Bordat, and O. Cogis. Generating all the minimal separators of a graph. *International Journal of Foundations of Computer Science*, 11(03):397–403, 2000.

[26] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, Inc., 1972.

[27] U. Bertele and F. Brioschi. On non-serial dynamic programming. *Journal of Combinatorial Theory, Series A*, 14(2):137–148, 1973.

[28] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets möbius: Fast subset convolution. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 67–74, 2007.

[29] J. R. S. Blair and B. Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation*, pages 1–29. 1993.

[30] M. Bloznelis. Degree and clustering coefficient in sparse random intersection graphs. *Annals of Applied Probability*, 23:1254–1289, 2013.

[31] M. Bloznelis, J. Jaworski, and V. Kurauskas. Assortativity and clustering of sparse random intersection graphs. *Electronic Journal of Probability*, 18:1–24, 2013.

[32] M. Bloznelis and V. Kurauskas. Large cliques in sparse random intersection graphs. *arXiv e-prints*, 2013, arXiv:1302.4627.

[33] H. L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *Automata, Languages and Programming: 15th International Colloquium Tampere, Finland, July 11–15, 1988 Proceedings*, pages 105–118, 1988.

[34] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[35] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. *Mathematical Foundations of Computer Science 1997*, pages 19–36, 1997.

[36] H. L. Bodlaender. Fixed-parameter tractability of treewidth and pathwidth. In *The Multivariate Algorithmic Revolution and Beyond*, pages 196–227, 2012.

[37] H. L. Bodlaender, J. S. Deogun, K. Jansen, T. Kloks, D. Kratsch, H. Müller, and Z. Tuza. Rankings of graphs. *SIAM Journal of Discrete Mathematics*, 11(1):168–181, 1998.

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

[39] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995.

[40] H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996.

[41] H. L. Bodlaender and D. Kratsch. Personal communication, 2014.

[42] R. Borie, R. Parker, and C. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(1-6):555–581, 1992.

[43] R. Borie, R. Parker, and C. Tovey. Solving problems on recursively constructed graphs. *ACM Computing Surveys*, 41(1):4, 2008.

[44] G. Borradaile and H. Le. Optimal dynamic program for r-domination problems over tree decompositions. *arXiv e-prints*, 2015, arXiv:1502.00716.

[45] M. Bougeret and I. Sau. How much does a treedepth modulator help to obtain polynomial kernels beyond sparse graphs? *arXiv e-prints*, 2016, arXiv:1609.08095.

[46] A. Bouland, A. Dawar, and E. Kopczyński. On tractable parameterizations of graph isomorphism. In *Parameterized and Exact Computation: 7th International Symposium, IPEC 2012, Ljubljana, Slovenia, September 12-14, 2012. Proceedings*, pages 218–230, 2012.

[47] F. Branin. A sparse matrix modification of Kron's method of piecewise analysis. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 383–386, 1975.

[48] J. Bu, C. Chen, X. Liu, M. Song, L. Zhang, and Q. Zhao. Probabilistic graphlet transfer for photo cropping. *Transactions on Image Processing*, 22(2):802–815, 2013.

[49] J. Bulian. Parameterized complexity of distances to sparse graph classes. Technical report, Computer Laboratory, University of Cambridge, 2017.

[50] J. Buresh-Oppenheim, S. Davis, and R. Impagliazzo. A stronger model of dynamic programming algorithms. *Algorithmica*, 60(4):938–968, 2011.

[51] C. Calabro, R. Impagliazzo, and R. Paturi. The complexity of satisfiability of small depth circuits. In *Parameterized and Exact Computation: 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, pages 75–85, 2009.

[52] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir. A model of internet topology using k-shell decomposition. *Proceedings of the National Academy of Sciences*, 104(27):11150–11154, 2007.

[53] C. Chekuri and J. Chuzhoy. Polynomial bounds for the grid-minor theorem. *Journal of the ACM*, 63(5):40, 2016.

[54] H. Chen and M. Müller. One hierarchy spawns another: graph deconstructions and the complexity classification of conjunctive queries. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, page 32, 2014.

[55] C. Chevalier and F. Pellegrini. PT-Scotch: A tool for efficient parallel graph ordering. *Parallel computing*, 34(6):318–331, 2008.

[56] L. Chua and L.-K. Chen. Diakoptic and generalized hybrid analysis. *IEEE Transactions on Circuits and Systems*, 23(12):694–705, 1976.

[57] F. R. Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

[58] B. Courcelle. The monadic second-order theory of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[59] P. Crescenzi, R. Grossi, M. Habib, L. Lanzi, and A. Marino. On computing the diameter of real-world undirected graphs. *Theoretical Computer Science*, 514:84–95, 2013.

[60] P. Crescenzi, R. Grossi, C. Imbrenda, L. Lanzi, and A. Marino. Finding the diameter in real-world graphs. In *Algorithms – ESA 2010: 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part I*, pages 302–313. 2010.

[61] M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlström. On problems as hard as CNF-SAT. *ACM Transactions on Algorithms*, 12(3):41, 2016.

[62] M. Cygan, S. Kratsch, and J. Nederlof. Fast hamiltonicity checking via bases of perfect matchings. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 301–310. ACM, 2013.

[63] P. de la Torre, R. Greenlaw, and A. A. Schäffer. Optimal edge ranking of trees in polynomial time. *Algorithmica*, 13(6):592–618, 1995.

[64] M. Deijfen and W. Kets. Random intersection graphs with tunable distribution and clustering. *Probability in the Engineering and Informational Sciences*, 23:661–674, 2009.

[65] E. D. Demaine, F. Reidl, P. Rossmanith, F. Sánchez Villaamil, S. Sikdar, and B. D. Sullivan. Structural sparsity of complex networks: Bounded expansion in random models and real-world graphs. *arXiv e-prints*, 2014, arXiv:1406.2587.

[66] J. S. Deogun, T. Kloks, D. Kratsch, and H. Müller. On vertex ranking for permutation and other graphs. In *11th Annual Symposium on Theoretical Aspects of Computer Science Caen, France, February 24-26, 1994 Proceedings*, pages 747–758. Springer, 1994.

[67] J. S. Deogun, T. Kloks, D. Kratsch, and H. Müller. On the vertex ranking problem for trapezoid, circular-arc and other graphs. *Discrete Applied Mathematics*, 98(1):39–63, 1999.

[68] D. Dereniowski and A. Nadolski. Vertex rankings of chordal graphs and weighted trees. *Information Processing Letters*, 98(3):96–100, 2006.

[69] T. S. Developers. *SageMath, the Sage Mathematics Software System (Version 7.1.0)*, 2016. `http://www.sagemath.org`.

[70] J. Diaz, O. Pottonen, M. Serna, and E. J. van Leeuwen. Complexity of metric dimension on planar graphs. *Journal of Computer and System Sciences*, 83(1):132–158, 2017.

[71] G. Ding. Subgraphs and well-quasi-ordering. *Journal of Graph Theory*, 16(5):489–502, 1992.

[72] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[73] A. Drucker, J. Nederlof, and R. Santhanam. Exponential time paradigms through the polynomial time lens. In *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57, 2016.

[74] I. Duff. A multifrontal approach for solving sparse linear equations. In *Numerical Methods*, pages 87–98. 1983.

[75] I. S. Duff. Full matrix techniques in sparse gaussian elimination. In *Numerical Analysis*, pages 71–84. 1982.

[76] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear. *ACM Transactions on Mathematical Software*, 9(3):302–325, 1983.

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

[78] Z. Dvořák, D. Kráľ, and R. Thomas. Testing first-order properties for subclasses of sparse graphs. *Journal of the ACM*, 60(5):36, 2013.

[79] L. C. Eggan et al. Transition graphs and the star-height of regular events. *The Michigan mathematical journal*, 10(4):385–397, 1963.

[80] K. Eickmeyer, M. Elberfeld, and F. Harwath. Expressivity and succinctness of order-invariant logics on depth-bounded structures. In *Mathematical Foundations of Computer Science 2014: 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, pages 256–266, 2014.

[81] M. Elberfeld. *Space and Circuit Complexity of Monadic Second-order Definable Problemes on Tree-decomposable Structures*. PhD thesis, Zentrale Hochschulbibliothek Lübeck, 2012.

[82] M. Elberfeld, M. Grohe, and T. Tantau. Where first-order and monadic second-order logic coincide. *ACM Transactions on Computational Logic*, 17(4):25, 2016.

[83] M. Elberfeld, A. Jakoby, and T. Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. In *Proceedings of the 29th Symposium on Theoretical Aspects of Computer Science*, volume 14, pages 66–77, 2012.

[84] J. A. Ellis, I. H. Sudborough, and J. S. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–79, 1994.

[85] M. R. Fellows, B. M. Jansen, and F. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European Journal of Combinatorics*, 34(3):541–566, 2013.

[86] M. R. Fellows, D. Lokshtanov, N. Misra, F. A. Rosamond, and S. Saurabh. Graph layout problems parameterized by vertex cover. In *International Symposium on Algorithms and Computation*, pages 294–305. Springer, 2008.

[87] S. Finbow, A. King, G. MacGillivray, and R. Rizzi. The firefighter problem for graphs of maximum degree three. *Discrete Mathematics*, 307(16):2094–2105, 2007.

[88] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

[89] T. Fluschnik, C. Komusiewicz, G. B. Mertzios, A. Nichterlein, R. Niedermeier, and N. Talmon. When can graph hyperbolicity be computed in linear time? *arXiv e-prints*, 2017, arXiv:1702.06503.

[90] T. Fluschnik, S. Kratsch, R. Niedermeier, and M. Sorge. The parameterized complexity of the minimum shared edges problem. *arXiv e-prints*, 2016, arXiv:1602.01739.

[91] F. V. Fomin, A. C. Giannopoulou, and M. Pilipczuk. Computing tree-depth faster than 2^n. In *Parameterized and Exact Computation*, volume 8246 of *Lecture Notes in Computer Science*, pages 137–149. 2013.

[92] F. V. Fomin, D. Lokshtanov, M. Pilipczuk, S. Saurabh, and M. Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1419–1432, 2017.

[93] M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 345–354, 1989.

[94] M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130(1–3):3–31, 2004.

[95] M. Fürer and H. Yu. Space saving by dynamic algebraization based on tree-depth. *Theory of Computing Systems*, pages 1–22, 2017.

[96] J. Gajarsky and P. Hliněný. Faster deciding MSO properties of trees of fixed height, and some consequences. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012)*, volume 18, 2012.

[97] J. Gajarskỳ, P. Hliněnỳ, J. Obdržálek, S. Ordyniak, F. Reidl, P. Rossmanith, F. Sánchez Villaamil, and S. Sikdar. Kernelization using structural parameters on sparse graph classes. *Journal of Computer and System Sciences*, 84:219–242, 2017.

[98] B. A. Galler and M. J. Fisher. An improved equivalence algorithm. *Communications of the ACM*, 7(5):301–303, 1964.

[99] R. Ganian, P. Hliněný, J. Kneis, A. Langer, J. Obdržálek, and P. Rossmanith. Digraph width measures in parameterized algorithmics. *Discrete Applied Mathematics*, 168:88–107, 2014.

[100] A. George and J. W. Liu. The evolution of the minimum degree ordering algorithm. *Siam review*, 31(1):1–19, 1989.

[101] A. C. Giannopoulou, P. Hunter, and D. M. Thilikos. LIFO-search: A min-max theorem and a searching game for cycle-rank and tree-depth. *Discrete Applied Mathematics*, 160(15):2089–2097, 2012.

[102] A. C. Giannopoulou, B. M. Jansen, D. Lokshtanov, and S. Saurabh. Uniform kernelization complexity of hitting forbidden minors. In *Automata, Languages, and Programming: 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 629–641, 2015.

[103] A. C. Giannopoulou, G. B. Mertzios, and R. Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *arXiv e-prints*, 2015, arXiv:1506.01652.

[104] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis. Evaluating cooperation in communities with the *k*-core structure. In *Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on*, pages 87–93, 2011.

[105] E. Godehardt, J. Jarowski, and K. Rybarczyk. Clustering coefficients of random intersection graphs. In *Challenges at the Interface of Data Analysis, Computer Science and Optimization: Proceedings of the 34th Annual Conference of the Gesellschaft für Klassifikation e. V., Karlsruhe, July 21 - 23, 2010*, pages 243–253. 2012.

[106] P. Golovach, P. Heggernes, D. Kratsch, D. Lokshtanov, D. Meister, and S. Saurabh. Bandwidth on AT-free graphs. *Algorithms and Computation*, pages 573–582, 2009.

[107] M. C. Golumbic. Trivially perfect graphs. *Discrete Mathematics*, 24(1):105–107, 1978.

[108] C. Groër, B. D. Sullivan, and D. Weerapurage. INDDGO: Integrated network decomposition & dynamic programming for graph optimization. *ORNL/TM-2012/176*, 2012.

[109] H. Gruber and M. Holzer. Finite automata, digraph connectivity, and regular expression size. *Automata, Languages and Programming*, pages 39–50, 2008.

[110] H. Gruber and M. Holzer. Provably shorter regular expressions from finite automata. *International Journal of Foundations of Computer Science*, 24(08):1255–1279, 2013.

[111] S. Gulan. *On the relative descriptional complexity of regular expressions and finite automata*. PhD thesis, Universität Trier, 2011.

[112] G. Gutin, M. Jones, and M. Wahlström. Structural parameterizations of the mixed chinese postman problem. In *Algorithms – ESA 2015: 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 668–679. 2015.

[113] H. Hafsteinsson. Parallel sparse Cholesky factorization. Technical report, Cornell University, 1988.

[114] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, pages 11–15, Aug. 2008.

[115] R. Halin. Zur Klassifikation der endlichen Graphen nach H. Hadwiger und K. Wagner. *Mathematische Annalen*, 172(1):46–78, 1967.

[116] R. Halin. S-functions for graphs. *Journal of Geometry*, 8(1-2):171–186, 1976.

[117] W. Hayes, K. Sun, and N. Pržulj. Graphlet-based measures are suitable for biological network comparison. *Bioinformatics*, 29(4):483–491, 2013.

[118] P. Helman. A common schema for dynamic programming and branch and bound algorithms. *Journal of the ACM*, 36(1):97–128, 1989.

[119] B. Hendrickson et al. Effective sparse matrix ordering: Just around the bend. In *Proceedings of 8th SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[120] M. Holzer, F. Schulz, and D. Wagner. Engineering multilevel overlay graphs for shortest-path queries. *Journal of Experimental Algorithmics*, 13:5, 2009.

[121] P. Hunter. LIFO-search on digraphs: A searching game for cycle-rank. In *Fundamentals of Computation Theory: 18th International Symposium, FCT 2011, Oslo, Norway, August 22-25, 2011. Proceedings*, pages 217–228, 2011.

[122] R. Impagliazzo and R. Paturi. Complexity of *k*-SAT. In *Computational Complexity, 1999. Proceedings. Fourteenth Annual IEEE Conference on*, pages 237–240, 1999.

[123] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 653–662, 1998.

[124] S. Itzkovitz, R. Levitt, N. Kashtan, R. Milo, M. Itzkovitz, and U. Alon. Coarse-graining and self-dissimilarity of complex networks. *Physical Review E*, 71(1):016127, 2005.

[125] A. V. Iyer, H. D. Ratliff, and G. Vijayan. On a node ranking problem of trees and graphs. Technical report, DTIC Document, 1986.

[126] A. V. Iyer, H. D. Ratliff, and G. Vijayan. Optimal node ranking of trees. *Information Processing Letters*, 28(5):225–229, 1988.

[127] A. V. Iyer, H. D. Ratliff, and G. Vijayan. On an edge ranking problem of trees and graphs. *Discrete Applied Mathematics*, 30(1):43–52, 1991.

[128] L. Jaffke and B. M. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. *arXiv e-prints*, 2017, arXiv:1701.06985.

[129] B. M. Jansen. *The power of data reduction: Kernels for Fundamental Graph Problems*. PhD thesis, Utrecht University, 2013.

[130] K. Jasik. Treewidth on Fire. Bachelor's thesis, RWTH Aachen University, Germany, 2015.

[131] J. Jaworski, M. Karoński, and D. Stark. The degree of a typical vertex in generalized random intersection graph models. *Discrete Mathematics*, 306:2152–2165, 2006.

[132] J. A. G. Jess and H. G. M. Kees. A data structure for the parallel L\U-decomposition. *THE-afdeling Electrotechniek*.

[133] J. A. G. Jess and H. G. M. Kees. A data structure for parallel L/U decomposition. *IEEE Transactions on Computers*, 31(3):231–239, 1982.

[134] D. S. Johnson and M. A. Trick. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*, volume 26. American Mathematical Soc., 1996.

[135] E. Jones, T. Oliphant, and P. Peterson. *SciPy: open source scientific tools for Python*, 2014.

[136] M. Karoński and K. Singer-Cohen. On random intersection graphs: the subgraph problem. *Combinatorics, Probability and Computing*, 8:131–159, 1999.

[137] R. M. Karp and M. Held. Finite-state processes and dynamic programming. *SIAM Journal on Applied Mathematics*, 15(3):693–718, 1967.

[138] I. Karpas, O. Neiman, and S. Smorodinsky. On vertex rankings of graphs and its relatives. *Discrete Mathematics*, 338(8):1460–1467, 2015.

[139] G. Karypis and V. Kumar. MeTis: Unstructured graph partitioning and sparse matrix ordering system, version 4.0. `http://www.cs.umn.edu/~metis`.

[140] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[141] G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48(1):71–95, 1998.

[142] M. Katchalski, W. McCuaig, and S. Seager. Ordered colourings. *Discrete Mathematics*, 142(1–3):141–154, 1995.

[143] I. Katsikarelis, M. Lampis, and V. T. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *arXiv e-prints*, 2017, arXiv:1704.08868.

[144] K. Kawarabayashi and B. Rossman. A polynomial excluded-minor approximation of treedepth. `http://www.math.toronto.edu/rossman/treedepth.pdf`, 2017.

[145] H. G. M. Kees. *The organization of circuit analysis on array architectures*. PhD thesis, Dept. of Electrical Engineering, Eindhoven University of Technology, 1982.

[146] A. Kevorkian and J. Snoek. Decomposition in large scale systems: Theory and applications in solving large sets of nonlinear simultaneous equations. *Decomposition of Large Scale Problems. North Holland Publ. Comp*, pages 146–160, 1973.

[147] N. G. Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, 42(6):345–350, 1992.

[148] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse. Identification of influential spreaders in complex networks. *Nature physics*, 6(11):888–893, 2010.

[149] T. Kloks. *Treewidth: computations and approximations*, volume 842. Springer Science & Business Media, 1994.

[150] J. Kneis and A. Langer. A practical approach to Courcelle's Theorem. In *Proceedings of the 4th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS)*, pages 99–106. Z. Novotný, 2008.

[151] Y. Kobayashi and H. Tamaki. Treedepth parameterized by vertex cover number. In *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63, 2017.

[152] A. M. C. A. Koster, H. L. Bodlaender, and S. P. M. V. Hoesel. Treewidth: Computational experiments. *Electronic Notes in Discrete Mathematics*, 8:54–57, 2001.

[153] G. Kron. A set of principles to interconnect the solutions of physical systems. *Journal of Applied Physics*, 24(8):965–980, 1953.

[154] O. Kuchaiev, T. Milenković, V. Memišević, W. Hayes, and N. Pržulj. Topological network alignment uncovers biological function and phylogeny. *Journal of The Royal Society Interface*, 7(50):1341–1354, 2010.

[155] T. W. Lam and F. L. Yue. Edge ranking of graphs is hard. *Discrete Applied Mathematics*, 85(1):71–86, 1998.

[156] M. Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012.

[157] A. Langer. *Fast algorithms for decomposable graphs*. PhD thesis, RWTH Aachen University, 2013.

[158] A. Langer, F. Reidl, P. Rossmanith, and S. Sikdar. Evaluation of an MSO-solver. In D. A. Bader and P. Mutzel, editors, *Proceedings of ALENEX'12*, pages 55–63. Society for Industrial and Applied Mathematics, 2012.

[159] A. Langer, F. Reidl, P. Rossmanith, and S. Sikdar. Sequoia homepage. `http://sequoia.informatik.rwth-aachen.de/sequoia/`, 2012. Visited 2012-09-16.

[160] A. Langer, F. Reidl, P. Rossmanith, and S. Sikdar. Practical algorithms for MSO model-checking on tree-decomposable graphs. *Computer Science Review*, 13:39–74, 2014.

[161] D. LaSalle and G. Karypis. Efficient nested dissection for multicore architectures. In *Euro-Par 2015: Parallel Processing: 21st International Conference on Parallel and Distributed Computing, Vienna, Austria, August 24-28, 2015, Proceedings*, pages 467–478, 2015.

[162] C. E. Leiserson. Area-efficient graph layouts. In *Foundations of Computer Science, 1980., 21st Annual Symposium on*, pages 270–281, 1980.

[163] K. Levenberg. A method for the solution of certain non–linear problems in least squares. *Quarterly of Applied Mathematics*, 2(2):164–168, 1944.

[164] T. Ligotti and M. Cardin. *Born to Fear: Interviews With Thomas Ligotti*. Subterranean, 2014.

[165] S. Lindell. A logspace algorithm for tree canonization. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 400–404, 1992.

[166] J. W. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, 11(2):141–153, 1985.

[167] J. W. Liu. Equivalent sparse matrix reordering by elimination tree rotations. *SIAM Journal on Scientific and Statistical Computing*, 9(3):424–444, 1988.

[168] J. W. Liu. A graph partitioning algorithm by node separators. *ACM Transactions on Mathematical Software*, 15(3):198–219, 1989.

[169] J. W. Liu. Reordering sparse matrices for parallel elimination. *Parallel computing*, 11(1):73–91, 1989.

[170] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM Journal on Matrix Analysis and Applications*, 11(1):134–172, 1990.

[171] D. C. Llewellyn, C. Tovey, and M. Trick. Local optimization on graphs. *Discrete Applied Mathematics*, 23(2):157–178, 1989.

[172] D. Lokshtanov, D. Marx, and S. Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete algorithms*, pages 777–789, 2011.

[173] D. Lokshtanov and J. Nederlof. Saving space by algebraization. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 321–330, 2010.

[174] C. Magnien, M. Latapy, and M. Habib. Fast computation of empirically tight bounds for the diameter of massive graphs. *Journal of Experimental Algorithmics*, 13:10, 2009.

[175] F. Manne. An algorithm for computing an elimination tree of minimum height for a tree. Technical report, University of Bergen, Norway, 1991.

[176] F. Manne. *Reducing the height of an elimination tree through local reorderings*. University of Bergen. Department of Informatics, 1991.

[177] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[178] D. Marx and V. Mitsou. Double-exponential and triple-exponential bounds for choosability problems parameterized by treewidth. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55, 2016.

[179] A. Masoudi-Nejad, F. Schreiber, and Z. R. M. Kashani. Building blocks of biological networks: a review on major network motif discovery algorithms. *IET Systems Biology*, 6:164–174, 2012.

[180] R. Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–136, 1979.

[181] G. B. Mertzios, A. Nichterlein, and R. Niedermeier. Fine-grained algorithm design for matching. *arXiv e-prints*, 2016, arXiv:1609.08879.

[182] T. Milenković, V. Memišević, A. K. Ganesan, and N. Pržulj. Systems-level cancer gene identification from protein interaction network topology applied to melanogenesis-related functional genomics data. *Journal of The Royal Society Interface*, 7(44):423–437, 2010.

[183] T. Milenković and N. Pržulj. Uncovering biological network function via graphlet degree signatures. *Cancer informatics*, 6(6):257, 2008.

[184] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[185] J. Nederlof. *Space and time efficient structural improvements of dynamic programming algorithms*. PhD thesis, University of Bergen, 2011.

[186] J. Nešetřil and P. Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European Journal of Combinatorics*, 27(6):1022–1041, 2006.

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

[188] J. Nešetřil and S. Shelah. On the order of countable graphs. *European Journal of Combinatorics*, 24(6):649–663, 2003.

[189] J. Nešetřil and P. Ossona de Mendez. Grad and classes with bounded expansion I. and II. *European Journal of Combinatorics*, 29(3):760–791, 2008.

[190] J. Nešetřil and P. Ossona de Mendez. Grad and classes with bounded expansion I. Decompositions. *European Journal of Combinatorics*, 29(3):760–776, 2008.

[191] J. Nešetřil and P. Ossona de Mendez. First order properties on nowhere dense structures. *The Journal of Symbolic Logic*, 75(3):868–887, 2010.

[192] J. Nešetřil and P. Ossona de Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 32(4):600–617, 2011.

[193] J. Nešetřil and P. Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

[194] J. Nešetřil, P. Ossona de Mendez, and D. R. Wood. Characterisations and examples of graph classes with bounded expansion. *European Journal of Combinatorics*, 33(3):350–373, 2012.

[195] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, 64(2), 2001.

[196] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[197] M. P. O'Brien et al. CONCUSS: Version 1.0, Sept. 2015. 10.5281/zenodo.30281.

[198] T. Oelschlägel. Treewidth from Treedepth. Bachelor's thesis, RWTH Aachen University, Germany, 2014.

[199] T. Oelschlägel. Graph Partitioning Problems on Graphs of Bounded Treedepth. Master's thesis, RWTH Aachen University, Germany, 2016.

[200] J. T. M. Pieck. Formele definitie van een e-tree. Technical report, Technische Hogeschool Eindhoven, 1980.

[201] M. Pilipczuk and M. Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. In *33rd Symposium on Theoretical Aspects of Computer Science*, pages 57:1–57:15, 2016.

[202] B. Pittel, J. Spencer, and N. Wormald. Sudden emergence of a giant k-core in a random graph. *Journal of Combinatorial Theory, Series B*, 67(1):111–151, 1996.

[203] P. Pongpaibool, S. Pukkawanna, and V. Visoottiviseth. Lightweight detection of DoS attacks. In *2007 15th IEEE International Conference on Networks*, pages 77–82, 2007.

[204] A. Pothen. The complexity of optimal elimination trees. Technical report, Pennsylvannia State University, 1988.

[205] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Analysis and Applications*, 11(3):430–452, 1990.

[206] N. Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):177–183, 2007.

[207] F. Reidl. *Structural sparseness and complex networks*. PhD thesis, RWTH Aachen, Aachen, 2016.

[208] F. Reidl, P. Rossmanith, F. Sánchez Villaamil, and S. Sikdar. A faster parameterized algorithm for treedepth. In *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 931–942, 2014.

[209] P. Ribeiro, F. Silva, and M. Kaiser. Strategies for network motifs discovery. In *Fifth IEEE International Conference on e-Science*, pages 80–87, 2009.

[210] N. Robertson and P. D. Seymour. Graph minors XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63:65–110, 1995.

[211] N. v. Roden. Spanheight, a natural extension of bandwidth and treedepth. Master's thesis, 2015.

[212] J. V. Rooij, H. L. Bodlaender, and P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Algorithms – ESA 2009: 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, number 5193 in LNCS, pages 566–577. Springer, 2009.

[213] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on computing*, 5(2):266–283, 1976.

[214] K. Rybarczyk. The coupling method for inhomogeneous random intersection graphs. *The Electronic Journal of Combinatorics*, 24(2):P2–10, 2017.

[215] A. Sangiovanni-Vincentelli. A graph theoretical interpretation of nonsymmetric permutation on sparse matrices. *International Journal of Circuit Theory and Applications*, 5(2):139–147, 1977.

[216] A. A. Schäffer. Optimal node ranking of trees in linear time. *Information Processing Letters*, 33(2):91–96, 1989.

[217] P. Scheffler. *Dynamic programming algorithms for tree-decomposition problems*. Institut für Mathematik, Akademie der Wissenschaft der DDR, 1986.

[218] R. Schreiber. A new implementation of sparse gaussian elimination. *ACM Transactions on Mathematical Software*, 8(3):256–276, 1982.

[219] P. D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993.

[220] K. Singer-Cohen. *Random Intersection Graphs*. PhD thesis, Department of Mathematical Sciences, The Johns Hopkins University, 1995.

[221] O. Sporns and R. Kötter. Motifs in brain networks. *PLoS Biology*, 2(11):369, 2004.

[222] J. A. Storer. *An introduction to data structures and algorithms*. Springer Science & Business Media, 2012.

[223] A. Takahashi, S. Ueno, and Y. Kajitani. Minimal acyclic forbidden minors for the family of graphs with bounded path-width. *Discrete Mathematics*, 127(1-3):293–304, 1994.

[224] F. W. Takes and W. A. Kosters. Computing the eccentricity distribution of large graphs. *Algorithms*, 6(1):100, 2013.

[225] J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial *k*-trees. *SIAM Journal on Discrete Mathematics*, 10(4):529–550, 1997.

[226] S. Teso, J. Staiano, B. Lepri, A. Passerini, and F. Pianesi. Ego-centric graphlets for personality and affective states recognition. In *Social Computing (SocialCom), 2013 International Conference on*, pages 874–877, 2013.

[227] J. Ugander, L. Backstrom, and J. Kleinberg. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1307–1318, 2013.

[228] J. D. Ullman. *Computational aspects of VLSI*. Computer Science Press, 1984.

[229] T. van Dijk, J.-P. van den Heuvel, and W. Slob. Computing treewidth with LibTW. `http://www.treewidth.com/treewidth/docs/LibTW.pdf`, 2006.

[230] M. van Ee. Some notes on bounded starwidth graphs. *Information Processing Letters*, 125:9—14, 2017.

[231] D. Wagner and T. Willhalm. Speed-up techniques for shortest-path computations. In *STACS 2007: 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007. Proceeding*, pages 23–36, 2007.

[232] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[233] M. Wrochna. Reconfiguration in bounded bandwidth and treedepth. *arXiv e-prints*, 2014, arXiv:1405.0847.

[234] B. Yang. Strong-mixed searching and pathwidth. *Journal of Combinatorial Optimization*, 13(1):47–59, 2007.

Part VIII

APPENDIX

A
PROBLEMS

q-Coloring

Input:	A graph G and an integer q.
Problem:	Is there a function $f\colon V(G) \to \{1, \ldots, q\}$ such that for every $uv \in E(G)$ it holds that $f(u) \neq f(v)$?

Vertex Cover

Input:	A graph G and an integer k.
Problem:	Is there a subset $X \subseteq V(G)$ with at least k vertices such that $G \setminus X$ is edgeless?

Independent Set

Input:	A graph G and an integer k.
Problem:	Is there a subset $X \subseteq V(G)$ with at least k vertices such that $G[X]$ is edgeless?

Dominating Set

Input:	A graph G and an integer k.
Problem:	Is there a subset $X \subseteq V(G)$ with at most k vertices such that the closed neighborhood of X is $V(G)$?

k-SAT

Input:	A Boolean formula ϕ in conjunctive normal form such that every clause has size at most k.
Problem:	Is there a satisfying assignment for ϕ?

B EXPERIMENTAL RESULT OF TREEWIDTH HEURISTICS

This part of the appendix contains all the results of the experiments which were described in Section 25. The first table contains the results for running the improvement heuristic for the stretch starting from a treedepth decomposition which was not computed from an elimination ordering; the overall statistics for these results can be found in Table 25.1 in Section 25. The second table contains the result for the improvement of the stretch starting from a treedepth decomposition generated from the elimination order given by a different treewidth heuristic.

The graphs are ordered alphabetically, since this automatically clusters graphs which have been generated in a similar way or arise from the same practical context. The first two column give the number of nodes and edges of the graph. An empty entry in the table signifies that the experiment did not finish in five minutes. For a description of the graphs please refer to the datasets used and their corresponding sources [1, 2, 134, 229].

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	dfs	max	num	ev	result	factor	#
4x12_torusGrid	48	96	11	11	**8**	**8**	24	**8**	11	10	24	10	11	12	12	10	24	9	11	✗	1.12	2/5
6s10.gaifman	33900	94299	878	730	832		2722						772	**570**	635	1997	1997	1997		✗	3.50	7/7
6s11-opt.gaifman	33276	92832	827	978	864		3044						778	**592**	661	2457	2457	2457		✗	4.15	7/7
6s12.gaifman	34033	94695	812	849	714		2715						740	722	**569**	2164	2164	2164		✗	3.80	7/7
8x6_torusGrid	48	96	14	14	14	**12**	**12**	**12**	14	**12**	**12**	**12**	14	15	14	**12**	**12**	13	13	=	1.00	1/3
aes_32_3_keyfind_1.gaifman	708	2292	82	78	85	77	135	77	82	129	158	142	82	90	**75**	137	114	160	206	✗	1.52	7/10
aes_64_1_keyfind_1.gaifman	596	2092	79	63	71	**55**	81	**55**	79	99	84	80	79	69	67	134	108	91	155	✗	1.65	10/10
AhrensSzekeresGeneralizedQuadrangleGraph_3	27	135	20	20	20	20	21	20	**17**	**17**	22	**17**	20	20	**17**	19	**17**	**17**	19	=	1.00	1/4
alarm	37	65	**4**	**4**	**4**	**4**	5	**4**	**4**	**4**	5	**4**	**4**	5	5	5	5	5	5	✗	1.25	2/2
anna	110	259	**8**	**8**	**8**	**8**	9	**8**	**8**	9	11	9	**8**	9	9	11	**8**	10	17	=	1.00	1/3
anna	134	423	**12**	**12**	13	**12**	15	**12**	**12**	15	14	17	**12**	13	13	19	16	18	22	✗	1.33	5/5
anna	138	493	**12**	**12**	13	**12**	13	**12**	**12**	14	14	13	**12**	13	13	21	16	13	22	✗	1.08	2/3
anna-pp	22	148	**12**	**12**	13	**12**	13	**12**	**12**	13	16	13	**12**	14	14	13	13	14	14	✗	1.08	2/4
AProVE07-01.gaifman	7502	394687	1066	1019	1056		1177				1153		1055	**853**	855	1637	1637	1637		✗	1.92	
AProVE07-03.gaifman	3114	17553	257	241	265	**226**	480	**226**	257		434	567	229	246	250	310	443	310	1003	✗	1.37	8/10
BalancedTree_3,5	364	363	**1**	**1**	**1**	3	81	3	3	**1**	**1**	**1**	4	4	4	**1**	**1**	27	10	=	1.00	1/4
barley	48	126	8	**7**	8	**7**	10	**7**	8	9	11	**7**	9	8	8	10	8	8	11	✗	1.14	2/5
barley-pp	26	78	**7**	**7**	**7**	**7**	9	**7**	**7**	8	10	**7**	**7**	9	9	8	9	**7**	8	=	1.00	1/4
BiggsSmithGraph	102	153	25	24	24	22	30	23	22	24	28	24	24	22	**21**	28	24	24	25	✗	1.14	4/7
BlanusaSecondSnarkGraph	18	27	**5**	**5**	**5**	**5**	6	**5**	**5**	6	7	6	**5**	**5**	**5**	**5**	6	6	**5**	=	1.00	1/3
BrinkmannGraph	21	42	9	9	9	9	9	9	10	9	11	9	9	10	9	**8**	9	9	10	✓	0.89	1/3
BrouwerHaemersGraph	81	810	62	61	61	**54**	70	**54**	**54**	63	75	63	61	64	61	61	**54**	64	63	=	1.00	1/7
BubbleSortGraph_5	120	240	31	31	31	29	48	29	31	**23**	36	**23**	31	28	28	31	36	31	35	✗	1.35	4/6
CameronGraph	231	3465	191	188	193	177	186	177	186	**175**	187	**175**	190	184	184	186	187	182	187	✗	1.04	3/9
celar02	100	311	**10**	**10**	11	**10**	19	**10**	**10**	**10**	13	**10**	**10**	13	13	21	11	12	16	✗	1.10	2/4
celar07	200	817	18	**16**	18	**16**	29	**16**	18	28	21	19	18	24	24	44	25	22	29	✗	1.38	5/7
Cell120	600	1200	124	126	110	119	237	114	125	**78**	112	**78**	118	94	93	132	128	89	146	✗	1.14	2/12
ChvatalGraph	12	24	**6**	**6**	**6**	**6**	7	**6**	**6**	**6**	7	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	=	1.00	1/2
ClebschGraph	16	40	10	**8**	**8**	**8**	10	**8**	10	**8**	10	**8**	**8**	**8**	**8**	9	**8**	**8**	10	=	1.00	1/2
contiki_calc_input_to_operand1	31	33	**2**	**2**	**2**	**2**	3	**2**	**2**	**2**	**2**	**2**	**2**	3	3	**2**	3	3	3	=	1.00	1/2

Table B.1: [1/13] For a list of what the INDDGO heuristics are, see page 115. For the heuristics derived from starting from a treedepth decomposition, the treedepth decomposition is computed via a (dfs) depth first search; finding separators minimizing the (max) size of the biggest component, (num) maximizing the number of components or (ev) via eigenvectors. The result shows if the best value d of the treedepth heuristic was better, equal or worse than the best of value i of the INDDGO heuristics and factor is d/i. The last column shows in which position d would be in a ordered list of the INDDGO values out of how many different INDDGO values there are.

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	dfs	max	num	ev	result	factor	#
contiki_collect_enqueue_dummy_packet	46	46	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	=	1.00	1/1
contiki_collect_received_announcement	52	59	2	2	2	2	3	2	2	2	3	2	3	3	3	4	3	4	5	✗	1.50	2/2
contiki_collect_send_ack	53	52	1	1	1	2	2	2	1	1	1	1	2	2	2	1	2	2	2	=	1.00	1/2
contiki_collect_send_next_packet	26	25	1	1	1	2	2	2	1	1	1	1	2	2	2	1	2	2	2	=	1.00	1/2
contiki_collect_send_queued_packet	95	99	2	2	2	2	3	2	2	3	3	3	2	3	3	3	3	3	6	✗	1.50	2/2
contiki_contikimac_input_packet	116	127	3	3	3	3	5	4	3	4	5	4	3	3	3	4	5	3	8	=	1.00	1/3
contiki_contikimac_powercycle	166	194	5	5	5	5	10	5	5	7	11	10	5	6	6	11	9	6	13	✗	1.20	2/5
contiki_ctk_ctk_menu_add	25	27	2	2	2	2	3	2	2	3	3	3	2	3	3	3	3	2	3	=	1.00	1/2
contiki_cxmac_input_packet	90	97	3	3	3	3	4	3	3	4	4	4	3	3	3	4	5	4	6	✗	1.33	2/2
contiki_dhcpc_dhcpc_init	34	34	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	2	3	=	1.00	1/1
contiki_dhcpc_dhcpc_request	27	27	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	2	3	=	1.00	1/1
contiki_dhcpc_handle_dhcp	276	313	6	6	6	6	14	6	6	17	11	17	6	7	9	13	15	15	15	✗	2.17	5/6
contiki_httpd-cfs_send_file	44	48	3	3	3	3	4	3	3	4	5	5	4	5	5	5	6	4	4	✗	1.33	2/3
contiki_httpd-cfs_send_headers	106	116	3	3	3	3	5	3	3	7	5	10	3	4	4	5	5	4	10	✗	1.33	2/5
contiki_ifft_ifft	172	180	2	2	2	2	4	2	2	3	3	3	2	5	4	3	4	3	8	✗	1.50	2/4
contiki_ircc_handle_connection	138	161	6	5	6	7	9	6	7	13	12	13	6	6	6	12	14	12	13	✗	2.40	5/6
contiki_ircc_list_channel	70	76	3	3	3	3	6	3	3	6	4	6	3	5	5	5	5	4	4	✗	1.33	2/4
contiki_lpp_dutycycle	102	114	5	5	5	6	6	5	5	6	7	6	5	6	6	6	8	7	10	✗	1.20	2/3
contiki_lpp_init	22	21	1	1	1	2	2	2	1	1	1	1	2	2	2	1	2	2	2	=	1.00	1/2
contiki_lpp_send_packet	116	120	2	2	2	3	2	2	2	4	4	4	2	3	3	4	3	2	3	=	1.00	1/3
contiki_lpp_send_probe	92	94	2	2	2	2	2	2	2	3	3	3	2	3	3	3	3	3	4	✗	1.50	2/2
contiki_nullrdc_packet_input	28	30	3	3	3	3	3	3	3	3	3	3	3	3	3	4	4	3	3	=	1.00	1/1
contiki_polite-announcement_send_timer	31	31	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	2	2	=	1.00	1/1
contiki_powertrace_add_stats	46	47	2	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3	4	✗	1.50	2/2
contiki_powertrace_powertrace_print	323	323	2	2	2	2	2	2	2	2	2	2	2	3	3	2	3	2	3	=	1.00	1/2
contiki_process_exit_process	72	82	3	3	3	3	7	3	3	5	4	3	3	5	5	3	5	4	5	=	1.00	1/4
contiki_profile_profile_episode_start	31	32	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	=	1.00	1/1
contiki_psock_psock_generator_send	61	68	4	4	4	4	7	4	4	5	6	5	4	5	5	6	4	6	6	=	1.00	1/4
contiki_psock_psock_readto	56	61	4	3	4	3	4	3	4	4	6	4	4	4	4	6	6	5	6	✗	1.67	3/3
contiki_ringbuf_ringbuf_put	29	29	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	=	1.00	1/1

[2/13]

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	dfs	max	num	ev	result	factor	#
contiki_route-discovery_route_discovery_discover	20	20	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	=	1.00	1/1
contiki_rudolph1_rudolph1_open	27	26	**1**	**1**	**1**	2	2	2	**1**	**1**	**1**	**1**	2	2	2	**1**	2	2	2	=	1.00	1/2
contiki_rudolph1_write_data	35	36	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	**2**	3	3	4	=	1.00	1/2
contiki_serial-line_process_thread_serial_line_pro-cess	72	81	**4**	**4**	**4**	**4**	6	**4**	**4**	5	6	6	**4**	**4**	**4**	7	7	5	6	✗	1.25	2/3
contiki_shell-base64_base64_add_char	70	74	**2**	**2**	**2**	**2**	3	**2**	**2**	5	3	5	3	3	3	3	3	3	6	✗	1.50	2/3
contiki_shell-collect-view_process_thread	61	62	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	3	**2**	3	3	3	**2**	3	3	=	1.00	1/2
contiki_shell-netperf_memcpy_misaligned	30	32	**2**	**2**	**2**	**2**	3	**2**	**2**	3	3	3	**2**	3	3	4	3	3	3	✗	1.50	2/2
contiki_shell-ps_process_thread_shell_ps_process	45	46	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	3	**2**	4	4	3	3	3	4	✗	1.50	2/3
contiki_shell-rime-debug_recv_broadcast	24	23	**1**	**1**	**1**	2	2	2	**1**	**1**	**1**	**1**	2	2	2	**1**	2	2	2	=	1.00	1/2
contiki_shell-rime-ping_recv_mesh	47	47	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	**2**	3	=	1.00	1/1
contiki_shell-rime_process_thread_shell_send_pro-cess	89	95	**3**	**3**	**3**	**3**	4	4	**3**	**3**	5	4	**3**	4	4	5	4	4	6	✗	1.33	2/3
contiki_shell-rime_recv_collect	62	64	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	4	3	3	5	3	4	✗	1.50	2/3
contiki_shell-sendtest_read_chunk	30	32	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	**2**	3	3	4	=	1.00	1/2
contiki_shell-text_process_thread_shell_echo_process	25	25	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	**2**	3	3	3	=	1.00	1/2
contiki_shell_process_thread_shell_server_process	76	85	**3**	**3**	**3**	**3**	6	**3**	**3**	4	4	4	**3**	4	4	4	6	4	5	✗	1.33	2/3
contiki_shell_shell_register_command	42	45	**2**	**2**	**2**	**2**	4	**2**	**2**	3	3	3	3	3	3	3	4	3	4	✗	1.50	2/3
contiki_tcpip_eventhandler	98	112	**2**	**2**	**2**	**2**	4	**2**	**2**	6	4	6	3	5	4	6	6	4	10	✗	2.00	3/5
contiki_uip-neighbor_uip_neighbor_add	67	71	**3**	**3**	**3**	**3**	4	**3**	**3**	4	4	4	**3**	**3**	**3**	5	4	4	4	✗	1.33	2/2
contiki_uip-neighbor_uip_neighbor_periodic	20	21	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	**2**	3	3	3	=	1.00	1/2
contiki_uip-over-mesh_recv_data	85	88	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	**2**	**2**	3	3	3	4	4	4	✗	1.50	2/2
contiki_uip_uip_connect	111	120	**3**	**3**	**3**	**3**	5	**3**	**3**	**3**	4	**3**	**3**	4	4	4	4	4	7	✗	1.33	2/3
contiki_uip_uip_init	26	27	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	3	**2**	3	**2**	3	=	1.00	1/2
contiki_uip_uip_unlisten	19	20	**2**	**2**	**2**	**2**	3	**2**	**2**	3	3	3	**2**	3	3	3	3	3	3	✗	1.50	2/2
contiki_webclient_senddata	108	109	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	3	**2**	**2**	**2**	3	3	**2**	3	=	1.00	1/2
contiki_webclient_webclient_appcall	98	111	**3**	**3**	**3**	**3**	4	**3**	**3**	5	**3**	5	**3**	**3**	**3**	4	4	6	7	✗	1.33	2/3
CycleGraph_100	100	100	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	=	1.00	1/1
david	87	406	**13**	14	14	**13**	15	**13**	**13**	17	16	17	14	**13**	14	24	19	15	27	✗	1.15	3/5
david-pp	29	191	**13**	14	14	**13**	14	**13**	**13**	16	14	16	**13**	14	14	17	20	**13**	15	=	1.00	1/3
DejterGraph	112	336	48	42	42	42	52	42	41	**39**	50	**39**	43	41	42	40	49	42	46	✗	1.03	2/7
DesarguesGraph	20	30	7	**6**	7	**6**	10	**6**	7	8	8	8	**6**	**6**	**6**	7	**6**	**6**	**6**	=	1.00	1/4

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	dfs	max	num	ev	result	factor	#
DodecahedralGraph	20	30	7	**6**	**6**	**6**	7	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	=	1.00	1/2
DorogovtsevGoltsevMendesGraph	3282	6561	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	4	**2**	**2**		184	=	1.00	1/3
DoubleStarSnark	30	45	8	8	**7**	**7**	9	8	**7**	**7**	8	**7**	8	**7**	**7**	8	8	**7**	9	=	1.00	1/3
DSJC125.9	125	6961	120	120	121	**119**	122	**119**	120	120	121	121	120	121	120	121	121	120	121	✗	1.01	2/4
DSJR500.1c	221	23512	215	**212**	217	213	216	213	215	214	218	213	214	**212**	213	216	214	214	216	✗	1.01	3/7
DyckGraph	32	48	9	9	9	9	10	9	9	9	10	9	9	9	**8**	9	10	9	12	✗	1.12	2/3
eil51.tsp	51	140	10	11	10	10	18	10	10	12	**9**	13	11	10	10	16	17	13	15	✗	1.44	5/6
ErreraGraph	17	45	7	7	7	7	7	7	7	7	**6**	8	7	7	7	8	7	7	7	✗	1.17	2/3
FibonacciTree_10	143	142	**1**	**1**	**1**	2	3	2	2	**1**	**1**	**1**	3	3	3	**1**	3	5	5	=	1.00	1/3
FlowerSnark	20	30	7	7	7	**6**	8	**6**	**6**	7	7	7	7	**6**	**6**	7	**6**	7	**6**	=	1.00	1/3
FoldedCubeGraph_7	64	224	**30**	**30**	32	33	42	31	31	31	48	31	31	32	34	34	33	33	34	✗	1.10	4/7
FolkmanGraph	20	40	7	7	7	7	10	**6**	9	8	10	8	7	7	7	8	**6**	7	8	=	1.00	1/5
FosterGraph	90	135	22	23	22	21	23	22	22	24	23	24	24	**20**	22	22	24	22	27	✗	1.10	3/5
fpsol2.i.1-pp	191	4418	72	72	77	72	71	72	72	163	76	163	72	**58**	63	96	96	60	85	✗	1.03	2/7
fpsol2.i.3-pp	193	2721	**28**	**28**	38	**28**	38	**28**	**28**	134	46	134	**28**	35	35	70	31	36	64	✗	1.11	2/5
fpsol2.i.3	206	2645	32	32	30	**28**	39	**28**	32	161	40	161	31	41	34	58	95	40	64	✗	1.43	7/9
fpsol2.i.1	210	5489	**50**	**50**	73	**50**	55	**50**	**50**	147	70	141	**50**	56	56	110	81	59	75	✗	1.18	4/7
fpsol2.i.1-pp	233	10783	**66**	**66**	84	**66**	70	**66**	**66**	216	**66**	**66**	**66**	**66**	**66**	72	115	75	104	✗	1.09	3/4
fpsol2.i.1	269	11654	**66**	**66**	84	**66**	**66**	**66**	**66**	216	**66**	238	**66**	**66**	**66**	72	140	68	115	✗	1.03	2/4
fpsol2.i.3	363	8688	**31**	**31**	52	**31**	41	**31**	**31**	69	42	332	**31**	38	38	35	178	44	81	✗	1.13	2/7
fpsol2.i.2	363	8691	**31**	**31**	52	**31**	41	**31**	**31**	69	42	332	**31**	38	35	35	178	44	74	✗	1.13	2/8
FriendshipGraph_10	21	30	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	=	1.00	1/1
fuzix_abort_abort	21	20	**1**	**1**	**1**	2	2	2	**1**	**1**	**1**	**1**	2	2	2	**1**	2	2	2	=	1.00	1/2
fuzix_bankfixe_pagemap_alloc	21	22	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	**2**	4	3	**2**	=	1.00	1/2
fuzix_clock_gettime_clock_gettime	39	40	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	3	**2**	3	3	3	3	**2**	3	=	1.00	1/2
fuzix_clock_gettime_div10quickm	30	29	**1**	**1**	**1**	2	2	2	**1**	**1**	**1**	**1**	2	2	2	**1**	2	2	2	=	1.00	1/2
fuzix_clock_settime_clock_settime	20	21	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	3	**2**	**2**	**2**	3	**2**	**2**	3	=	1.00	1/2
fuzix_devf_fd_transfer	119	129	**3**	**3**	**3**	**3**	5	**3**	**3**	6	6	6	**3**	4	5	5	4	5	6	✗	1.33	2/4
fuzix_devio_bfind	27	29	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	4	**3**	**3**	4	=	1.00	1/1
fuzix_devio_kprintf	69	78	**3**	**3**	**3**	**3**	4	**3**	**3**	5	4	7	**3**	4	4	4	4	5	6	✗	1.33	2/4

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	dfs	max	num	ev	result	factor	#
fuzix_difftime_difftime	74	73	**1**	**1**	**1**	2	2	2	**1**	**1**	**1**	**1**	2	2	2	**1**	2	2	2	=	1.00	1/2
fuzix_fgets_fgets	53	58	**3**	**3**	**3**	**3**	4	**3**	**3**	**3**	5	**3**	**3**	5	6	5	5	4	4	✗	1.33	2/4
fuzix_filesys_filename	45	48	**3**	**3**	**3**	**3**	**3**	**3**	**3**	4	4	4	**3**	**3**	**3**	4	**3**	**3**	4	=	1.00	1/2
fuzix_filesys_getinode	52	57	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	5	**3**	**3**	**3**	**3**	5	**3**	**3**	**3**	=	1.00	1/2
fuzix_filesys_i_open	129	143	**3**	**3**	**3**	**3**	5	4	**3**	7	5	7	**3**	6	5	6	5	5	10	✗	1.67	3/5
fuzix_filesys_newfstab	20	21	**2**	**2**	**2**	**2**	3	**2**	**2**	**2**	3	3	**2**	3	3	3	3	**2**	**2**	=	1.00	1/2
fuzix_filesys_srch_mt	31	33	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	4	**3**	**3**	**3**	**3**	4	=	1.00	1/2
fuzix_gethostname_gethostname	30	31	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	3	**2**	3	3	3	3	**2**	4	=	1.00	1/2
fuzix_getpass__gets	31	35	**3**	**3**	**3**	**3**	4	**3**	**3**	**3**	4	**3**	**3**	4	4	4	4	4	6	✗	1.33	2/2
fuzix_inode_rwsetup	77	83	**2**	**2**	**2**	**2**	3	**2**	**2**	3	3	3	3	4	4	5	3	3	4	✗	1.50	2/3
fuzix_malloc___insert_chunk	104	116	**3**	**3**	**3**	**3**	6	**3**	**3**	6	4	6	**3**	**3**	5	5	4	5	5	✗	1.33	2/4
fuzix_nanosleep_clock_nanosleep	110	121	**3**	**3**	**3**	**3**	4	**3**	**3**	5	5	5	**3**	4	4	4	4	4	9	✗	1.33	2/3
fuzix_process_getproc	32	35	**2**	**2**	**2**	**2**	3	**2**	**2**	3	3	3	**2**	3	3	3	3	3	3	✗	1.50	2/2
fuzix_qsort__lqsort	89	94	**3**	**3**	**3**	**3**	5	**3**	**3**	4	4	**3**	**3**	4	4	4	4	**3**	4	=	1.00	1/3
fuzix_ran_rand	46	48	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	**2**	**2**	=	1.00	1/1
fuzix_readdir_readdir	60	65	**3**	**3**	**3**	**3**	4	**3**	**3**	4	4	4	**3**	4	4	5	4	4	5	✗	1.33	2/2
fuzix_regexp_regcomp	118	129	**2**	**2**	**2**	3	4	3	**2**	6	4	6	3	3	3	5	4	4	7	✗	2.00	3/4
fuzix_se_ycomp	83	96	**3**	**3**	**3**	**3**	6	4	**3**	5	5	5	**3**	4	4	6	**3**	4	4	=	1.00	1/4
fuzix_setbuffer_setbuffer	43	44	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	**2**	**2**	3	4	=	1.00	1/2
fuzix_setenv_setenv	122	131	**3**	**3**	**3**	**3**	**3**	**3**	**3**	4	4	6	**3**	5	5	5	4	5	6	✗	1.33	2/4
fuzix_stat_statfix	52	51	**1**	**1**	**1**	2	2	2	**1**	**1**	**1**	**1**	2	2	2	**1**	2	2	2	=	1.00	1/2
fuzix_syscall_fs2__fchdir	22	22	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	=	1.00	1/1
fuzix_syscall_fs2_chown_op	27	28	**2**	**2**	**2**	**2**	3	**2**	**2**	**2**	3	**2**	**2**	3	3	3	3	3	3	✗	1.50	2/2
fuzix_syscall_proc__time	48	49	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	3	**2**	3	3	3	3	**2**	3	=	1.00	1/2
fuzix_sysconf_sysconf	142	162	**3**	**3**	**3**	**3**	**3**	**3**	**3**	20	8	20	**3**	**3**	**3**	16	11	**3**	8	=	1.00	1/3
fuzix_tty_tty_read	123	137	**4**	**4**	**4**	**4**	8	**4**	**4**	6	6	7	**4**	5	5	9	5	5	7	✗	1.25	2/5
fuzix_usermem_ugets	24	25	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	3	3	**2**	**2**	**2**	3	=	1.00	1/2
fuzix_vfscanf_vfscanf	587	668	**6**	7	**6**	**6**	8	**6**	**6**	22	13	22	**6**	8	9	12	11	10	29	✗	1.67	5/6
games120	119	423	29	28	28	**25**	37	**25**	29	33	33	34	28	31	31	32	30	32	32	✗	1.20	4/7
games120	120	638	46	46	46	**39**	62	**39**	46	51	42	41	45	46	46	43	48	40	49	✗	1.03	2/7

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	dfs	max	num	ev	result	factor	#
GeneralizedPetersenGraph_10_4	20	30	**6**	**6**	**6**	**6**	10	**6**	**6**	**6**	8	**6**	**6**	**6**	7	**6**	**6**	7	8	=	1.00	1/4
GNP_20_10_0	20	28	**4**	5	5	**4**	**4**	**4**	**4**	5	5	5	**4**	5	**4**	5	**4**	**4**	6	=	1.00	1/2
GNP_20_10_1	18	23	**3**	**3**	**3**	**3**	4	**3**	**3**	**3**	**3**	**3**	**3**	4	4	4	4	4	4	✗	1.33	2/2
GNP_20_20_0	20	46	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	7	7	**6**	7	7	**6**	8	7	7	=	1.00	1/2
GNP_20_20_1	20	48	**7**	**7**	**7**	**7**	8	**7**	**7**	8	9	9	**7**	8	8	8	8	8	8	✗	1.14	2/3
GNP_20_30_0	20	56	9	**8**	9	**8**	9	**8**	9	10	10	11	**8**	10	10	9	9	9	9	✗	1.12	2/4
GNP_20_30_1	20	63	**8**	**8**	**8**	**8**	9	**8**	**8**	9	10	10	**8**	**8**	9	9	10	9	10	✗	1.12	2/3
GNP_20_40_0	20	78	**10**	**10**	11	**10**	11	**10**	**10**	11	12	13	**10**	11	11	12	11	11	11	✗	1.10	2/4
GNP_20_40_1	20	71	**8**	**8**	**8**	**8**	**8**	**8**	**8**	11	11	11	**8**	**8**	**8**	9	9	9	9	✗	1.12	2/2
GNP_20_50_0	20	91	**10**	**10**	**10**	**10**	12	**10**	**10**	11	13	14	**10**	11	13	11	12	11	11	✗	1.10	2/5
GNP_20_50_1	20	106	**13**	**13**	14	**13**	15	**13**	**13**	**13**	14	14	**13**	**13**	**13**	**13**	14	14	**13**	=	1.00	1/3
GoethalsSeidelGraph_2_3	16	72	**11**	**11**	**11**	**11**	12	**11**	**11**	12	13	12	**11**	**11**	**11**	12	12	**11**	12	=	1.00	1/3
GoldnerHararyGraph	11	27	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	4	4	**3**	4	5	=	1.00	1/2
GossetGraph	56	756	44	44	50	44	45	44	44	**43**	**43**	**43**	49	49	44	**43**	**43**	44	45	=	1.00	1/5
graph09	458	1667	123	127	125	**118**	161	**118**	122	148	138	137	124	121	124	162	157	136	179	✗	1.15	8/11
GrayGraph	54	81	14	14	14	13	17	**12**	14	**12**	15	**12**	14	14	14	16	14	13	15	✗	1.08	2/5
GrotzschGraph	11	20	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	7	6	**5**	**5**	**5**	**5**	**5**	**5**	**5**	=	1.00	1/3
HallJankoGraph	100	1800	85	85	88	87	90	87	85	87	93	87	87	87	87	**84**	85	88	86	✓	0.99	1/5
HanoiTowerGraph_4_3	64	168	**16**	**16**	**16**	**16**	26	**16**	**16**	19	17	18	**16**	**16**	**16**	**16**	19	19	22	=	1.00	1/5
HararyGraph_6_15	15	45	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	8	8	8	8	8	7	8	✗	1.17	2/2
HarborthGraph	52	104	**5**	6	**5**	**5**	9	**5**	7	8	6	8	6	6	6	9	10	6	8	✗	1.20	2/5
HarriesGraph	70	105	19	19	18	18	25	**17**	18	19	24	19	18	18	**17**	**17**	**17**	18	20	=	1.00	1/5
HeawoodGraph	14	21	6	6	6	**5**	6	**5**	6	**5**	6	**5**	6	**5**	**5**	**5**	**5**	6	6	=	1.00	1/2
HigmanSimsGraph	100	1100	75	77	78	77	77	77	78	76	90	76	77	77	76	77	**72**	77	81	✓	0.96	1/5
HoffmanGraph	16	32	7	7	7	7	10	7	**6**	7	8	7	7	**6**	**6**	7	**6**	7	8	=	1.00	1/4
HoffmanSingletonGraph	50	175	29	29	29	27	27	27	28	27	34	27	28	29	29	27	27	**26**	27	✓	0.96	1/4
homer	403	1029	27	27	28	**25**	34	**25**	27	36	30	40	27	31	29	48	36	34	64	✗	1.36	7/9
huck	69	297	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	11	12	**10**	13	=	1.00	1/1
HyperStarGraph_10_2	45	72	**8**	**8**	**8**	**8**	**8**	**8**	**8**	14	10	**8**	**8**	**8**	**8**	**8**	13	**8**	9	=	1.00	1/3
IcosahedralGraph	12	30	7	7	7	**6**	8	**6**	7	**6**	**6**	**6**	7	7	7	**6**	**6**	7	**6**	=	1.00	1/3

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	dfs	max	num	ev	result	factor	#
inithx.i.3-pp	196	2185	29	29	30	**26**	32	**26**	29	80	46	80	29	29	29	37	35	35	48	✗	1.35	5/6
inithx.i.2-pp	220	4165	**28**	**28**	36	**28**	34	**28**	**28**	185	49	181	**28**	29	29	54	78	31	80	✗	1.11	3/7
inithx.i.2	299	5162	**28**	**28**	36	**28**	31	**28**	**28**	243	50	243	**28**	34	34	53	112	34	55	✗	1.21	3/6
inithx.i.1	309	7585	58	58	55	**50**	54	**50**	58	255	60	255	56	54	54	78	113	54	121	✗	1.08	2/7
inithx.i.1-pp	317	12720	**56**	**56**	84	**56**	**56**	**56**	**56**	**56**	61	**56**	**56**	**56**	**56**	74	176	61	102	✗	1.09	2/3
inithx.i.2-pp	363	8897	35	35	49	**31**	41	**31**	35	54	43	44	35	35	35	35	86	35	68	✗	1.13	2/7
inithx.i.1	519	18707	**56**	**56**	89	**56**	**56**	**56**	**56**	295	**56**	488	**56**	**56**	58	74	189	63	99	✗	1.12	3/5
inithx.i.2	558	13979	35	35	56	**31**	35	**31**	35	243	43	527	35	39	42	36	188	45	90	✗	1.16	3/8
inithx.i.3	559	13969	35	35	56	**31**	35	**31**	35	244	43	528	35	40	41	36	189	43	87	✗	1.16	3/8
jean	70	184	**7**	8	8	**7**	11	**7**	**7**	10	9	10	8	10	9	13	9	9	11	✗	1.29	3/5
jean	77	254	**9**	**9**	**9**	**9**	**9**	**9**	**9**	11	11	11	**9**	13	13	13	10	10	12	✗	1.11	2/3
JohnsonGraph_10_4	210	2520	157	156	153	140	174	140	159	**118**	147	**118**	152	158	154	129	126	124	130	✗	1.05	2/11
KittellGraph	23	63	**8**	**8**	**8**	**8**	**8**	**8**	**8**	10	**8**	10	**8**	**8**	**8**	10	11	**8**	10	=	1.00	1/2
KneserGraph_10_2	45	630	39	39	41	**35**	40	**35**	**35**	39	42	39	39	39	39	39	39	39	38	✗	1.09	2/5
KneserGraph_8_3	56	280	32	32	34	34	38	34	32	35	42	35	32	34	34	30	36	**29**	32	✓	0.91	1/5
LadderGraph_20	40	58	**2**	**2**	**2**	**2**	14	**2**	**2**	**2**	**2**	**2**	3	4	4	3	4	3	3	✗	1.50	2/4
le450_15a	434	4315	206	195	198	**178**	228	**178**	206	211	200	189	185	191	184	222	200	192	226	✗	1.08	6/11
le450_15a	450	8168	300	305	311	**290**	333	**290**	300	314	335	315	302	306	310	300	295	294	315	✗	1.01	2/11
le450_15a-pp	431	4256	180	182	190	**179**	226	**179**	182	259	189	188	186	198	189	211	195	190	201	✗	1.06	7/10
le450_15b	427	5615	242	235	242	**229**	255	**229**	237	260	258	262	236	246	249	236	234	235	250	✗	1.02	2/11
le450_15b	450	8169	304	307	314	301	326	301	304	320	320	314	300	308	307	**297**	300	303	313	✓	0.99	1/8
le450_15c	445	11776	308	308	314	300	322	300	308	319	319	318	308	311	312	302	300	**298**	308	✓	0.99	1/8
le450_15c	450	16680	376	384	385	**373**	401	**373**	376	384	400	391	383	386	383	376	380	378	386	✗	1.01	2/9
le450_15d	447	9218	244	233	240	**232**	300	**232**	239	**232**	244	253	241	251	260	272	243	238	240	✗	1.03	3/10
le450_15d	450	16750	**375**	379	383	**375**	397	**375**	**375**	385	398	386	379	383	389	379	380	379	377	✗	1.01	2/8
le450_25a	422	5565	207	207	210	**194**	224	**194**	202	211	207	208	203	207	202	213	208	202	208	✗	1.04	2/8
le450_25a	450	8260	265	264	273	254	301	254	265	267	279	270	265	259	268	252	251	**249**	281	✓	0.98	1/10
le450_25a-pp	413	5569	206	204	212	**198**	225	**198**	209	211	208	207	202	204	205	204	202	206	213	✗	1.02	2/11
le450_25b	423	4295	162	160	170	**146**	233	**146**	162	183	173	201	165	162	172	189	177	176	204	✗	1.21	8/10
le450_25b	450	8263	273	276	288	267	317	267	281	261	293	291	272	262	266	**248**	**248**	258	282	✓	0.95	1/12

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	dfs	max	num	ev	result	factor	#
le450_25b-pp	415	4280	**152**	156	158	153	237	153	157	183	167	202	**152**	160	160	183	175	161	198	✗	1.06	7/10
le450_25c	442	9589	**215**	221	231	218	302	218	**215**	225	320	303	228	237	242	253	232	229	257	✗	1.07	6/11
le450_25c	450	17343	365	362	369	360	383	360	365	368	383	368	363	365	370	**346**	347	349	370	✓	0.96	1/8
le450_25d	444	12169	297	295	303	290	318	290	297	301	318	328	297	309	306	288	**286**	**286**	309	✓	0.99	1/9
le450_25d	450	17425	367	367	374	363	383	363	367	363	380	377	367	378	372	360	362	**357**	367	✓	0.98	1/8
le450_5a	438	3018	183	189	208	181	225	**177**	183	236	206	213	192	182	190	214	212	184	205	✗	1.04	5/12
le450_5a	450	5714	323	319	326	315	359	315	323	327	348	325	323	324	314	**308**	315	312	314	✓	0.98	1/10
le450_5b	435	2949	186	193	191	**180**	223	**180**	192	249	198	209	196	189	197	203	202	182	215	✗	1.01	2/12
le450_5b	450	5734	321	323	330	318	354	318	322	328	341	326	323	314	**305**	320	311	311	317	✗	1.02	2/11
le450_5c	440	5177	227	218	218	**203**	254	**203**	227	274	228	300	210	214	218	242	225	211	246	✗	1.04	3/9
le450_5c	450	9803	344	328	343	315	391	315	344	367	377	362	336	**296**	304	341	339	334	347	✗	1.13	5/11
le450_5d	444	6845	277	272	282	**252**	310	**252**	277	307	304	290	272	271	257	287	278	270	288	✗	1.07	3/10
le450_5d	450	9757	329	325	328	299	396	299	329	365	387	361	316	**290**	300	340	335	314	361	✗	1.08	4/11
LjubljanaGraph	112	168	26	26	28	27	35	26	27	29	34	29	28	26	**24**	27	29	28	29	✗	1.12	3/7
mainuk	48	198	**7**	**7**	8	**7**	12	**7**	**7**	**7**	8	**7**	**7**	11	11	10	10	12	13	✗	1.43	3/4
mainuk-pp	9	28	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	7	=	1.00	1/1
MarkstroemGraph	24	36	**4**	**4**	**4**	**4**	**4**	**4**	**4**	6	7	6	5	5	5	7	7	6	**4**	=	1.00	1/4
McGeeGraph	24	36	8	**7**	**7**	**7**	10	8	8	8	9	8	8	**7**	**7**	8	**7**	8	8	=	1.00	1/4
MCSTestGraph	7	11	**2**	**2**	**2**	**2**	**2**	**2**	3	**2**	**2**	**2**	3	3	3	3	3	3	3	✗	1.50	2/2
MCSTestGraph2	9	13	**2**	**2**	**2**	**2**	**2**	**2**	3	**2**	**2**	**2**	3	3	3	3	3	3	4	✗	1.50	2/2
MeredithGraph	70	140	15	**7**	**7**	**7**	15	**7**	**7**	13	13	13	8	**7**	11	15	8	10	11	✗	1.14	2/5
mildew	35	80	**4**	**4**	**4**	**4**	7	**4**	**4**	5	7	5	5	5	5	7	5	6	8	✗	1.25	2/3
miles1000	128	1594	32	32	36	**27**	63	**27**	32	46	48	78	34	37	37	65	49	35	55	✗	1.30	4/9
miles1000	128	3216	54	54	58	**50**	70	**50**	54	54	70	51	59	78	76	65	57	59	69	✗	1.14	4/8
miles1500	128	5198	83	83	91	**77**	82	**77**	83	83	83	83	87	104	104	78	90	82	83	✗	1.01	2/6
miles250	77	196	**8**	**8**	**8**	**8**	12	**8**	**8**	9	9	9	**8**	**8**	11	11	**8**	**8**	10	=	1.00	1/4
miles250	92	327	**9**	**9**	10	**9**	14	**9**	**9**	10	10	13	10	11	11	18	10	10	20	✗	1.11	2/5
miles500	128	1170	27	34	29	**23**	46	**23**	27	29	37	26	28	36	36	36	37	32	45	✗	1.39	6/9
miles750	125	1251	29	29	**28**	**28**	43	**28**	29	36	42	29	29	35	35	53	34	39	40	✗	1.21	3/6
miles750	128	2113	43	40	**38**	40	66	40	43	41	58	45	**38**	53	53	48	50	50	73	✗	1.26	6/8

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	dfs	max	num	ev	result	factor	#
mulsol.i.5-pp	77	974	**29**	**29**	41	**29**	30	**29**	**29**	42	34	42	**29**	**29**	**29**	31	**29**	30	33	=	1.00	1/5
mulsol.i.4-pp	78	1062	**29**	**29**	43	**29**	30	**29**	**29**	46	39	46	**29**	30	34	33	30	30	32	✗	1.03	2/6
mulsol.i.1	100	1725	43	43	47	43	**42**	43	43	50	50	54	43	**42**	**42**	43	46	**42**	50	=	1.00	1/5
mulsol.i.2	101	1233	**29**	**29**	41	**29**	30	**29**	**29**	50	30	50	**29**	**29**	**29**	32	43	30	34	✗	1.03	2/4
mulsol.i.5	102	1224	**28**	**28**	39	**28**	30	**28**	**28**	52	29	52	**28**	**28**	**28**	32	36	30	36	✗	1.07	3/5
mulsol.i.3	102	1233	**29**	**29**	41	**29**	30	**29**	**29**	47	30	47	**29**	**29**	**29**	33	38	30	40	✗	1.03	2/4
mulsol.i.5-pp	119	2556	**31**	**31**	54	**31**	34	**31**	**31**	38	36	38	32	**31**	**31**	**31**	62	**31**	42	=	1.00	1/6
mulsol.i.1	138	3925	**50**	**50**	65	**50**	**50**	**50**	**50**	70	51	107	**50**	**50**	**50**	51	72	52	62	✗	1.02	2/5
mulsol.i.2	173	3885	**32**	**32**	44	**32**	**32**	**32**	**32**	77	44	142	**32**	36	36	**32**	66	44	62	=	1.00	1/5
mulsol.i.3	174	3916	**32**	**32**	45	**32**	**32**	**32**	**32**	77	44	143	**32**	36	36	**32**	68	44	59	=	1.00	1/6
mulsol.i.4	175	3946	**32**	**32**	46	**32**	**32**	**32**	**32**	77	44	144	**32**	36	36	**32**	78	44	52	=	1.00	1/6
mulsol.i.5	176	3973	**31**	**31**	46	**31**	**31**	**31**	**31**	77	44	145	**31**	36	36	32	69	45	65	✗	1.03	2/6
munin1	189	366	**11**	**11**	13	**11**	**11**	**11**	**11**	20	27	31	**11**	12	12	22	28	17	33	✗	1.55	4/6
munin2	1003	1662	**7**	8	**7**	**7**	10	**7**	**7**	47	12	47	**7**	10	9	15	22	31	78	✗	2.14	6/6
munin2-wpp	317	674	**7**	8	**7**	**7**	9	**7**	**7**	10	17	32	8	8	8	16	16	32	61	✗	2.29	5/6
munin3	1044	1745	**7**	**7**	8	**7**	8	**7**	**7**	52	43	52	**7**	10	9	15	42	27	89	✗	2.14	5/6
munin4	1041	1843	**8**	**8**	9	**8**	9	**8**	**8**	29	18	29	**8**	9	11	17	23	53	87	✗	2.12	4/5
myciel2	5	5	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	=	1.00	1/1
myciel3	11	20	**5**	**5**	**5**	**5**	**5**	**5**	**5**	6	6	6	**5**	**5**	**5**	6	6	**5**	**5**	=	1.00	1/2
myciel4	23	71	11	11	11	11	**10**	11	11	15	14	16	11	11	**10**	**10**	16	11	11	=	1.00	1/5
myciel5	46	139	**12**	14	14	13	15	13	**12**	22	15	23	14	13	14	17	17	14	19	✗	1.17	3/6
myciel5	47	236	20	20	20	21	21	21	20	34	24	37	20	**19**	21	28	28	21	24	✗	1.11	3/6
myciel6	94	550	32	**29**	30	**29**	32	**29**	32	60	41	61	**29**	32	32	39	41	38	44	✗	1.31	4/6
myciel6	95	755	**35**	**35**	38	**35**	36	**35**	**35**	76	43	81	**35**	38	39	51	55	39	45	✗	1.11	4/7
myciel7	191	2360	78	78	72	**66**	70	**66**	78	161	85	171	74	73	79	74	113	79	85	✗	1.12	5/10
NauruGraph	24	36	8	8	8	7	12	7	7	8	9	8	8	**6**	**6**	7	8	8	7	✗	1.17	2/5
NonisotropicOrthogonalPolarGraph_3_5	15	60	11	11	11	11	11	11	11	**10**	**10**	**10**	11	11	11	**10**	**10**	**10**	**10**	=	1.00	1/2
NonisotropicUnitaryPolarGraph_3_3	63	1008	55	55	55	55	56	55	**54**	**54**	58	**54**	55	55	**54**	55	**54**	55	55	=	1.00	1/4
OddGraph_4	35	70	15	13	13	14	18	14	16	**12**	18	**12**	13	15	15	14	14	13	14	✗	1.08	2/6
oesoca+	67	208	**11**	**11**	12	**11**	12	**11**	**11**	13	16	13	**11**	14	**11**	13	12	14	16	✗	1.09	2/5

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	dfs	max	num	ev	result	factor	#
oesoca+-pp	14	75	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	13	**11**	**11**	**11**	**11**	=	1.00	1/2
oesoca	39	67	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	4	4	4	4	4	4	10	✗	1.33	2/2
oesoca42	42	72	**3**	**3**	**3**	**3**	**3**	**3**	**3**	4	**3**	4	4	4	4	4	4	4	10	✗	1.33	2/2
PaleyGraph_17	17	68	**11**	**11**	12	**11**	13	**11**	**11**	**11**	13	**11**	**11**	12	12	12	**11**	12	12	=	1.00	1/3
PappusGraph	18	27	**6**	**6**	**6**	**6**	8	**6**	**6**	**6**	7	**6**	**6**	**6**	**6**	7	**6**	7	7	=	1.00	1/3
pathfinder	109	211	**6**	7	**6**	**6**	7	**6**	**6**	7	8	7	7	7	7	10	8	8	9	✗	1.33	3/3
pathfinder-pp	12	43	**6**	7	8	**6**	7	**6**	**6**	**6**	7	7	7	7	7	7	7	7	7	✗	1.17	2/3
PoussinGraph	15	39	7	7	7	**6**	7	**6**	7	7	**6**	7	7	7	7	**6**	7	**6**	7	=	1.00	1/2
queen10_10	100	1470	81	80	84	79	84	79	83	**77**	90	79	83	83	83	78	**77**	79	79	=	1.00	1/7
queen11_11	121	1265	57	60	64	**54**	66	**54**	57	63	58	77	60	59	60	60	68	62	63	✗	1.11	5/9
queen11_11	121	1980	100	101	103	95	104	95	101	93	105	93	99	101	99	95	95	**92**	96	✓	0.99	1/8
queen12_12	144	1750	74	**71**	75	73	82	73	74	73	84	76	**71**	77	75	82	75	77	76	✗	1.06	4/8
queen12_12	144	2596	120	120	124	117	127	117	122	**112**	132	114	122	121	123	116	113	114	115	✗	1.01	2/10
queen13_13	169	2165	75	75	75	**65**	81	**65**	75	69	80	71	72	71	71	82	74	75	75	✗	1.14	5/7
queen13_13	169	3328	145	146	148	137	148	137	147	133	157	**131**	141	139	146	135	135	**131**	138	=	1.00	1/10
queen14_14	196	3526	151	151	154	142	157	142	151	151	165	141	151	154	155	**138**	141	141	142	✓	0.98	1/7
queen14_14	196	4186	169	166	173	160	172	160	168	155	182	156	168	167	167	155	160	**153**	159	✓	0.99	1/10
queen15_15	225	3467	109	110	110	106	122	106	109	106	110	106	110	114	110	102	109	**101**	115	✓	0.95	1/5
queen15_15	225	5180	188	193	195	183	200	183	195	178	203	**175**	190	193	194	182	185	179	186	✗	1.02	3/10
queen16_16	256	4382	141	139	140	134	152	134	141	**128**	141	134	142	141	139	132	137	131	135	✗	1.02	2/7
queen16_16	256	6320	228	224	228	218	231	218	226	205	240	204	224	217	221	210	210	**203**	217	✓	1.00	1/10
queen5_5	25	106	**11**	**11**	12	**11**	13	**11**	**11**	13	13	12	**11**	12	12	14	12	13	12	✗	1.09	2/3
queen5_5	25	160	**18**	**18**	19	**18**	19	**18**	**18**	**18**	20	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	=	1.00	1/3
queen6_6	36	217	20	20	21	**19**	21	**19**	20	20	22	21	20	22	21	**19**	20	20	20	=	1.00	1/4
queen6_6	36	290	28	28	28	**26**	29	**26**	**26**	**26**	30	27	28	**26**	28	27	27	27	27	✗	1.04	2/5
queen7_7	49	388	31	31	31	30	34	30	31	31	34	31	31	32	32	**29**	**29**	30	30	✓	0.97	1/4
queen7_7	49	476	38	38	40	37	41	37	**36**	**36**	40	37	38	39	39	37	37	37	37	✗	1.03	2/6
queen8_12	96	1261	71	70	74	68	75	68	71	69	76	75	70	75	74	**63**	64	**63**	64	✓	0.93	1/7
queen8_12	96	1368	78	76	79	72	81	72	80	72	83	77	78	79	80	70	71	**69**	73	✓	0.96	1/8
queen8_8	64	728	50	49	53	48	55	48	50	**47**	56	49	52	51	50	49	49	48	49	✗	1.02	2/9

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	dfs	max	num	ev	result	factor	#
queen9_9	81	968	59	61	61	**57**	62	**57**	59	58	71	62	62	62	61	**57**	58	**57**	58	=	1.00	1/6
queen9_9	81	1056	63	66	67	65	68	65	63	**61**	72	62	66	65	66	63	62	**61**	63	=	1.00	1/8
random4reg	48	96	15	**14**	15	**14**	18	**14**	**14**	18	**14**	17	15	**14**	15	18	18	15	17	✗	1.07	2/4
RandomBarabasiAlbert_100_2	100	196	13	**12**	13	**12**	15	**12**	13	22	23	21	**12**	13	14	17	17	16	22	✗	1.33	5/7
RandomBarabasiAlbert_100_5	100	475	36	36	37	**35**	42	**35**	36	49	46	45	36	39	40	42	50	44	44	✗	1.20	6/9
RandomBipartite_10_50_3	60	138	**9**	**9**	**9**	**9**	**9**	**9**	**9**	13	11	19	**9**	**9**	**9**	12	12	**9**	12	=	1.00	1/4
RandomBipartite_25_50_1	69	114	**9**	10	10	10	12	10	**9**	18	17	14	10	11	11	14	14	13	17	✗	1.44	5/7
RandomBipartite_25_50_3	75	368	**23**	**23**	24	**23**	27	**23**	**23**	37	34	33	**23**	24	24	29	33	24	25	✗	1.04	2/6
RandomBoundedToleranceGraph_60	60	1168	**30**	**30**	40	**30**	39	**30**	**30**	43	31	**30**	38	42	44	39	37	37	36	✗	1.20	3/8
RandomBoundedToleranceGraph_80	80	1717	**32**	**32**	44	**32**	46	**32**	**32**	37	33	36	36	48	48	51	43	40	40	✗	1.25	5/7
RandomGNM_100_100	76	96	**6**	**6**	7	**6**	9	**6**	**6**	9	10	10	7	7	7	9	9	11	11	✗	1.50	3/4
RandomGNM_250_1000	250	1000	107	108	110	**105**	130	**105**	107	121	134	118	110	111	118	114	116	118	120	✗	1.09	6/9
RandomGNM_500_500	400	483	24	23	24	23	34	24	**22**	32	38	33	25	24	26	37	37	37	46	✗	1.68	9/9
RandomHolmeKim_300_2_2	300	596	29	**26**	**26**	27	35	27	29	45	38	48	27	32	28	39	50	43	59	✗	1.50	8/9
RandomHolmeKim_700_2_2	700	1396	60	60	63	**59**	76	**59**	60	115	96	125	61	61	64	78	96	86	123	✗	1.32	7/9
RandomNewmanWattsStrogatz_100_5_3	100	269	25	26	25	**22**	34	**22**	25	30	26	29	25	23	**22**	32	30	29	30	✗	1.32	5/7
RandomNewmanWattsStrogatz_250_10_3	250	1636	104	114	114	110	159	110	113	123	107	128	112	114	**102**	111	111	110	122	✗	1.08	4/10
RandomTriangulation_800	800	2394	70	62	60	63	196	63	70	57	**50**	58	60	54	55	91	95	86	199	✗	1.72	10/10
RingedTree_10	1023	2043	22	22	22	**21**		**21**	22	257		106	23	22	24	24	96	153	206	✗	1.14	4/6
RingedTree_6	63	123	10	11	10	10	19	10	10	17	17	10	10	10	10	10	11	**9**	11	✓	0.90	1/4
RingedTree_8	255	507	17	**15**	**15**	16	68	16	17	65	65	32	16	16	16	18	28	30	50	✗	1.20	4/6
RKT_100_80_30_0	100	507	**27**	**27**	28	**27**	28	**27**	**27**	31	39	35	**27**	28	28	29	33	28	32	✗	1.04	2/5
RKT_100_90_30_0	98	254	**22**	**22**	**22**	**22**	25	**22**	**22**	29	30	29	**22**	25	24	25	29	24	26	✗	1.09	2/5
RKT_20_40_10_0	20	87	**9**	**9**	**9**	**9**	**9**	**9**	**9**	12	12	12	**9**	10	10	10	11	**9**	10	=	1.00	1/3
RKT_20_40_10_1	20	87	**10**	**10**	**10**	**10**	11	**10**	**10**	11	12	12	**10**	**10**	**10**	**10**	11	**10**	11	=	1.00	1/3
RKT_20_50_10_0	20	73	**9**	**9**	**9**	**9**	**9**	**9**	**9**	11	**9**	**9**	**9**	**9**	**9**	10	**9**	**9**	**9**	=	1.00	1/2
RKT_20_50_10_1	20	73	9	9	10	**8**	11	**8**	11	**8**	10	12	9	10	10	9	9	9	10	✗	1.12	2/5
RKT_20_60_10_0	20	58	**7**	**7**	**7**	**7**	8	**7**	**7**	9	8	9	**7**	**7**	**7**	10	10	**7**	9	=	1.00	1/3
RKT_20_60_10_1	20	58	**8**	**8**	**8**	**8**	9	**8**	**8**	9	12	10	**8**	**8**	**8**	10	9	**8**	9	=	1.00	1/4
RKT_20_70_10_0	20	44	**6**	**6**	**6**	**6**	7	**6**	**6**	**6**	8	7	**6**	**6**	7	7	7	7	8	✗	1.17	2/3

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	dfs	max	num	ev	result	factor	#
RKT_20_70_10_1	20	44	**6**	7	7	7	9	7	7	7	10	7	7	7	7	9	7	7	8	✗	1.17	2/4
RKT_20_80_10_0	20	29	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	5	5	**4**	**4**	5	6	7	**4**	6	=	1.00	1/2
RKT_20_80_10_1	17	29	**5**	**5**	**5**	**5**	6	**5**	**5**	**5**	6	**5**	**5**	**5**	**5**	**5**	6	6	6	=	1.00	1/2
RKT_300_75_30_0	300	2134	**29**	**29**	**29**	**29**	**29**	**29**	**29**	138	50	91	**29**	**29**	**29**	31	56	**29**	39	=	1.00	1/4
RKT_300_90_30_0	293	854	**27**	**27**	**27**	**27**	**27**	**27**	**27**	71	38	46	**27**	**27**	**27**	32	37	32	34	✗	1.19	2/4
RKT_500_80_30_0	499	2907	**29**	**29**	30	**29**	**29**	**29**	**29**	111	46	120	**29**	30	30	37	53	30	42	✗	1.03	2/5
SchlaefliGraph	27	216	23	23	23	**21**	23	**21**	23	**21**	**21**	**21**	23	22	23	**21**	**21**	**21**	**21**	=	1.00	1/3
school1	370	10290	149	151	153	**132**	229	**132**	149	157	144	201	144	139	139	193	159	172	163	✗	1.20	8/9
school1	377	19091	244	246	271	**225**	295	**225**	244	266	303	312	244	255	259	259	242	251	261	✗	1.08	2/10
school1-pp	352	12929	196	196	198	**181**	234	**181**	196	270	236	229	201	208	209	194	192	190	202	✗	1.05	2/10
school1_nsh	337	7696	108	108	110	**90**	226	**90**	108	126	119	132	108	110	110	153	154	159	181	✗	1.70	7/7
school1_nsh	344	14608	214	214	236	**204**	263	**204**	214	237	269	266	218	242	232	212	212	229	241	✗	1.04	2/10
school1_nsh-pp	324	7387	99	119	110	**98**	204	**98**	99	154	122	180	119	101	108	164	168	134	166	✗	1.37	8/10
ship-ship-pp	30	77	**8**	9	**8**	9	**8**	9	**8**	9	10	13	**8**	9	9	11	9	9	12	✗	1.12	2/4
ShrikhandeGraph	16	48	**9**	**9**	**9**	**9**	10	**9**	**9**	10	10	10	**9**	10	10	**9**	**9**	**9**	**9**	=	1.00	1/2
SierpinskiGasketGraph_3	15	27	**3**	**3**	**3**	**3**	4	**3**	**3**	4	**3**	4	4	4	4	5	5	4	4	✗	1.33	2/2
SimsGewirtzGraph	56	280	36	36	37	35	40	36	37	36	45	36	35	37	33	34	**32**	38	37	✓	0.97	1/6
SquaredSkewHadamardMatrixGraph_2	49	588	**40**	41	41	41	44	41	41	41	44	41	**40**	41	41	41	41	41	41	✗	1.02	2/3
SquaredSkewHadamardMatrixGraph_3	121	3630	**109**	110	112	**109**	116	**109**	110	**109**	117	**109**	**109**	**109**	110	111	110	**109**	111	=	1.00	1/5
StarGraph_100	101	100	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	=	1.00	1/1
stdlib_gmtime	117	123	**2**	**2**	**2**	**2**	3	**2**	**2**	3	5	4	3	3	3	3	4	3	6	✗	1.50	2/4
stdlib_mktime	93	97	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	4	=	1.00	1/1
stdlib_print_format	544	609	**4**	**4**	**4**	**4**	6	**4**	**4**	13	17	18	**4**	5	5	5	11	6	17	✗	1.25	2/6
stdlib_sincoshf	110	117	**2**	**2**	**2**	**2**	4	**2**	**2**	4	4	4	3	3	3	4	5	4	6	✗	2.00	3/3
SylvesterGraph	36	90	17	17	17	17	20	17	**16**	17	21	17	17	18	17	17	18	17	17	✗	1.06	2/5
SymplecticDualPolarGraph_4_4	85	850	**64**	**64**	66	**64**	72	**64**	**64**	66	80	66	**64**	**64**	**64**	66	65	**64**	68	=	1.00	1/4
SymplecticPolarGraph_4_4	85	850	64	64	66	64	77	64	65	**63**	73	**63**	64	65	65	67	65	64	67	✗	1.02	2/6
SzekeresSnarkGraph	50	75	**7**	**7**	8	**7**	13	**7**	8	11	11	11	8	**7**	**7**	9	13	11	10	✗	1.29	3/4
TaylorTwographDescendantSRG_3	27	135	20	**17**	19	20	20	20	20	**17**	22	**17**	**17**	20	20	19	**17**	**17**	19	=	1.00	1/4
TaylorTwographSRG_3	28	210	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	25	**22**	**22**	**22**	**22**	23	**22**	**22**	**22**	=	1.00	1/2

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	dfs	max	num	ev	result	factor	#
Toroidal6RegularGrid2dGraph_4_6	24	72	12	11	11	11	14	11	12	10	10	10	11	11	11	10	**9**	10	11	✓	0.90	1/4
Tutte12Cage	126	189	29	30	32	29	34	29	30	**24**	34	**24**	31	29	29	38	32	28	32	✗	1.17	2/6
water	32	123	11	11	12	**10**	13	**10**	11	**10**	11	12	11	16	16	11	14	**10**	12	=	1.00	1/5
WheelGraph_100	100	198	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	49	**3**	**3**	44	=	1.00	1/1
WorldMap	157	318	6	**5**	6	6	11	**5**	6	12	6	9	6	6	6	13	15	9	28	✗	1.80	3/5
zeroin.i.3-pp	49	651	36	36	31	**29**	37	**29**	36	31	32	31	31	33	33	32	33	30	31	✗	1.03	2/6
zeroin.i.3	83	917	29	29	26	**24**	28	**24**	29	37	25	37	30	**24**	**24**	30	26	28	32	✗	1.08	3/7
zeroin.i.2	85	951	29	29	26	**24**	28	**24**	29	41	25	41	29	**24**	26	30	26	28	28	✗	1.08	3/6
zeroin.i.1	126	4100	**50**	**50**	79	**50**	**50**	**50**	**50**	96	52	**50**	**50**	54	54	52	77	54	61	✗	1.04	2/5
zeroin.i.3	157	3540	**33**	**33**	44	**33**	45	**33**	**33**	123	43	133	**33**	35	35	35	91	34	62	✗	1.03	2/7
zeroin.i.2	157	3541	**33**	**33**	44	**33**	45	**33**	**33**	123	43	133	**33**	35	35	35	91	34	60	✗	1.03	2/7

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	result	best
4x12_torusGrid	48	96					[24→11]				[24→9]					✗	8
AhrensSzekeresGeneralizedQuadrangleGraph_3	27	135	[20→19]	[20→19]		[20→18]	[21→19]	[20→18]			[22→19]		[20→18]	[20→19]		✗	17
alarm	37	65									[5→4]			[5→4]	[5→4]	=	4
anna	110	259									[11→10]			[9→8]	[9→8]	=	8
anna	134	423								[15→14]		[17→13]				✗	12
anna-pp	22	148									[16→14]	[13→12]		[14→13]	[14→13]	=	12
BalancedTree_3,5	364	363				[3→1]	[81→26]	[3→1]								=	1
barley	48	126					[10→8]			[9→8]			[9→8]			✗	7
barley-pp	26	78					[9→8]				[10→9]				[9→8]	✗	7
BiggsSmithGraph	102	153					[30→24]				[28→26]					✗	21
BlanusaSecondSnarkGraph	18	27									[7→5]					=	5
BrinkmannGraph	21	42									[11→10]					✗	9
BrouwerHaemersGraph	81	810					[70→62]				[75→63]					✗	54
BubbleSortGraph_5	120	240		[31→28]			[48→33]				[36→32]					✗	23
CameronGraph	231	3465			[193→185]		[186→185]						[190→185]	[184→182]	[184→179]	✗	175
celar02	100	311					[19→11]							[13→11]		✗	10
celar07	200	817					[29→26]			[28→27]					[24→23]	✗	16
Cell120	600	1200					[237→141]				[112→107]		[118→116]			✗	78
ClebschGraph	16	40	[10→9]				[10→9]				[10→9]					✗	8
contiki_collect_send_ack	53	52					[2→1]	[2→1]								=	1
contiki_collect_send_next_packet	26	25					[2→1]									=	1
contiki_contikimac_input_packet	116	127					[5→4]	[4→3]								=	3
contiki_contikimac_powercycle	166	194					[10→9]									✗	5
contiki_ctk_ctk_menu_add	25	27												[3→2]	[3→2]	=	2
contiki_dhcpc_handle_dhcp	276	313								[17→16]		[17→16]				✗	6
contiki_httpd-cfs_send_file	44	48													[5→4]	✗	3
contiki_ifft_ifft	172	180												[5→3]	[4→3]	✗	2
contiki_ircc_list_channel	70	76					[6→5]									✗	3
contiki_lpp_dutycycle	102	114													[6→5]	=	5
contiki_lpp_init	22	21					[2→1]	[2→1]								=	1

Table B.2: [1/10] For a list of what the INDDGO heuristics are, see page 115. The result column shows if a non-optimal value could be improved to be equal to better than the previous best value between all heuristics. The last column contains the best value found by an INDDGO heuristic.

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	result	best
contiki_lpp_send_packet	116	120													[3→2]	=	2
contiki_process_exit_process	72	82					[7→4]							[5→4]	[5→4]	✗	3
contiki_psock_psock_generator_send	61	68					[7→5]									✗	4
contiki_rudolph1_rudolph1_open	27	26					[2→1]	[2→1]								=	1
contiki_rudolph1_write_data	35	36												[3→2]	[3→2]	=	2
contiki_serial-line_process_thread_serial_line_process	72	81					[6→5]									✗	4
contiki_shell-ps_process_thread_shell_ps_process	45	46												[4→3]	[4→3]	✗	2
contiki_shell-rime-debug_recv_broadcast	24	23					[2→1]	[2→1]								=	1
contiki_shell-text_process_thread_shell_echo_process	25	25												[3→2]		=	2
contiki_shell_shell_register_command	42	45					[4→3]									✗	2
contiki_tcpip_eventhandler	98	112												[5→4]		✗	2
contiki_uip_uip_init	26	27											[3→2]	[3→2]	[3→2]	=	2
contiki_uip_uip_unlisten	19	20					[3→2]									=	2
DejterGraph	112	336					[52→48]				[50→47]					✗	39
DesarguesGraph	20	30					[10→7]			[8→7]	[8→7]	[8→7]				✗	6
DodecahedralGraph	20	30					[7→6]									=	6
DoubleStarSnark	30	45					[9→8]									✗	7
DSJC125.9	125	6961					[122→121]					[121→120]		[121→120]		✗	119
DSJR500.1c	221	23512			[217→214]		[216→213]				[218→216]					✗	212
eil51.tsp	51	140					[18→15]									✗	9
FibonacciTree_10	143	142											[3→2]			✗	1
FlowerSnark	20	30					[8→6]									=	6
FoldedCubeGraph_7	64	224					[42→34]				[48→35]					✗	30
FolkmanGraph	20	40					[10→8]									✗	6
FosterGraph	90	135					[23→22]									✗	20
fpsol2.i.1-pp	191	4418								[163→79]		[163→96]				✗	58
fpsol2.i.3-pp	193	2721								[134→89]		[134→89]				✗	28
fpsol2.i.3	206	2645								[161→98]		[161→97]		[41→40]		✗	28
fpsol2.i.1	210	5489								[147→103]		[141→100]				✗	50
fpsol2.i.1-pp	233	10783								[216→82]						✗	66

[2/10]

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	result	best
fpsol2.i.1	269	11654								[216→97]		[238→118]				✗	66
fpsol2.i.3	363	8688								[69→64]		[332→178]				✗	31
fpsol2.i.2	363	8691								[69→64]		[332→178]				✗	31
fuzix_abort_abort	21	20					[2→1]	[2→1]								=	1
fuzix_clock_gettime_clock_gettime	39	40												[3→2]		=	2
fuzix_clock_gettime_div10quickm	30	29					[2→1]	[2→1]								=	1
fuzix_devf_fd_transfer	119	129									[6→5]				[5→4]	✗	3
fuzix_difftime_difftime	74	73					[2→1]									=	1
fuzix_fgets_fgets	53	58					[4→3]							[5→3]	[6→3]	=	3
fuzix_filesys_i_open	129	143												[6→5]		✗	3
fuzix_gethostname_gethostname	30	31												[3→2]	[3→2]	=	2
fuzix_getpass__gets	31	35												[4→3]		=	3
fuzix_inode_rwsetup	77	83												[4→3]	[4→3]	✗	2
fuzix_malloc___insert_chunk	104	116													[5→4]	✗	3
fuzix_nanosleep_clock_nanosleep	110	121													[4→3]	=	3
fuzix_process_getproc	32	35					[3→2]									=	2
fuzix_qsort__lqsort	89	94					[5→4]							[4→3]	[4→3]	=	3
fuzix_regexp_regcomp	118	129				[3→2]										=	2
fuzix_setbuffer_setbuffer	43	44												[3→2]	[3→2]	=	2
fuzix_setenv_setenv	122	131												[5→4]	[5→4]	✗	3
fuzix_stat_statfix	52	51					[2→1]	[2→1]								=	1
fuzix_sysconf_sysconf	142	162								[20→16]		[20→16]				✗	3
fuzix_tty_tty_read	123	137					[8→5]									✗	4
fuzix_usermem_ugets	24	25												[3→2]		=	2
games120	119	423					[37→33]			[33→29]	[33→30]					✗	25
games120	120	638			[46→45]		[62→43]			[51→44]						✗	39
GeneralizedPetersenGraph_10_4	20	30					[10→7]				[8→6]					=	6
GNP_20_20_1	20	48									[9→8]	[9→8]				✗	7
GNP_20_30_0	20	56										[11→9]				✗	8
GNP_20_30_1	20	63								[9→8]	[10→9]	[10→8]				=	8

[3/10]

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	result	best
GNP_20_40_0	20	78									[12→11]	[13→12]				✗	10
GNP_20_40_1	20	71								[11→10]		[11→9]				✗	8
GNP_20_50_0	20	91									[13→12]	[14→13]			[13→12]	✗	10
GNP_20_50_1	20	106			[14→13]		[15→13]					[14→13]				=	13
GoethalsSeidelGraph_2_3	16	72									[13→12]					✗	11
GossetGraph	56	756			[50→44]		[45→44]						[49→45]	[49→45]	[44→43]	=	43
graph09	458	1667					[161→146]				[138→131]					✗	118
GrayGraph	54	81					[17→14]									✗	12
GrotzschGraph	11	20									[7→5]	[6→5]				=	5
HallJankoGraph	100	1800		[85→83]	[88→85]	[87→83]	[90→83]	[87→83]	[85→84]	[87→84]	[93→85]	[87→84]	[87→86]	[87→85]	[87→85]	✓	85
HanoiTowerGraph_4_3	64	168					[26→17]			[19→17]	[17→15]					✓	16
HarborthGraph	52	104					[9→7]									✗	5
HarriesGraph	70	105					[25→20]				[24→19]					✗	17
HeawoodGraph	14	21					[6→5]				[6→5]					=	5
HigmanSimsGraph	100	1100									[90→80]					✗	75
HoffmanGraph	16	32					[10→7]				[8→7]					✗	6
HoffmanSingletonGraph	50	175			[29→28]				[28→27]		[34→27]			[29→27]		=	27
homer	403	1029								[36→35]		[40→36]				✗	25
HyperStarGraph_10_2	45	72								[14→8]						=	8
IcosahedralGraph	12	30	[7→6]		[7→6]		[8→6]		[7→6]					[7→6]	[7→6]	=	6
inithx.i.3-pp	196	2185								[80→50]		[80→50]				✗	26
inithx.i.2-pp	220	4165								[185→70]	[49→45]	[181→68]				✗	28
inithx.i.2	299	5162								[243→121]		[243→86]				✗	28
inithx.i.1	309	7585								[255→169]		[255→107]				✗	50
inithx.i.2-pp	363	8897			[49→47]											✗	31
inithx.i.1	519	18707								[295→89]		[488→189]				✗	56
inithx.i.2	558	13979								[243→83]		[527→189]				✗	31
inithx.i.3	559	13969								[244→83]		[528→189]				✗	31
jean	70	184			[8→7]								[8→7]	[10→8]		=	7
jean	77	254												[13→12]	[13→12]	✗	9

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	result	best
JohnsonGraph_10_4	210	2520	[157→131]	[156→133]	[153→129]	[140→130]	[174→129]	[140→130]	[159→130]		[147→125]		[152→128]	[158→130]	[154→129]	✗	118
KittellGraph	23	63									[8→7]					✓	8
KneserGraph_10_2	45	630			[41→37]		[40→39]			[39→38]	[42→37]	[39→38]			[39→38]	✗	35
KneserGraph_8_3	56	280					[38→32]			[35→32]	[42→31]	[35→32]				✓	32
LadderGraph_20	40	58					[14→3]									✗	2
le450_15a	434	4315	[206→195]		[198→196]		[228→199]			[211→186]		[189→188]		[191→178]	[184→176]	✓	178
le450_15a	450	8168					[333→301]			[314→301]	[335→303]	[315→311]	[302→301]	[306→297]	[310→304]	✗	290
le450_15a-pp	431	4256				[179→176]	[226→199]	[179→176]		[259→191]		[188→187]	[186→185]	[198→190]	[189→178]	✓	179
le450_15b	427	5615					[255→237]			[260→239]	[258→244]	[262→240]		[246→237]	[249→241]	✗	229
le450_15b	450	8169					[326→309]			[320→304]	[320→309]	[314→310]		[308→300]		=	300
le450_15c	445	11776		[308→307]			[322→304]			[319→305]	[319→306]	[318→303]	[308→305]	[311→305]	[312→308]	✗	300
le450_15c	450	16680					[401→384]				[400→383]	[391→384]	[383→379]	[386→381]	[383→381]	✗	373
le450_15d	447	9218					[300→258]			[232→231]				[251→241]	[260→249]	✓	232
le450_15d	450	16750					[397→383]				[398→384]	[386→384]		[383→380]	[389→383]	✗	375
le450_25a	422	5565					[224→201]			[211→203]	[207→203]	[208→204]			[202→201]	✗	194
le450_25a	450	8260					[301→290]				[279→255]	[270→257]	[265→259]	[259→247]	[268→259]	✓	254
le450_25a-pp	413	5569				[198→197]	[225→201]	[198→197]		[211→205]	[208→204]	[207→203]			[205→204]	✓	198
le450_25b	423	4295					[233→175]			[183→180]	[173→161]	[201→191]	[165→161]	[162→159]	[172→167]	✗	146
le450_25b	450	8263				[267→265]	[317→269]	[267→265]		[261→260]	[293→262]	[291→268]		[262→256]	[266→248]	✓	261
le450_25b-pp	415	4280					[237→193]			[183→176]	[167→162]	[202→190]		[160→155]	[160→159]	✗	152
le450_25c	442	9589					[302→245]			[225→220]	[320→232]	[303→250]		[237→232]	[242→233]	✗	215
le450_25c	450	17343									[383→364]	[368→358]		[365→356]	[370→363]	✓	360
le450_25d	444	12169	[297→294]		[303→294]	[290→289]	[318→296]	[290→289]	[297→294]	[301→289]	[318→305]	[328→301]	[297→291]	[309→299]	[306→293]	✓	290
le450_25d	450	17425			[374→369]		[383→369]			[363→359]	[380→361]	[377→363]		[378→364]	[372→356]	✓	363
le450_5a	438	3018					[225→211]			[236→189]		[213→205]		[182→176]	[190→185]	✓	177
le450_5a	450	5714					[359→314]			[327→316]	[348→315]	[325→310]	[323→316]	[324→310]	[314→301]	✓	314
le450_5b	435	2949					[223→211]		[192→178]	[249→185]	[198→195]	[209→194]	[196→191]	[189→182]	[197→185]	✓	180
le450_5b	450	5734					[354→319]			[328→323]	[341→311]	[326→319]		[314→310]	[305→292]	✓	305
le450_5c	440	5177					[254→239]			[274→223]		[300→240]			[218→208]	✗	203
le450_5c	450	9803					[391→349]			[367→349]	[377→353]	[362→344]				✗	296

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	result	best
le450_5d	444	6845					[310→291]			[307→284]	[304→292]	[290→289]	[272→271]	[271→261]	[257→253]	✗	252
le450_5d	450	9757					[396→350]			[365→349]	[387→347]	[361→341]				✗	290
LjubljanaGraph	112	168					[35→28]				[34→27]					✗	24
McGeeGraph	24	36					[10→7]				[9→8]					=	7
MeredithGraph	70	140	[15→13]													✗	7
miles1000	128	1594					[63→49]					[78→54]				✗	27
miles1000	128	3216									[70→54]			[78→58]	[76→63]	✗	50
miles1500	128	5198	[83→80]	[83→81]	[91→80]		[82→77]		[83→80]	[83→78]	[83→77]	[83→78]	[87→81]	[104→77]	[104→77]	=	77
miles250	77	196					[12→9]								[11→9]	✗	8
miles250	92	327					[14→11]							[11→10]	[11→10]	✗	9
miles500	128	1170					[46→41]					[26→25]		[36→30]	[36→31]	✗	23
miles750	125	1251					[43→40]								[35→34]	✗	28
miles750	128	2113	[43→42]				[66→49]		[43→42]		[58→56]			[53→52]	[53→49]	✗	38
mulsol.i.5-pp	77	974			[41→36]					[42→30]	[34→29]	[42→34]				=	29
mulsol.i.4-pp	78	1062			[43→35]					[46→34]	[39→33]	[46→34]			[34→31]	✗	29
mulsol.i.1	100	1725								[50→46]	[50→43]	[54→48]				✗	42
mulsol.i.2	101	1233								[50→42]		[50→39]				✗	29
mulsol.i.5	102	1224								[52→38]		[52→38]				✗	28
mulsol.i.3	102	1233								[47→38]		[47→38]				✗	29
mulsol.i.5-pp	119	2556			[54→51]					[38→37]		[38→37]				✗	31
mulsol.i.1	138	3925								[70→62]		[107→73]				✗	50
mulsol.i.2	173	3885								[77→61]		[142→79]				✗	32
mulsol.i.3	174	3916								[77→61]		[143→76]				✗	32
mulsol.i.4	175	3946								[77→61]		[144→69]				✗	32
mulsol.i.5	176	3973								[77→61]		[145→79]				✗	31
munin3	1044	1745								[52→50]		[52→50]				✗	7
myciel3	11	20								[6→5]	[6→5]					=	5
myciel4	23	71								[15→11]	[14→11]	[16→11]				✗	10
myciel5	46	139								[22→15]	[15→14]	[23→14]				✗	12
myciel5	47	236								[34→23]		[37→22]				✗	19

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	result	best
myciel6	94	550								[60→42]	[41→40]	[61→40]				✗	29
myciel6	95	755								[76→45]	[43→39]	[81→45]				✗	35
myciel7	191	2360								[161→99]	[85→83]	[171→97]			[79→76]	✗	66
NauruGraph	24	36				[7→6]	[12→7]	[7→6]			[9→8]					=	6
NonisotropicOrthogonalPolarGraph_3_5	15	60	[11→10]	[11→10]	[11→10]	[11→10]	[11→10]	[11→10]	[11→10]					[11→10]	[11→10]	=	10
NonisotropicUnitaryPolarGraph_3_3	63	1008	[55→54]	[55→54]	[55→54]	[55→54]	[56→54]	[55→54]			[58→54]					=	54
OddGraph_4	35	70					[18→14]				[18→14]					✗	12
oesoca+-pp	14	75													[13→11]	=	11
PaleyGraph_17	17	68					[13→12]				[13→12]			[12→11]		=	11
PappusGraph	18	27					[8→6]				[7→6]					=	6
pathfinder-pp	12	43			[8→7]											✗	6
PoussinGraph	15	39			[7→6]					[7→6]						=	6
queen10_10	100	1470	[81→80]		[84→78]	[79→78]	[84→81]	[79→78]	[83→79]		[90→79]	[79→78]	[83→81]	[83→79]	[83→79]	✗	77
queen11_11	121	1265					[66→63]			[63→60]		[77→61]				✗	54
queen11_11	121	1980	[100→96]	[101→96]	[103→97]		[104→95]		[101→98]		[105→96]			[101→95]	[99→98]	✗	93
queen12_12	144	1750					[82→81]			[73→68]	[84→74]	[76→69]		[77→73]		✓	71
queen12_12	144	2596	[120→117]	[120→116]	[124→116]		[127→114]		[122→116]		[132→116]		[122→116]	[121→116]	[123→116]	✗	112
queen13_13	169	2165					[81→77]			[69→67]	[80→70]	[71→70]	[72→69]			✗	65
queen13_13	169	3328	[145→134]	[146→138]	[148→138]		[148→132]		[147→139]		[157→136]		[141→134]	[139→136]	[146→137]	✗	131
queen14_14	196	3526	[151→143]	[151→143]	[154→143]		[157→143]		[151→143]	[151→142]	[165→142]		[151→144]	[154→143]	[155→144]	✗	141
queen14_14	196	4186	[169→160]	[166→161]	[173→160]		[172→163]		[168→160]		[182→162]		[168→157]	[167→160]	[167→158]	✗	155
queen15_15	225	3467	[109→102]	[110→100]	[110→104]	[106→102]	[122→104]	[106→102]	[109→102]	[106→101]	[110→103]		[110→102]	[114→104]	[110→102]	✓	106
queen15_15	225	5180	[188→187]	[193→188]	[195→186]	[183→180]	[200→183]	[183→180]	[195→183]		[203→185]		[190→189]	[193→181]	[194→187]	✗	175
queen16_16	256	4382	[141→132]	[139→134]	[140→133]		[152→134]		[141→132]	[128→127]	[141→134]	[134→128]	[142→132]	[141→132]	[139→131]	✓	128
queen16_16	256	6320	[228→214]	[224→213]	[228→212]	[218→210]	[231→211]	[218→210]	[226→214]		[240→209]		[224→214]	[217→211]	[221→208]	✗	204
queen5_5	25	106								[13→12]						✗	11
queen5_5	25	160					[19→18]				[20→18]					=	18
queen6_6	36	217			[21→20]		[21→20]			[20→19]	[22→20]	[21→20]		[22→20]		=	19
queen6_6	36	290	[28→27]	[28→27]	[28→27]		[29→27]				[30→27]	[27→26]	[28→27]		[28→27]	=	26
queen7_7	49	388		[31→30]	[31→30]		[34→30]			[31→30]	[34→30]	[31→30]	[31→29]	[32→30]	[32→30]	✓	30

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	result	best
queen7_7	49	476	[38→37]	[38→37]	[40→36]		[41→37]				[40→38]	[37→36]	[38→36]	[39→37]	[39→38]	=	36
queen8_12	96	1261	[71→66]	[70→64]	[74→65]	[68→64]	[75→65]	[68→64]	[71→66]	[69→64]	[76→65]	[75→63]	[70→63]	[75→65]	[74→66]	✓	68
queen8_12	96	1368	[78→71]	[76→71]	[79→70]	[72→71]	[81→73]	[72→71]	[80→72]	[72→71]	[83→72]	[77→72]	[78→71]	[79→71]	[80→70]	✓	72
queen8_8	64	728	[50→48]		[53→50]		[55→48]				[56→50]	[49→48]	[52→49]	[51→49]	[50→49]	✗	47
queen9_9	81	968	[59→56]	[61→59]	[61→58]		[62→57]		[59→56]		[71→57]	[62→59]	[62→59]	[62→57]	[61→57]	✓	57
queen9_9	81	1056		[66→61]	[67→63]	[65→63]	[68→63]	[65→63]	[63→62]		[72→63]		[66→62]	[65→64]	[66→64]	=	61
RandomBarabasiAlbert_100_2	100	196								[22→20]						✗	12
RandomBarabasiAlbert_100_5	100	475								[49→43]	[46→40]	[45→44]		[39→38]		✗	35
RandomBipartite_10_50_3	60	138								[13→12]		[19→13]				✗	9
RandomBipartite_25_50_1	69	114								[18→13]	[17→13]					✗	9
RandomBipartite_25_50_3	75	368								[37→32]	[34→32]	[33→30]				✗	23
RandomBoundedToleranceGraph_60	60	1168			[40→38]					[43→35]				[42→38]	[44→38]	✗	30
RandomBoundedToleranceGraph_80	80	1717					[46→41]						[36→35]	[48→47]	[48→46]	✗	32
RandomGNM_250_1000	250	1000					[130→107]			[121→113]	[134→118]				[118→110]	✗	105
RandomGNM_500_500	400	483					[34→32]			[32→28]	[38→35]					✗	22
RandomHolmeKim_700_2_2	700	1396								[115→103]	[96→86]	[125→110]				✗	59
RandomNewmanWattsStrogatz_100_5_3	100	269					[34→28]									✗	22
RandomNewmanWattsStrogatz_250_10_3	250	1636					[159→112]			[123→113]	[107→106]	[128→120]	[112→111]	[114→106]		✗	102
RandomTriangulation_800	800	2394	[70→68]				[196→120]		[70→68]							✗	50
RingedTree_6	63	123					[19→12]				[17→15]					✗	10
RingedTree_8	255	507					[68→51]				[65→41]					✗	15
RKT_100_80_30_0	100	507									[39→34]	[35→33]				✗	27
RKT_100_90_30_0	98	254										[29→28]				✗	22
RKT_20_40_10_0	20	87								[12→10]	[12→10]	[12→10]				✗	9
RKT_20_40_10_1	20	87									[12→11]	[12→10]				=	10
RKT_20_50_10_0	20	73								[11→9]						=	9
RKT_20_50_10_1	20	73					[11→8]		[11→10]			[12→9]		[10→9]	[10→9]	=	8
RKT_20_60_10_0	20	58								[9→8]		[9→8]				✗	7
RKT_20_60_10_1	20	58									[12→8]					=	8
RKT_20_70_10_1	20	44									[10→9]					✗	6

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	result	best
RKT_20_80_10_0	20	29									[5→4]					=	4
RKT_20_80_10_1	17	29									[6→5]					=	5
RKT_300_75_30_0	300	2134								[138→42]	[50→38]	[91→37]				✗	29
RKT_300_90_30_0	293	854								[71→35]	[38→35]	[46→41]				✗	27
RKT_500_80_30_0	499	2907								[111→46]		[120→45]				✗	29
SchlaefliGraph	27	216	[23→21]	[23→21]	[23→21]		[23→21]		[23→21]					[22→21]	[23→21]	=	21
school1	370	10290					[229→186]					[201→186]				✗	132
school1	377	19091			[271→270]					[266→250]	[303→255]	[312→249]	[244→238]		[259→239]	✗	225
school1-pp	352	12929					[234→201]			[270→198]	[236→216]	[229→187]		[208→193]	[209→194]	✗	181
school1_nsh	337	7696					[226→161]			[126→121]						✗	90
school1_nsh	344	14608	[214→213]		[236→219]		[263→226]		[214→213]	[237→213]	[269→200]	[266→202]	[218→207]	[242→229]	[232→207]	✓	204
school1_nsh-pp	324	7387					[204→152]			[154→136]		[180→165]				✗	98
ship-ship-pp	30	77										[13→10]				✗	8
ShrikhandeGraph	16	48					[10→9]			[10→9]	[10→9]	[10→9]		[10→9]	[10→9]	=	9
SimsGewirtzGraph	56	280					[40→36]		[37→35]		[45→36]					✗	33
SquaredSkewHadamardMatrixGraph_2	49	588					[44→41]				[44→41]					✗	40
SquaredSkewHadamardMatrixGraph_3	121	3630					[116→110]				[117→111]				[110→109]	=	109
SylvesterGraph	36	90					[20→17]				[21→17]					✗	16
SymplecticDualPolarGraph_4_4	85	850					[72→68]			[66→65]	[80→67]	[66→65]				✗	64
SymplecticPolarGraph_4_4	85	850					[77→66]				[73→68]					✗	63
SzekeresSnarkGraph	50	75					[13→10]			[11→10]		[11→10]				✗	7
TaylorTwographDescendantSRG_3	27	135	[20→18]			[20→19]	[20→18]	[20→19]	[20→18]		[22→20]			[20→19]	[20→19]	✗	17
TaylorTwographSRG_3	28	210									[25→22]					=	22
Toroidal6RegularGrid2dGraph_4_6	24	72			[11→10]		[14→10]		[12→10]		[10→9]					✓	10
Tutte12Cage	126	189					[34→32]				[34→31]					✗	24
water	32	123					[13→12]							[16→11]	[16→11]	✗	10
WorldMap	157	318					[11→7]			[12→10]						✗	5
zeroin.i.3-pp	49	651	[36→31]		[31→30]		[37→31]		[36→31]	[31→29]	[32→31]			[33→31]		=	29
zeroin.i.3	83	917								[37→35]		[37→35]				✗	24
zeroin.i.2	85	951								[41→31]		[41→35]				✗	24

[9/10]

Graph	n	m	mind	mult	amd	minf	beta	bmf	mmd	lexm	mcs	mcsm	metm	metn	parm	result	best
zeroin.i.1	126	4100			[79→70]					[96→58]	[52→50]					=	50
zeroin.i.3	157	3540								[123→85]		[133→89]				✗	33
zeroin.i.2	157	3541								[123→85]		[133→89]				✗	33

[10/10]

C

LIST OF AUTHOR'S PUBLICATIONS

Journal publications

2017 Matthew Farrell, Timothy D. Goodrich, Nathan Lemons, Felix Reidl, Fernando Sánchez Villaamil, and Blair D. Sullivan. Hyperbolicity, degeneracy, and expansion of random intersection graphs. To appear in *Internet Mathematics*.

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

2015 Aaron B. Adcock, Erik D. Demaine, Martin L. Demaine, Michael P. O'Brien, Felix Reidl, Fernando Sánchez Villaamil, and Blair D. Sullivan. Zig-zag numberlink is NP-complete. *Journal of Information Processing*, 23(3):239–245, 2015.

Conference and Workshop proceedings

2016 Pål Grønås Drange, Markus Sortland Dregi, Fedor V. Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Felix Reidl, Fernando Sánchez Villaamil, Saket Saurabh, Sebastian Siebertz, and Somnath Sikdar. Kernelization and sparseness: the case of dominating set. In *STACS*, volume 47 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

2015 Pål Grønås Drange, Felix Reidl, Fernando Sánchez Villaamil, and Somnath Sikdar. Fast biclustering by dual parameterization. In *IPEC*, volume 43 of *LIPIcs*, pages 402–413. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

Matthew Farrell, Timothy D. Goodrich, Nathan Lemons, Felix Reidl, Fernando Sánchez Villaamil, and Blair D. Sullivan. Hyperbolicity, degeneracy, and expansion of random intersection graphs. In *WAW*, volume 9479 of *Lecture Notes in Computer Science*, pages 29–41. Springer, 2015.

2014 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In *ICALP*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014.

Jakub Gajarský, Jan Obdrzálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. Finite integer index of pathwidth and treewidth. In *IPEC*, volume 8894 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2014.

2013 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. In *ESA*, volume 8125 of *Lecture Notes in Computer Science*, pages 529–540. Springer, 2013.

2011 Ling-Ju Hung, Ton Kloks, and Fernando Sánchez Villaamil. Black-and-white threshold graphs. In *CATS*, volume 119 of *CRPIT*, pages 121–130. Australian Computer Society, 2011.

Philipp Kranen, Felix Reidl, Fernando Sánchez Villaamil, and Thomas Seidl. Hierarchical clustering for real-time stream data with noise. In *SSDBM*, volume 6809 of *Lecture Notes in Computer Science*, pages 405–413. Springer, 2011.

Preprints

2016 Li-Hsuan Chen, Felix Reidl, Peter Rossmanith, and Fernando Sánchez Villaamil. Width, depth and space. *arXiv e-prints*, 2016, arXiv:1607.00945.

Felix Reidl, Fernando Sánchez Villaamil, and Konstantinos Stavropoulos. Characterising bounded expansion by neighbourhood complexity. *arXiv e-prints*, 2016, arXiv:1603.09532.

2014 Erik D. Demaine, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, Somnath Sikdar, and Blair D. Sullivan. Structural sparsity of complex networks: Random graph models and linear algorithms. *arXiv e-prints*, 2014, arXiv:1406.2587.