Exploiting graph structures for computational efficiency

Torstein J. F. Strømme

Thesis for the degree of Philosophiae Doctor (PhD)
University of Bergen, Norway
2020

UNIVERSITY OF BERGEN

Exploiting graph structures for computational efficiency

Torstein J. F. Strømme

Thesis for the degree of Philosophiae Doctor (PhD)
at the University of Bergen

Date of defense: 07.05.2020

Year: 2020

Title: Exploiting graph structures for computational efficiency

Name: Torstein J. F. Strømme

Print: Skipnes Kommunikasjon / University of Bergen

Scientific environment

The author has carried out the research reported in this dissertation at the Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Bergen, Norway.

Acknowledgements

I am deeply grateful for the opportunity to submit and defend this thesis, and I owe everyone who has supported me in this endeavor a very big "thank you." First and foremost, this goes to my advisor Fedor Fomin, who has guided me with a steady hand through the cliffs and waters of research in theoretical computer science. It has been an honor! Secondly, I wish to thank my co-advisor Daniel Lokshtanov, who also made very valuable contributions to my journey; your enthusiasm and stunning ability to dumb things down have been much appreciated.

My co-authors are all simply great people. Thank you, Petr Golovach, Lars Jaffke, Dong Yeap Kang, O-jung Kwon, and Dimitrios M. Thilikos, for what I have learned from you, and everything you have brought to this thesis. A very special thanks in this category also goes to Jan Arne Telle; you have for all practical purposes been my third advisor, and I always appreciated our morning discussions, whether they were about width parameters, competitive programming or university politics.

I wish to thank Fredrik Manne, Marthe Bonamy and Robert Ganian for agreeing to review the thesis. I hope you enjoy it!

My years as a PhD candidate at the Department of Informatics in the University of Bergen have been a blast. The corner office in the far end of the Algorithms hallway on the 3rd floor of Høyteknologisenteret has always been a great place to be, thanks to the cool office mates I've had there: Naim Md, Paloma Thomé de Lima, Sritanu Chakraborty, Emmanuel Arrighi, and Emmanuel Sam. You rock!

I also wish to thank the Algorithms group as a whole, past and present. You have all been part of my journey; I am thankful for the friendly spirit you have fostered, and the culture of collaboration, encouragement and mathematical rigor you promote.

The department administration deserves a mention as well. I have received much support and encouragement from you, and I truly appreciate all the efforts you have made to support our initiatives within competitive programming at the department. More than once have you cleared the schedule to make room for a programming contest, even when this required tough negotiations with professors from other departments. You have been a great support in organizing teaching, and you have helped me out with many practical matters. I also wish to thank the IT people, both at the department and university level; for always being flexible and service-minded regardless of how work-intensive a task I gave you.

Thank you, Pinar Heggernes and the Department of Informatics, for believing in me. You gave me responsibilities that were challenging, but which I could still enjoy and learn from.

Finally, I thank my family and friends. You have shaped me to become who I am. A special shout-out goes to Marijn Visscher, who took the time to read and provide valuable

feedback on a thesis which is in a field completely orthogonal to her own. I also want to thank those who have been praying for me, my work and my family during these years; your efforts have for sure made a difference. Thanks you, Sjur Dyrkolbotn; this journey might never have started without your example and encouragement.

Thank you, Maria, for being the best wife in the world. And thank you Arild and Emil, for being the best sons; I have actually been working on this book your entire lives. A defining era in your dad's life is now coming to a close.

Abstract

Coping with NP-hard graph problems by doing better than simply brute force is a field of significant practical importance, and which have also sparked wide theoretical interest. One route to cope with such hard graph problems is to *exploit structures* which can possibly be found in the input data or in the witness for a solution. In the framework of parameterized complexity, we attempt to quantify such structures by defining numbers which describe "how structured" the graph is. We then do a fine-grained classification of its computational complexity, where not only the input size, but also the structural measure in question come in to play.

There is a number of structural measures called *width* parameters, which includes tree-width, clique-width, and mim-width. These width parameters can be compared by how many classes of graphs that have bounded width. In general there is a tradeoff; if more graph classes have bounded width, then fewer problems can be efficiently solved with the aid of a small width; and if a width is bounded for only a few graph classes, then it is easier to design algorithms which exploit the structure described by the width parameter.

For each of the mentioned width parameters, there are known meta-theorems describing algorithmic results for a wide array of graph problems. Hence, showing that decompositions with bounded width can be found for a certain graph class yields algorithmic results for the given class. In the current thesis, we show that several graph classes have bounded width measures, which thus gives algorithmic consequences.

Algorithms which are FPT or XP parameterized by width parameters are exploiting structure of the input graph. However, it is also possible to exploit structures that are required of a witness to the solution. We use this perspective to give a handful of polynomial-time algorithms for NP-hard problems whenever the witness belongs to certain graph classes.

It is also possible to combine structures of the input graph with structures of the solution witnesses in order to obtain parameterized algorithms, when each structure individually is provably insufficient to provide so under standard complexity assumptions. We give an example of this in the final chapter of the thesis.

List of papers

This thesis is based on the following publications.

(I) **A width parameter useful for chordal and co-comparability graphs**

Dong Yeap Kang, O-joung Kwon, Torstein J. F. Strømme and Jan Arne Telle. In *Theoretical Computer Science*, Volume 704, 2017, Pages 1–17.

(II) **Mim-width III. Graph powers and generalized distance domination problems**

Lars Jaffke, O-joung Kwon, Torstein J. F. Strømme and Jan Arne Telle. In *Theoretical Computer Science*, Volume 796, 2019, Pages 216–236.

(III) **Subgraph complementation**

Fedor V. Fomin, Petr A. Golovach, Torstein J. F. Strømme and Dimitrios M. Thilikos. In *Algorithmica* 2020.

(IV) **Time-inconsistent planning: simple motivation is hard to find**

Fedor V. Fomin and Torstein J. F. Strømme. Presented at the *34th AAAI conference*.

Contents

Part I

Background

Chapter 1

Introductions

1.1 Computers and algorithms

Computers have revolutionized the world. One might even argue that they have taken over the world. Luckily for the human race, though, modern digital computers are ridiculously receptive; they can be persuaded to do whatever we want them to. All it takes is to describe in detail what logical operations each computer should do, and put that description in the respective computer's memory. This is called to *program* the computer.

In fact, what constitutes a computer can almost be defined in terms of its programmability. Will it accurately follow instructions? Well, then it is a computer! This definition is consistent with the historical use of the term, when a *computer* referred to a job title — a human being performing computations.

Such a computer was perhaps a promising student, employed in the 18th century by an acclaimed mathematician who found mundane calculations to be below their dignity. Or the computer could be an educated woman assisting NASA engineers in the 1950's. A computer's job was sometimes menial and repetitive, such as performing tedious computations; but the answers they reported were important. What the engineers wanted from the computers, then, were essentially two things:

- The computer should find the *correct answer* when asked to solve a problem.
- The computer should solve the problem *as quickly as possible*.

Both these traits could be highly influenced by which computer the engineer chose to hire. However, another factor was even more important: which algorithm the computer used.

An *algorithm* is quite simply a set of instructions for how to solve a problem. It can be written down on a piece of paper, it can be stored electronically as a sequence of 1's and 0's, it can float around in someone's mind, or it can merely exist, with no one but God being aware it does. Typically an algorithm will expect some input and produce some output, see Figure 1.1.

Algorithms have been essential to the work of computers ever since they joined the work force in the early 1600's. One of the founding fathers of modern computer science, Alan Turing, observed that a human computer has "no authority to deviate from [the algorithm] in any detail," and that a digital computer can do essentially everything a human computer

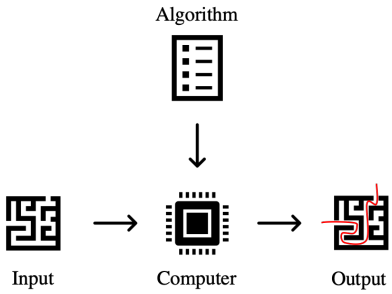


Figure 1.1: An *algorithm* is an instruction set for how to solve a problem.

can (Turing, 1950). Apart for a very large (but constant) factor difference in computational speed, human and digital computers are thus essentially the same.

An algorithm should therefore not be viewed primarily as an instruction set for a digital computer, but rather as a set of instructions that anyone could use in order to solve a problem. However, the reader ought to be aware that not all algorithms are good. Indeed, algorithms can be problematic for several reasons; most critically, not all algorithms produce good answers for the problem at hand. Secondly, some algorithms instruct the computer to perform terribly many steps of computation before reaching a good answer. If the required number of steps is too high, then the computer can simply not use that algorithm to solve the problem within reasonable time.

Looking for algorithms which require the computer to perform few steps, whilst still reaching good answers, is hence a useful endeavor: the realm of computations which we know can be performed in reasonable time will be extended whenever a better algorithm is found. Finding such good algorithms — or proving that no good algorithms are likely to exist for a particular problem — is the purpose of much research in algorithms, and the main topic of this thesis.

1.2 Graph problems

A *graph* is an abstract mathematical object which closely corresponds with what we in the real world call *networks*. We have road networks, railway networks, power grid networks, electric circuit networks, social networks, computer networks, citation networks, task dependency networks, protein networks, climate networks, physiological networks, financial networks, just to name a few. What these networks all have in common, is that "entities" in the network are "related" in various ways. This is exactly the property which is captured by a mathematical graph.

Basically, a graph consists of *vertices* (sometimes called *nodes*) and *edges*, where each edge relate two vertices; in terms of a network, the vertices are the entities, and edges are the connections between entities. See Figure 1.2.

In all of the mentioned domains, there are tasks to be performed and problems to be solved. Many of them can be modeled as abstract graph problems, and good solutions for them can potentially be found with the aid of a computer. For example:

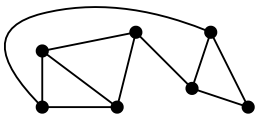


Figure 1.2: A simple graph

Path-finding A fire broke out in the underground coal mine were you work, and you need to evacuate; however, you can not travel through your usual tunnels because of the fire. Is there another way out? Modeled as a mathematical graph, we can let each vertex represent an intersection (that is not yet on fire), and each edge represent a traversable tunnel which connects two intersections.

Cave design In order to have sufficient escape routes in the underground coal mine, the authorities decided that it should be possible to reach the exit from any point in the mine if a fire breaks out at a different point (possibly an intersection). To meet this demand, you sadly need to dig more tunnels, which can be quite expensive. You only have a budget of k; can you afford to build some tunnels such that the requirement is met? This problem is known for abstract graphs as *weighted biconnectivity augmentation*.

Data cleaning In a high-profile chemistry experiment, you made n observations; however, some of the observations are contradictory. Can you remove at most k observations from your data set such that there are no contradictions among the remaining observations? This problem can be modeled as a graph where each observation is a vertex, and if two observations are contradictory, then the corresponding vertices are related by an edge. The problem on the abstract graph is known as *vertex cover*.

As we see, graphs can be used to model a variety of scenarios and problems. Some of these problems are *easy*, in the sense that there exist good algorithms that quickly compute correct answers to the problems. Other graph problems are *hard*, because there are no such algorithms.

1.3 P versus NP

One of the most famous open problems in computer science and even in mathematics as a whole, is whether $\mathsf{P} = \mathsf{NP}$. Roughly speaking, we can think of P as the class of all problems for which there exist good algorithms, whereas NP is the class of problems where a solution can be *verified* by a good algorithm[1] — the question is whether these two classes of problems

[1] Many problems exhibit the property that a *witness* of the solution can be concisely described. Verification is the operation of checking that a witness is "telling the truth." For instance, consider the problem of path-finding: the input is a maze with one entry and one exit, and the question is whether it is possible to move through the maze from entry to exit. If the answer is yes, a red line can be drawn through the maze highlighting the path. This line is a witness. Given the red line, it is easy to check whether it is a valid solution — simply trace the line with your finger and check that it never cross a wall. Because we could verify the solution quickly (with a "good" algorithm), this problem is in NP. We remark that for the path-finding problem we actually know a good algorithm for the full problem as well, hence the problem is also in P. For many problems, however, a good algorithm for verification is known, whereas a good algorithm for the full problem is not.

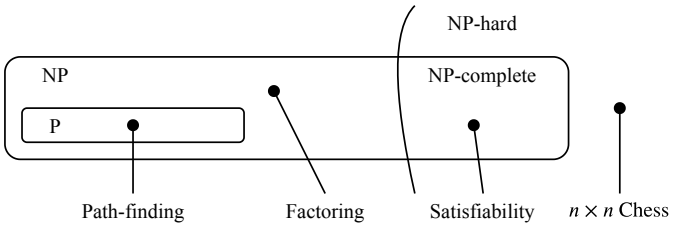


Figure 1.3: Classes P and NP under standard complexity assumptions.

really are the same class, or whether P is strictly contained in NP.

While it is widely believed that $\mathsf{P} \neq \mathsf{NP}$ (see Figure 1.3), nobody has yet been able to prove so. However, in a remarkable result by Cook (1971) and Levin (1973) it is shown that if a problem called Satisfiability (Sat) can be solved with a good algorithm, then *every* problem in NP can be solved with a good algorithm, and hence $\mathsf{P} = \mathsf{NP}$. Sat is the first of what we call NP-*complete* problems. These are in a sense the hardest problems in NP: find a good algorithm for one of them, and it will automatically yield good algorithms for all problems in NP.

One might at first think that NP-complete problems must be very special problems, since they can be used to solve every problem in NP. However, the variety of NP-complete problems is vast. It includes problems such as the cave design and data cleaning problems mentioned in Section 1.2, as well as thousands of other problems from vastly different domains: circuit design, system architecture, scheduling, mathematical proofs, drug development, city planning and large Sudoku's are just a few examples of NP-complete problems.

These problems represent real challenges that humanity is faced with on a daily basis. Hence, even if the problems are hard, we have a strong incentive to cope with them. Some mechanisms with which to cope with such problems are *approximation*, *heuristics* and *probabilistic algorithms*. With these approaches, we are not guaranteed to find an optimal solution in reasonable time, but at least we are often able to find something not too bad.

Another approach, which is what we are investigating in the current thesis, is that of exploiting *structures* in the problem input to speed up computation. For example in the data cleaning example in Section 1.2: if we can organize the observations from our chemistry experiment according to when and where they were observed, this might reveal some underlying structure such that we can make additional assumptions to aid our algorithm — two observations can for example not be contradictory if they are far apart in time.

Structures can sometimes be *parameterized*. This entails defining numerical values that describes "how structured" the input or solution is. In such a context, we might provide algorithms for NP-hard problems where the "bad" factors of the runtime are driven by said parameter — in which case the algorithm is good whenever the parameter is bounded.

A parameterized structure can be given explicitly or be implicitly assumed. Returning yet again to our data cleaning example, we can for instance exploit that the answer is *yes* only if the data points in fact have a strong structure — namely that there exists a set of observations of size k such that their removal yields a conflict-free set! For this problem, we do indeed have a good algorithm when k is small.

Coping with NP-hardness by exploiting structures

If a decision problem Q is in the class P there exists an algorithm A such that:

- A can be run on a normal computer,[2] and
- for all problem instances I of Q and for all executions of A on I:
  - A will compute the answer to I correctly, and
  - A will require at most $f(|I|)$ steps, where $f(\cdot)$ is a polynomial function.[3]

On the other hand; if Q is NP-complete, then there does not exist such an algorithm unless $\mathsf{P} = \mathsf{NP}$.

An algorithm designer facing an NP-hard problem has two options: either a) provide a good algorithm which ticks all the boxes above, and thereby prove that $\mathsf{P} = \mathsf{NP}$, or b) provide an algorithm which compromise on one or more of the nice mentioned features. Since the former option has proven to be a tall mountain to climb, we have so far needed to settle with option (b) every time. An obvious way to make such a compromise, and which is commonly done, is to simply not care whether the answer is always correct. There is something slightly unsatisfying about such a strategy, though, and we will not find any such approaches in the current thesis.

A different way to go about it, is to relax the requirement that our algorithm solve *all* instances — we could for instance settle with solving correctly only those instances which satisfy our structural requirement of choice. Another strategy is to relax the requirement that $f(\cdot)$ is a polynomial function; perhaps we only need to solve small instances.

In the current thesis, we primarily approach problems from the perspective of *parameterized complexity*. In this approach, we are in a sense relaxing both the requirement of solving all instances, and also the requirement of letting $f(\cdot)$ be a polynomial function — there is instead a trade-off: for highly structured instances our algorithm runs in polynomial time, but as the structure gradually descend, the algorithm requires more steps and resources to conclude.

For a *graph* to possess a structure, is equivalent to saying that the graph belongs to a *graph class* — the collection of all graphs possessing this structure. From the perspective of parameterized complexity, we can also talk about structural graph parameters and parameterized graph classes. For example, we can discuss the graph class $\mathcal{VC}_k$ containing all graphs with a vertex cover of size at most k, or the class $\mathcal{TW}_k$ of graphs with treewidth at most k.

Many graph problems can be described as the problem where input is a graph G, and we consider a fixed graph class $\mathcal{G}$ and ask: is G a member of $\mathcal{G}$? In such problems, there inherently is a structure — namely the structure given by $\mathcal{G}$. It is not always such that the

[2] When we here refer to a "normal computer," what we really mean is the mathematical model of a *random-access (stored-program) machine* introduced by Cook and Reckhow (1973). This model closely resembles how contemporary digital computers are designed, the von Neumann architecture (Neumann, 1945). They can also be simulated on a (deterministic) Turing machine with a polynomial overhead.

[3] A function f is *polynomial* if it can be written on the form $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ for some constants $a_0, a_1, \ldots a_n$. For example, $g(x) = 3x^2 - 2$ is a polynomial function ($a_2 = 3, a_1 = 0, a_0 = -2$), whereas $h(x) = 1.01^x$ is not. A polynomial function (such as g) will, for sufficiently large inputs, grow slower than an exponential function (such as h) whenever the base of the exponential is strictly larger than 1.

structure of $\mathcal{G}$ is sufficient to provide a good algorithm. However, it could be that the input graph G is known to be a member of another graph class $\mathcal{H}$, in which case we also have another source of structure.

In our first two papers, we are primarily exploiting the structure of the input graph in order to provide parameterized algorithms. In the third paper, we focus on the structure of solution witnesses, and in the fourth paper we combine structural parameters of the input graph and of the solution witness.

1.4 Overview of the thesis

This thesis contains four papers which are addressing NP-hard graph problems through the lens of parameterized complexity and exploiting graph structures:

- In Chapter 7, we investigate a new structural graph parameter, called *special induced matching width*, abbreviated to *sim*-width. We show that *sim*-width has a strong *modelling power*; in other words, there are many graph classes which have bounded *sim*-width. As such, we ought to expect that the analytic power of the parameter — how easy it is to exploit this structure to provide good algorithms — is limited. Still, we show that certain graph classes with bounded *sim*-width also have bounded *maximum induced matching width* (*mim*-width) for which there exists good algorithms for many problems. Since recognizing the *mim*-width of a graph is NP-hard in general, this result yield algorithmic results for those graph classes.

- In Chapter 8 we continue the theme of structural "width" parameters, as we investigate whether a large family of *distance domination* problems can be solved on graphs with bounded *mim*-width. The core result is purely a structural one: namely that taking arbitrary large graph powers does not increase the *mim*-width of the graph by more than a factor 2. This yields several algorithmic consequences.

- In Chapter 9 we investigate a tool for modifying graphs called a *subgraph complement*. Given a graph class $\mathcal{G}$, we ask if one can recognize in polynomial time whether an input graph G is in the graph class $\mathcal{G}^{(1)}$ — the class of graphs that has a subgraph complement in $\mathcal{G}$. We show that this can indeed be done for a handful of graph classes, but also that it is NP-complete when $\mathcal{G}$ is the class of regular graphs.

- In Chapter 10 we attempt to devise parameterized algorithms for the *motivating subgraph* problem. In this problem, we are asked whether a weighted acyclic directed graph G has a subgraph $H \subseteq G$ possessing certain special properties. Whilst NP-complete in general, we are investigating an additional structural parameter on H, for which we are able to give a pseudo-polynomial algorithm whenever the parameter is bounded.

The thesis is organized as follows. In Chapter 2 we give an introduction to the basic tools we are using in the thesis, including notation for set theory and graph theory. In Chapter 3 we give a brief introduction to complexity theory, including parameterized complexity. We

move on to define several graph classes in Chapter 4, before we investigate width parameters and graph decompositions in Chapter 5.

First, however, we give an extended prologue in order to motivate why computational complexity and algorithm design matters.

1.5 Extended prologue

Good algorithms makes a difference. In the upcoming extended prologue we give an illustration of how algorithm design can really matter. The section is intended for an audience with some programming experience, but which is not yet familiar with the field of algorithms. The material we cover in this section is covered in more detail by any standard textbook in algorithms and data structures, e. g. Sedgewick and Wayne (2011).

1.5.1 The menu problem

Let me introduce the problem of finding a *longest increasing subsequence*. Imagine you are running a restaurant where you serve n different dishes. For this year's menu, you decide to order the dishes according to how tasty they are, such that dish number 1 is the least tasty (although still very nice), and dish number n is the tastiest.

Your price scheme can then be described as a list of numbers $x_1, x_2, \ldots, x_n$, where x_i is the price of the i^{th} least tasty dish. However, ordering prices by tastiness in this way reveals that some dishes are both less tasty *and* more expensive than others! Such dishes will for sure never be chosen by any customer. What is the fewest dishes you must remove from the menu in order that the remaining dishes have strictly increasing price?

Rather than asking which dishes to remove, we can equivalently ask which ones to keep. These dishes will then correspond to a *longest increasing subsequence* of $x_1, x_2, \ldots, x_n$.[4] Finding such a sequence can be mundane work, so we want to employ a computer to find it for us. The computer will do nothing, though, unless we also provide it with an algorithm for how to solve the problem. We here provide a first attempt in Algorithm 1.1.

Algorithm 1.1: Longest increasing subsequence (first attempt)

Input: a sequence of numbers $\chi = \langle x_1, x_2, \ldots, x_n \rangle$
Output: the length of the longest increasing subsequences of χ

```
1 let m hold the value 1;
2 for each subsequence ψ of χ do
3     if ψ is an increasing subsequence longer than m then
4         let m hold the length of ψ;
5 return m;
```

[4] We could also model this problem as a graph problem, where there is one vertex for each dish, and we have a directed edge from one vertex to another if the second dish is both tastier and more expensive than the first. The task then becomes to find the longest path in this graph. However, we will not use this perspective here.

Algorithm 1.1 is correct, in the sense that a computer following the instructions carefully will find the optimal answer. It describes the most straightforward strategy we can imagine: try every possible subsequence, check if it is increasing and longer than the best sequence seen previously, and if so, remember its length. We have here omitted the details for how we go from one subsequence to the next, but for the sake of simplicity we assume the computer knows how to do this.

The algorithm is, however, problematic — it requires the computer to perform an enormous amount of steps to conclude. The number of possible subsequences is roughly 2^n, and the algorithm instructs the computer to check all of them. For instance, even if there are only $n = 30$ dishes on the menu, then there are still a whopping $2^{30} = 1\,073\,741\,824$ subsequences to check. Bear in mind that for us to find the *time* required for the computer, the number of steps must be multiplied with the average time spent to perform each step; in reality, operations such as checking whether a sequence is increasing will involve another algorithm that itself contains many steps. But even whilst pretending we can do so in a single step, a better algorithm is called for.

Luckily, a better algorithm does exist. Consider our second attempt in Algorithm 1.2. In this algorithm we define n variables $\ell_1, \ell_2, \ldots, \ell_n$, one for each item on the menu. The purpose of such a variable, say ℓ_i, is to store the length of the longest increasing subsequence ending with the i^{th} item on the menu. Initially, ℓ_1 receives the value 1, which is clearly correct — the longest increasing subsequence ending in the very first item is the subsequence containing that item only. For $i \in \{2, 3, \ldots, n\}$, consider how we may compute ℓ_i, assuming that ℓ_j is already known for all $j < i$. Either the longest subsequence ending at the i^{th} item contains only the single item itself, in which case the correct value for ℓ_i is 1, and all earlier items have prices which are higher than or equal to x_i. Otherwise, there is a "second last" element in the longest sequence ending at i, say j, such that $x_j < x_i$ and $j < i$ (item j is both cheaper and less tasty than item i). We can try all possibilities for this j, and see which of them have the longest subsequences ending at it. We extend the best of them with the i^{th} item.

Algorithm 1.2: Longest increasing subsequence (second attempt)

Input: a sequence of numbers $\chi = \langle x_1, x_2, \ldots, x_n \rangle$
Output: the length of the longest increasing subsequences of χ

```
1 let ℓ_1, ℓ_2, ..., ℓ_n be variables each holding the value 1;
2 for each i in {2, 3 ..., n} do
3     for each j in {1, 2, ..., i − 1} do
4         if x_j < x_i and ℓ_j ≥ ℓ_i then
5             let ℓ_i hold the value ℓ_j + 1;
6 let m be the maximum value held by any of ℓ_1, ℓ_2, ..., ℓ_n;
7 return m;
```

Algorithm 1.2 is also correct, and produces the same answers as Algorithm 1.1. However, the number of steps required by Algorithm 1.2 is appreciably less. For a menu of n items, the computer using Algorithm 1.2 will encounter the if-statement on line 4 exactly $1+2+3+\ldots+(n-1) = \frac{(n)(n-1)}{2}$ times. For $n = 30$, this is only 435; comparing this with Algorithm 1.1, we find that Algorithm 1.2 is more than 2 *million times faster* — even if we assume that asserting

the if-statement is done equally fast for both algorithms (in reality, this is also much quicker in Algorithm 1.2).

As n goes beyond 30, the factor difference of the runtimes become even worse. Merely going to $n = 31$, one more than in the previous paragraph, and Algorithm 1.2 is now more that 4 million times faster than Algorithm 1.1. Hence, saying that one algorithm is x times faster than another does not well capture the difference between the algorithms, since x could be vastly different depending on how large the input is. In the theoretical study of algorithms, we even want to compare the runtime of algorithms as n approaches infinity. In order to compare and classify algorithms according to how their runtime behaves on different inputs, we now enter the realm of computational complexity.

1.5.2 Computational complexity

It makes sense to describe the *runtime* of an algorithm — the number of steps the computer is instructed to perform — as a function f of the input. While the input can have many characteristics which affect runtime, the most natural one is the input *size* in some form or another, usually denoted by n. For example: in Algorithm 1.1 on page 9, we see that lines 1 and 5 are performed once each; lines 2 and 3 are performed 2^n times each; and line 4 is performed *at most* 2^n times, depending on other characteristics of the input. This illustrates that it is often impossible to give an exact function for the number of steps as a function of the input size only, but we can usually give a function describing an *upper bound*; here $f(n) = 3 \cdot 2^n + 2$.

We ought to be careful, however, with what we consider a *step*. Counting the number of required steps is, after all, only interesting if each step takes roughly the same time to perform. In Algorithm 1.1, we note that the step on line 3 involves checking whether a particular subsequence is strictly increasing, which is clearly more time-consuming than for instance line 1, where a variable is set to hold the number 1. So what is a good way of defining a step?

1.5.3 How to define a step

In the context of a contemporary digital computer, each processor core has a fixed set of machine code instructions it understands, and in principle it will execute one instruction every clock cycle. It might therefore be tempting to let one such instruction constitute a single step, such that our counting exactly corresponds to a fixed unit of time.[5] However, this proves difficult for several reasons: the principle that one instruction takes one clock cycle is not true in practice,[6] and the algorithm undergoes several transformations on the way from being abstractly described to becoming machine code; some of these transformations (e. g. compiling) are practically speaking black boxes to most of us, and the resulting machine

[5] In a 5 GHz processor, one clock cycle lasts exactly 0.0000000002 seconds.

[6] For example, there is an important instruction which says "go fetch whatever is in memory location x, and put it here right next to me (in the processor register)". Executing this instruction will potentially require lots of clock cycles; depending on how the operating system has been managing memory, the requested memory location could be physically stored in the cache (a couple of cycles), in RAM (many more clock cycles) or even on the hard drive (hordes of clock cycles). However, for all practical purposes we can assume there is some upper bound on how many clock cycles each operation requires.

code can be very hard to trace and reason over. There certainly are applications where finding exactly how many clock cycles an algorithm require is useful, but that is the topic of a very different dissertation.

In the area of complexity theory, which we are concerned with in the current thesis, we simplify things drastically. We simply say that *anything that can be done within a constant amount of time (and resources)* can constitute one step. Because we do not specify which constant amount we are talking about, this may seem like a too lenient approach; for instance, I may decide that my constant amount of time is a full year. For our purposes, that is perfectly fine — as long as that constant is fixed before we begin our analysis, it will not change our results when the input size approaches infinity.

Let us return to our example of Algorithm 1.1. With the restrictions that only operations which can be performed within constant time are allowed to constitute a step, this means that line 3 can *not* qualify as a single step: regardless of which constant amount of time and resources we allow, there exists an input with a subsequence so large we are unable verify whether it is increasing or not within that constant amount of time.

The nitpicky reader will now point out that also other lines of Algorithm 1.1 are disallowed under our loose definition of a step. For example, in line 4, the variable m is instructed to hold the length of the subsequence ψ; but if ψ is extremely long, then also writing down its length will require many digits. If n goes towards infinity, then also the number of digits will go towards infinity. This will complicate our runtime analysis slightly; if we care to account for it, the resulting analysis of the algorithm will reveal its *bit complexity*.[7]

More commonly, however, we will rudely choose to ignore this by defining a handful of *primitive* operations which we assume to take constant time. Inspired by the instruction sets of real-life digital computers, such operations include moving and storing a value, addition, negation, multiplication, division, logical operations, comparing numbers etc. In order for this assumption to be fair, though, we restrict each variable such that it can not (alone) hold any value that requires more than $c \cdot \log(n)$ bits to represent, where n is the input size and c is some arbitrary constant. The resulting analysis will yield the algorithm's *word complexity*.[8]

Different assumptions for what we are allowed to count as a single step yield different *abstract machines* for which our analysis will be performed. In the current thesis we will focus on word complexity; however, this does not limit our results. Most complexity classes (see Section 3.4), are designed to be agnostic, within reason, as to which abstract machine was used when carrying out the analysis.

1.5.4 Big $\mathcal{O}$ notation

Recall that our requirements for what constitutes a step allows any operation which can be performed in constant time. It follows that also any sequence with a bounded number of steps could also be defined as a single step if we had started with a larger constant at the outset.

[7] A *bit* is a binary piece of information, either 1 or 0.

[8] A word is, for a computer engineer dealing with real-life digital computers, a memory cell which can hold a constant number of bits. The constant is determined by the particular computer architecture; for instance, in 64-bit architectures, a word will contain 64 bits. For our purposes, however, the word size is not constant but instead bounded by $c \cdot \log(n)$, where c is an arbitrary constant and n is the number of bits required to describe the input.

Different steps can thus differ greatly in how much time is required to execute them (though there is a fixed bound on how big the factor difference is). In light of this, constant factors that appear in a function f describing the runtime of an algorithm are somewhat superfluous; the essence of f is in a sense not found in its constant factors.

In complexity theory we commonly use *asymptotic notation* in order to capture this essence of how a function f behaves as the argument n tends towards infinity. The most prominent member of the asymptotic notations is that of *big* $\mathcal{O}$, which we here define as a set of functions:

Definition 1.3 (Big $\mathcal{O}$)**.** Let $g : \mathbb{R} \to \mathbb{R}$ be a function on the reals. Then $\mathcal{O}(g)$ is the set of all functions f for which there exist constants $c_f \in \mathbb{R}_{>0}$ and $b_f \in \mathbb{R}_{>0}$ such that $f(n) \leq c_f \cdot g(n)$ for all $n \in \mathbb{R}_{>b_f}$.

If a function f is in the set $\mathcal{O}(g)$ for some function g, this should be interpreted as "f grows no faster than g." Let us provide an example.

Example. Say we have functions f and g:

$$f(n) = 4n^2 - 13n + 10$$

$$g(n) = n^2$$

We claim that $f \in \mathcal{O}(g)$. In order to prove this, consider the following choice of constants: $c_f = 4 + 0 + 10$ and $b_f = 1$. According to the definition, $f \in \mathcal{O}(g)$ will hold if

$$f(n) \leq c_f \cdot g(n)$$

$$4n^2 - 13n + 10 \leq (4 + 0 + 10)n^2$$

$$4n^2 - 13n + 10 \leq 4n^2 + 0n^2 + 10n^2$$

for all $n \geq b_f = 1$. Comparing term by term, we can observe that the inequality indeed holds for each pair whenever n is at least 1, and hence also for the expressions as a whole.

1.5.5 Runtime analysis and exploiting structures

Recall that Algorithm 1.1 required checking 2^n subsequences, and for each of those subsequences, it was required to check whether the sequence was increasing. We quickly deduce that the latter requires $\mathcal{O}(n)$ steps; hence the runtime of the algorithm is $\mathcal{O}(2^n n)$.

We can contrast that with Algorithm 1.2, which yields a runtime of $\mathcal{O}(n^2)$. The difference is vast: Algorithm 1.1 has an *exponential* runtime, whereas Algorithm 1.2 has a *polynomial* runtime.

Can we do even better? The answer is *yes*; there exists a standard algorithm for the problem which runs in $\mathcal{O}(n \log n)$ time relying on an application of binary search. We will leave out the details of that algorithm here, and ask instead: does there exist a *linear* algorithm (i. e. requiring only $\mathcal{O}(n)$ steps) for the longest increasing subsequence?

Essentially, the answer is *no* (Fredman, 1975). However, what if the input has some special properties? Say, for example, that the items in our sequence of numbers only take k distinct values; can we exploit this structure?

Algorithm 1.4: Longest increasing subsequence (k distinct values)

Input: a sequence of numbers $\chi = \langle x_1, x_2, \ldots, x_n \rangle$ such that there exists a set of reals $V \subseteq \mathbb{R}$ where $x_i \in V$ for all $i \in \{1, 2, \ldots, n\}$, and where V contains at most k numbers.

Output: the length of the longest increasing subsequences of χ

```
1  find the set V and give its elements some order; let v_j denote the j'th value in V
   according to said order;
2  let ℓ_1, ℓ_2, ..., ℓ_k be variables each holding the value 0;
3  for each i in ⟨1, 2, ..., n⟩ do
4      let b_i = 1;
5      let j_i = −1;
6      for each j in ⟨1, 2, ..., k⟩ do
7          if x_i = v_j then
8              let j_i = j;
9          if v_j < x_i then
10             let b_i be the maximum of b_i and ℓ_j + 1;
11     let ℓ_{j_i} hold the value b_i;
12 let m be the maximum value held by any of ℓ_1, ℓ_2, ..., ℓ_n;
13 return m;
```

Consider Algorithm 1.4. First, we find the set $V = \{v_1, v_2, \ldots, v_k\}$ of all distinct values in the input sequence, and give those elements some random order; this can be done in time $\mathcal{O}(n)$ using for instance a hashing set data structure and an array. We proceed on Line 2 to initialize variables $\ell_1, \ell_2, \ldots, \ell_k$ to 0. The purpose of such a variable ℓ_j is to keep track of the length of a longest increasing subsequence which ends on an item with value v_j. Next, we iterate over each element in the input sequence; the variable b_i is intended to hold the length of the longest sequence ending with element i, and j_i is intended to hold the index of the value x_i in V. We next interate over (the index j of) every possible value in V; if the value v_j is less than x_i, then a subsequence ending with value v_j can be extended with the value x_i — we remember it if it is better than what we have seen before.

Algorithm 1.4 is, as the previous algorithms we have seen, correct for every possible input. However, the Algorithm is *parameterized* — its runtime depends crucially on the number of distinct values k. Analyzing the runtime, we find that it requires $\mathcal{O}(n \cdot k)$ steps! If it so happens that our inputs all have very small k, say smaller than some constant, then the algorithm is really fast — asymptotically linear, $\mathcal{O}(n)$. Otherwise, we are better off with the standard $\mathcal{O}(n \log n)$ algorithm.[9]

1.5.6 Practical impact

We end by reflecting on just how big difference a good algorithm can make, by comparing how our different algorithms fare in practice. This is an area which is not discussed any

[9] Actually, a more careful analysis of the standard $\mathcal{O}(n \log n)$ algorithm reveals that the runtime is in fact $\mathcal{O}(n \log k)$; so we should probably use that regardless.

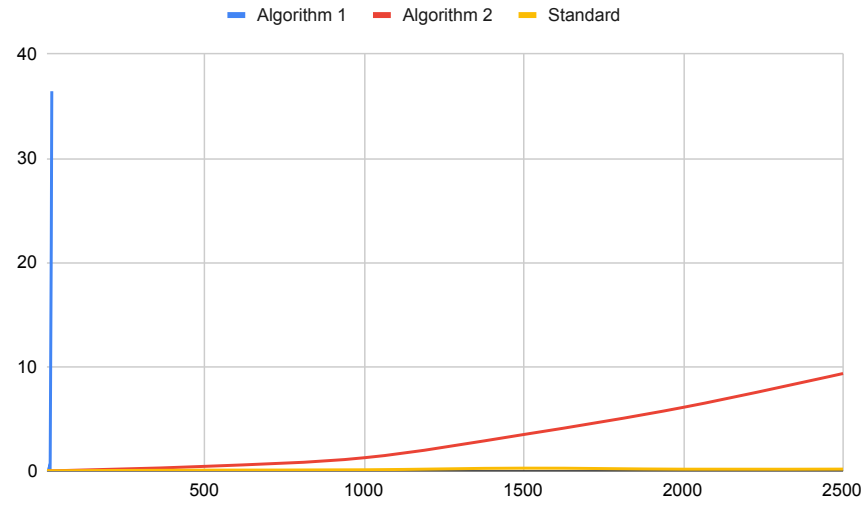


Figure 1.4: Comparison of runtimes for longest increasing subsequence. The unit of the x-axis is the size of the input array, and the unit of the y-axis is the number of seconds required to solve 10 randomized input arrays.

further in the current thesis, but it is worth noting that there *is* a practical side to runtime analysis and algorithm design, and a theoretical runtime analysis will inform the work of a wise practitioner.

In Figure 1.4, we compare the runtimes of Algorithm 1.1, which our theoretical analysis revealed to be $\mathcal{O}(2^n n)$, Algorithm 1.2, which is $\mathcal{O}(n^2)$, and the standard algorithm, which is $\mathcal{O}(n \log n)$. As we expected, the runtime of Algorithm 1.1 skyrockets from the outset, and we can handle at most random inputs of sizes around 30 before we get tired of waiting. Algorithm 1.2 fares much better, but as the input size approaches $n = 3000$, it gets boring waiting for these results too. The standard $\mathcal{O}(n \log n)$ algorithm can handle inputs as big as $n = 10^7$ within the same time frame (which in this informal experiment was roughly one second on a tired laptop).

Chapter 2

Foundations

A *proposition* is a statement which is either TRUE or FALSE. The ultimate goal of a mathematician is to prove that their favorite proposition, called a *theorem*, is TRUE — if necessary under certain assumptions. A *proof*, then, is a sequence of propositions, beginning with the assumptions and ending with the theorem, such that the truth of each proposition in the sequence is a necessary logical consequence of earlier propositions. How a proposition may be derived from earlier propositions is by a *rule of inference*, for example *modus ponens*:

"If A is known to imply B, and A is known to be TRUE, then B must be TRUE."

The rules of inference in mathematics are so blatantly obvious that nobody disagrees with them; if there is even a faint doubt that a rule of inference makes sense, then it will be degraded to an axiom, assumption or different field of study. Now, given a proposition, a rule of inference and pointers to which earlier propositions the rule is being applied on to form the new proposition, it is (supposed to be) straight-forward to verify whether the inference is correct.

This is the process of *peer review* that results in mathematics undergo before being published and accepted. If a proof is done in such excruciating detail that it can be verified by a computer, then it is a *formal* proof; unfortunately, such proofs can be really long, and gives limited intuition for the human reader. Formal proofs will also need to indulge in the mathematical foundations rather than focus on the abstract or "intuitive" meaning given to the mathematical objects in question.

In the current thesis you will find no formal proofs, but rather *rigorous informal proofs*. The principle for such proofs is the same, but it requires more of the reviewers — the author and reviewers need to share a common understanding of the terms and notation which is used and the nature of the mathematical objects in question. We therefore provide a brief background here.

2.1 Set theory

Set theory is the building block on which graph theory and complexity theory are built. We will use a reasonably standard vocabulary of set theory in order to describe our results, and only briefly recapitulate some central notation here. For a deeper introduction, we refer to Stoll (1979).

A *set* is a collection of mathematical objects. These objects can in principle be anything, such as propositions, strings of symbols, truth values, tuples, functions, predicates, relations, other sets, Turing machines, graphs or simply anything that we want to reason about;[1] including *numbers*, which we obviously also have sets for:

$\mathbb{N}$	The set of (positive) natural numbers, $\{1, 2, 3, \ldots\}$.
$\mathbb{Z}$	The set of integers, $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$.
$\mathbb{Q}$	The set of rational numbers. For example, $0, 1, -5, \frac{1}{3}, \frac{-7}{4}$.
$\mathbb{R}$	The set of real numbers. For example: $0, 1, -5, \frac{1}{3}, \frac{-7}{4}, \pi, e, \sqrt{2}$.

A *predicate* is a function $P : A \to \{\textsc{True}, \textsc{False}\}$ where A is a set called the *domain*. We say that P is a predicate on A. In other words, for an element $a \in A$, we get that $P(a)$ is a proposition. For a set A and predicate P on A, we can define the set $A_P = \{a \in A \mid P(a)\}$, which contains every element a of A for which the proposition $P(a)$ is True. Using this notation, we define more sets of numbers we commonly use:

$\mathbb{Z}_{\geq 0}$	$\{x \in \mathbb{Z} \mid x \geq 0\}$	The set of non-negative integers.
$[k]$	$\{x \in \mathbb{N} \mid x \leq k\}$	Defined for $k \in \mathbb{Z}_{\geq 0}$, it denotes the set of natural numbers less than or equal to k.

Assume A and B are sets, and a is a mathematical object. We use the following notations to express propositions/predicates:

$a \in A$	The mathematical object a is a member of A.
$A \subseteq B$	A is a subset of the set B; every element of A is also an element of B.
$A = B$	A and B are the same set, and contain the same elements; $A \subseteq B$ and $B \subseteq A$.
$A \subset B$	A is a proper subset of the set B; $A \subseteq B$ and $A \neq B$.

To form composite propositions, we use the standard logical connectives $\neg$ (not), $\vee$ (or), $\wedge$ (and), $\to$ (implies) and $\leftrightarrow$ (if and only if) from propositional logic. We refer to Walicki (2016) for a deeper introduction.

The *empty set* is denoted by $\emptyset$, and contains no elements. The empty set is a subset of every set. The *cardinality* of a set A is denoted $|A|$, and is a measure of how many elements there are in A. If A is finite, then $|A| \in \mathbb{Z}_{\geq 0}$. The *power set* of a set A is a set which contains every subset of A, denoted $2^A = \{B \mid B \subseteq A\}$. The *choice-$k$ set* of a set A is a set containing all subsets of cardinality k in A, denoted $\binom{A}{k} = \{B \in 2^A \mid |B| = k\}$.

The *cross product* of two sets A and B is the set containing all ordered pairs (a, b) where $a \in A$ and $b \in B$; it is denoted as $A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$. The *set power* of a set A to the power k for $k \in \mathbb{N}$ is the set $A^k = A \times A \times A \times \ldots$ (k times). The *union* of sets A and B is the set containing every element that is in A, B or both, denoted $A \cup B = \{x \mid x \in A \vee x \in B\}$.

[1] In formal set theory we run into strange situations if we place absolutely no restrictions on what a set can contain; for example consider the set of all sets which does not contain itself; this set appears to contain itself if and only if it does not contain itself, and it is known as Russel's paradox. We will, however, leave those worries with the pure set theorists.

The *intersection* of sets A and B is the set containing all elements that are in both A and B, denoted $A \cap B = \{x \mid x \in A \wedge x \in B\}$. The *difference* between a set A and a set B contains the elements which are in A but not in B, denoted $A \setminus B = \{x \mid x \in A \wedge x \notin B\}$.

2.2 Graph theory

A *graph* is essentially a set of *vertices* (sometimes called nodes) and a relation on the set of vertices called *edges*. If the relation is irreflexive, i. e. no element is related to itself, then the graph is *simple*; if the relation is not symmetric, then the graph is *directed*, also called a *digraph*. We will mostly discuss simple undirected graphs, but most of the vocabulary naturally carries over to digraphs as well. We give an overview of our vocabulary here, but for a deeper introduction we refer to Diestel (2017).

2.2.1 Simple graphs

Simple graph A *simple (undirected) graph* G is an ordered pair $G = (V(G), E(G))$, where $V(G)$ is a set of *vertices* and $E(G) \subseteq \binom{V(G)}{2}$ is a set of *edges*.

Vertex A vertex $u \in V(G)$ can in principle be any mathematical object, but for the purposes of graph theory we do not assume it to have any special properties aside from being distinguishable from the other vertices.

Edge An edge $e \in E(G)$ in an undirected graph is a set of two distinct vertices, $e = \{u, v\}$. We abbreviate an edge $\{u, v\}$ to simply uv (or equivalently, vu).

Incidence An edge $e \in E(G)$ is *incident* to a vertex $u \in V(G)$ if $u \in e$. Two edges $e_1, e_2 \in E(G)$ are incident if they share an endpoint, i. e. if $e_1 \cap e_2 \neq \emptyset$.

For a set of edges $F \subseteq E(G)$, the set of vertices which are incident to an edge in F is denoted $V(F) = \bigcup_{e \in F} e$.

Adjacency Two vertices $u, v \in V(G)$ are *adjacent* (and are also called *neighbors*) if there is an edge uv in $E(G)$, i. e. if $\{u, v\} \in E(G)$.

Open neighborhood For a graph G and a vertex $u \in V(G)$, its *open neighborhood* (also called simply its *neighborhood*) is the set of all its neighbors in G, denoted $N_G(u) = \{v \mid uv \in E(G)\}$.

For a set of vertices $X \subseteq V(G)$, the open neighborhood of X is the set $N_G(X) = (\bigcup_{u \in X} N_G(u)) \setminus X$. If the graph G is clear from the context, the subscript $_G$ may be dropped.

Closed neighborhood For a graph G and a vertex $u \in V(G)$, its *closed neighborhood* is the set $N_G[u] = N_G(u) \cup \{u\}$.

For a set of vertices $X \subseteq V(G)$, the closed neighborhood of X is the set $N_G[X] = N_G(X) \cup X$. If the graph G is clear from the context, the subscript $_G$ may be dropped.

Degree For a graph G and a vertex $u \in V(G)$, the *degree* of u is the size of its open neighborhood, denoted $\deg_G(u) = |N_G(u)|$. If the graph G is clear from the context, the subscript $_G$ may be dropped.

Isolated vertex For a graph G and a vertex $u \in V(G)$, the vertex u is *isolated* in G if $\deg_G(u) = 0$.

Universal vertex For a graph G and a vertex $u \in V(G)$, the vertex u is *universal* in G if $N_G[u] = V(G)$, i. e. if it is adjacent to every other vertex.

Subgraph For two graphs H and G, we say the H is a *subgraph* of G, denoted $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. Note that this is overloading the $\subseteq$ operator, so its meaning is slightly different when used on graphs (such as H and G) as opposed to when used on sets (such as $V(G), V(H), E(G), E(H)$).

Induced subgraph For a graph G and a vertex set $X \subseteq V(G)$, the graph G *induced* by X is the graph $G[X] = (X, E(G) \cap \binom{X}{2})$.

Path A graph P is a *path* if there exists a permutation σ over $V(P) = \{u_1, u_2, \ldots, u_n\}$, such that the neighborhood of each vertex is exactly its neighboring vertices in the permutation. In other words, the set of edges for P is exactly $E(P) = \{\sigma(u_i)\sigma(u_{i+1}) \mid i \in [n-1]\}$. The *endpoints* of P are $\sigma(u_1)$ and $\sigma(u_n)$.

The path on n vertices is denoted P_n. The path on n vertices is said to have *length* $n-1$.

Distance The *distance* between two vertices $u, v \in V(G)$, denoted $d_G(u, v)$ is the length of a shortest path with u and v as its endpoints, or ∞ if no such path exist.

Connected graph A graph G is *connected* if for every pair of vertices $u, v \in V(G)$, there exists a path $P \subseteq G$ with u and v as its endpoints.

Connected component A *connected component* in a graph G is an induced subgraph $G[X]$ for a non-empty set $X \subseteq V(G)$ such that a) $G[X]$ is connected, and b) there is no edge $uv \in E(G)$ where $u \in X$ and $v \notin X$.

Isomorphic graphs Two graphs G and H are *isomorphic* if they are representing the same abstract graph structure. Formally, G is isomorphic to H if there exists a bijection $\sigma : V(G) \to V(H)$ such that for every vertex pair $u, v \in V(G)$, there is an edge $uv \in E(G)$ if and only if there is an edge $\sigma(u)\sigma(v) \in E(H)$.

(True) twins For a graph G and distinct vertices $u, v \in V(G)$, we say that u and v are *twins* if $N_G(u) = N_G(v)$. They are *true twins* if $N_G[u] = N_G[v]$.

2.2.2 Modification of graphs

Whenever we have a graph G, we sometimes want to make some modifications to the graph. One example is given above, when we found an induced subgraph of G. There are also many other operations we can perform on a graph in order to obtain new graphs. We give a list of common operations here.

Remove or add vertices For a graph G and a vertex $u \in V(G)$, we denote the graph where we remove the vertex u and its incident edges by $G - u = G[V(G) \setminus \{u\}]$.

For a graph G and vertex set $X \subseteq V(G)$, we denote the graph where the vertices of X and their incident edges are removed from G as $G - X = G[V(G) \setminus X]$

For a graph G and a vertex u, we denote the graph where we add the vertex u (without any incident edges) as $G + u = (V(G) \cup \{u\}, E(G))$.

For a graph G and a set of vertices X, we denote the graph where we add the vertices of X to G as isolated vertices as $G + X = (V(G) \cup X, E(G))$.

Remove or add edges For a graph G and an edge $e \in E(G)$, we denote the graph where e is removed from G by $G - e = (V(G), E(G) \setminus \{e\})$.

For a graph G and a set of edges $F \subseteq E(G)$, we denote the graph where all edges of F are removed from G by $G - F = (V(G), E(G) \setminus F)$.

For a graph G and an edge $e \in \binom{V(G)}{2}$, we denote the graph where e is added to G by $G + e = (V(G), E(G) \cup \{e\})$.

For a graph G and a set of edges $F \subseteq \binom{V(G)}{2}$, we denote the graph where all edges of F are added to G by $G + F = (V(G), E(G) \cup F)$.

(Disjoint) union of graphs For graphs G and H, the *union* of G and H is the graph $G \cup H = (V(G) \cup V(H), E(G) \cup E(H))$.

The *disjoint union* of G and H entails making an isomorphic copy of H, called H' (which is vertex disjoint from G), and then taking the union $G \cup H'$. It is denoted by $G + H$ (or sometimes $G \oplus H$).

Contraction For a graph G and an edge $e \in E(G)$, *contracting* the edge $e = uv$ entails removing the vertices u and v from G and replacing it with a new vertex w whose neighborhood is the set $N_G(\{u, v\})$. Let $X = N_G(\{u, v\})$. Then contracting the edge e in G yields the graph $G/e = G[V(G) \setminus e] \cup (\{w\} \cup X, \{uw \mid u \in X\})$.

Smoothing For a graph G and vertex $u \in V(G)$ of degree 2 whose neighbors v and w are non-adjacent, contracting one of u's incident edges is called *smoothing* the vertex u.

Subdivision For a graph G and an edge $e \in E(G)$, *subdividing* the edge $e = uv$ entails removing e from the set of edges, and adding a new vertex w whose neighborhood is $\{u, v\}$. The resulting graph is $G \uparrow e = G - e + w + uw + wv$. An edge may be subdivided multiple times; for a non-negative integer $n \in \mathbb{Z}_{\geq 0}$, the n-subdivision of an edge e entails applying the subdivision n times on either e or one of the resulting edges created in the process.

Splitting For a graph G and a vertex $u \in V(G)$, *splitting* the vertex u entails adding a vertex v with the neighborhood $N_G(u)$.

Complement For a graph G, its *complement* graph is a graph on the same vertex set where every edge is removed, and every non-edge is added. The complement graph is denoted $\overline{G} = (V(G), \binom{V(G)}{2} \setminus E(G))$.

Graph power For a graph G and an integer $k \in \mathbb{N}$, the *graph power* G^k is the graph on the same vertex set $V(G) = V(G^k)$ where there is an edge in $uv \in G^k$ between distinct vertices u and v if their distance in G is at most k, i. e. if $d_G(u, v) \leq k$.

2.2.3 Generalizations of graphs

Directed graphs

A *directed graph*, also called a *digraph*, is a graph where each edge also has a direction. They differ from undirected graphs in that each edge is represented as an (ordered) 2-tuple, as opposed to an (unordered) set of size 2.

For directed graphs, most of the concepts and operations of undirected graphs may be naturally carried over to the context where edges have direction; for example, (induced) subgraphs, isomorphism and twins are defined pretty much identically as for simple graphs.

Other concepts require some adaptation. For example, a directed path require that all edges are directed in the expected direction.

With respect to connectivity, digraphs require some different notions; we say that a vertex v is *reachable* from a vertex u if there is a path from u to v, or if $u = v$. A *strongly connected component* is maximal induced subgraph where every vertex is reachable from every other vertex.

Other generalizations

Graphs comes in many variations. Each of the concepts below may be mixed and matched to form a new category of graphs.

Weighted graphs A graph or digraph $G = (V(G), E(G))$ can be weighted, either by their vertices or by their edges. Edge weights are most common, which are represented by a weight function $w : E(G) \to \mathbb{R}$ ascribing a certain weight to each edge.

Multigraphs A *mulitgraph* is a generalisation of graphs where there may be multiple edges between two vertices. The difference between a multigraph and an ordinary (simple) graph, is that each edge in a multigraph has a separate *identity* beyond merely connecting two vertices; as such, two distinct edges which both go from u to v can be distinguished, (and can have distinct weights, if the multigraph is weighted).

Hypergraphs A *hypergraph* is a generalization of (undirected) graphs where there is no requirement that an edge have exactly two endpoints. For this type of graph, the edge set $E(G)$ is a subset of $2^{V(G)} \setminus \emptyset$. If all edges have the same cardinality k, then it is called a *k-uniform hypergraph*.

Chapter 3

Computational complexity

As a computer executes an algorithm on some input data, it will start chewing away on its task, performing one step at a time. How many steps the computer will perform before coming to a conclusion is the *runtime* of the algorithm. Finding algorithms that require as few steps as possible is a useful endeavor, since computations may then be performed faster.

Sometimes we are able to find quite good algorithms for the problem at hand, and a modern computer will blaze through huge input data in a split-second. For other problems, however, it seems difficult to find algorithms that perform well; even on small input instances, the number of steps required is so huge that we have no hope that the computer will finish its computations in our lifetime.

We can thus categorize our problems: those that have good algorithms, the *easy* problems; and those that (probably) do not, the *hard* problems.

Categorizing problems according to their difficulty in this way is a central theme in computational complexity theory and its predecessor, computability theory. In this section we give a brief background for this area, but for a deeper introduction, we refer the reader to (Sipser, 2012).

An *algorithm* is a finite set of instructions for how to perform a task in any form; for our purposes, however, we restrict our attention to algorithms which are descriptions of *computational machines* whose purpose is to *compute functions*.

3.1 Problems and computability

Recall that a function $f : X \to Y$ is a map from elements in the set X to elements in the set Y. So if I have some element $x \in X$, I might want to know what the element $f(x)$ is. But even if we defined the function f ourselves, it is not necessarily easy for us to deduce which element x maps to.

For example, say that $X = Y = \mathbb{R}_{\geq 0}$ is the set of non-negative real numbers, and that f is a function expressed as $f(x) = \sqrt{x^3}$. Clearly, f is well-defined — even so, we do not know what $f(4)$ is without some way of *computing* it.

A *computable function* is informally speaking a function for which there exists a procedure by which the image of any input can be found.

According to the famous conjecture called the *Church-Turing thesis*, every computable function can be computed by a *Turing machine* (Copeland, 2019). Since every Turing ma-

chine also computes some function, it will follow that a function is computable if and only if there exists some Turing machine which computes it.

Whenever we refer to a *problem*, then its most general meaning is that we wish to compute the image(s) of some input(s) for a given function which we can precisely articulate. To *solve* the problem is to give an algorithm which correctly computes the function.

A *decision problem* is a question to which the answer for a particular input instance is either *yes* or *no* — the function we want to compute is in other words a *predicate*. Since the predicates on X is in a one-to-one correspondence with the subsets of X, we can think of a decision problem L as a subset of the domain, $L \subseteq X$. The question to answer for an element $x \in X$ is whether $x \in L$ is TRUE or not.

In an *optimization problem*, the input is a set X of elements with an *objective function* $f : X \to \mathbb{R}$, and the task is to find an element $x \in X$ such that $f(x)$ is minimum (or maximum) among all elements in X. Every optimization problem has a decision version, where a number $a \in \mathbb{R}$ is provided along with the input, and the question is whether there exists an element $x \in X$ such that $f(x) \leq a$ (or $f(x) \geq a$).

3.2 Turing machines

A (deterministic) *Turing machine* is a mathematical model introduced by (Turing, 1937); it is formally defined as a collection of sets possessing some special structure. Hence, calling this collections of sets a *machine* is a bit generous, because it is not like the sets will do anything if we only plug them into the power grid.

Turing machines are, however, precise models for a family of devices which *can* be built in practice, using either mechanical parts or electrical circuits — there is a one-to-one correspondence between what such a device will do and configurations of a Turing machine in the mathematical model. Moreover, these devices can simulate exactly the behavior of a modern digital computer, albeit with some polynomial overhead in computation time; and vice versa, a modern digital computer can simulate the behavior of a Turing machine.

Due to space constraints, we will omit the formal definition of a Turing machine here, but refer the reader to (Sipser, 2012; De Mol, 2019) for a thorough introduction. However, we will highlight some of their properties:

- The input to a Turing machine is always a string of symbols $s \in \Sigma^*$ over some finite alphabet Σ. Hence, the domain in which the Turing machine computes a function is Σ^*. We can, however, assume that all the mathematical objects we are working with in this thesis can indeed be described as a finite string of symbols. The *size* of an input s, denoted $|s|$, is the number of symbols in the string.

- For a given input $s \in \Sigma^*$, a Turing machine M will go through some sequence of *steps*. The sequence of steps is uniquely determined by s and M; it could be infinitely long, or it could be finite. If the sequence is finite, we say that the machine *halts* on that input. For a given input s, the number of steps before the machine halts is the *runtime* of M on s.

- When a Turing machine halts, it is in either an *accepting* or a *rejecting* state. If for a string $s \in \Sigma^*$ the machine M halts in an accepting (rejecting) state, we say that M accepts (rejects) s. The set of all strings that M accepts is the *language* of M, denoted $\mathcal{L}(M) = \{s \in \Sigma^* \mid M \text{ accepts } s\}$. For a language $L \subseteq \Sigma^*$, we say that M *recognize* L if $\mathcal{L}(M) = L$. If a machine M recognize a language L and additionally halts on every input, then M *decides* the problem L.

With respect to a decision problem on a domain X, i. e. a subset $L \subseteq X$, a Turing machine will compute the answer if it decides the language $L' = \{s \in \Sigma^* \mid s \text{ is a string representation of some } x \in L\}$.

Let $|s|$ denote the length of a string $s \in \Sigma^*$, and let $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ be a function on the non-negative real numbers. Then a Turing machine M is said to decide the language $\mathcal{L}(M)$ in f time if, for every string $s \in \Sigma^*$, the runtime of M on s is at most $f(|s|)$.

3.3 Nondeterminism

In *deterministic* philosophy, every action is seen as a logical necessity that follows from the current state of affairs. In this world view, everything is determined by cause and effect; free will is merely an illusion. Digital computers as we know them today are perfectly consistent with this world view; they are built from circuits which adhere to the omnipotent physical laws and always perform the same sequence of steps on the same input.[1]

Nondeterminism, on the other hand, is a view that there can be multiple possible courses of action from a given state of the universe. A *nondeterministic Turing machine* is a mathematical model which adheres to such a world view. As opposed to the deterministic Turing machine introduced in Section 3.2, the sequence of steps that follows by executing a given nondeterministic Turing machine M on a certain input string s is not unique; instead, there are multiple such sequences.

A nondeterministic Turing machine M is said to *accept* an input s if *there exists* a sequence which takes M to an accepting state, and is said to *reject* the input if *every* possible sequence ends in a rejecting state. Note that it could also happen that some sequences end up in a rejecting state, whereas others are infinitely long — in these cases, the machine neither accepts nor rejects the input, and does not halt.

The *runtime* of a nondeterministic Turing machine M on an input string s, is defined the length of the *shortest* possible sequence ending in an accepting state (if one exists), or the *longest* possible sequence otherwise. If M does not halt on input s, then the runtime is undefined.[2]

In terms of computability, nondeterministic Turing machines are no more powerful than their deterministic counterparts; a nondeterministic machine can be simulated by a deterministic machine at the cost of a longer runtime. The extra cost in runtime required on deterministic machines, is, however, significant. In fact, comparing the required runtime for solving

[1] In practice there are sources of random behavior in a digital computer, such as memory cells that physically stop working after an unknown quantity of strain, or that execution begins at a certain point in time. According to deterministic philosophy, however, these events are not *random*, they are simply too hard for us to predict.

[2] Keep in mind that nondeterministic Turing machines are *not* models for devices which we know how to build in practice — at least not with respect to how their "runtime" is defined.

problems on deterministic versus nondeterministic Turing machines has become one of the most prominent open problems in mathematics as a whole. It is the classical problem of P versus NP.

3.4 Complexity classes

3.4.1 P and NP

We have already discussed the complexity classes P and NP in our introduction (Section 1.3), where we defined P to be the class of all problems which can be solved by a polynomial-time algorithm, and NP as the class of problems for which a solution can be verified by a polynomial-time algorithm. The traditional definition, however, is that P is defined as the class of problems which can be decided in polynomial time on a *deterministic* Turing machine, and NP consists of those which can be decided in polynomial time on a *nondeterministic* Turing machine. This is how the classes got their names: problems solvable in *polynomial* time (P) and problems solvable in *nondeterministic polynomial* time (NP).

In the previously mentioned result by Cook (1971) and Levin (1973), it is shown that if the problem of Satisfiability (Sat) can be solved in f time on a Turing machine where f is a polynomial function, then *every* problem in NP can also be solved in polynomial time. Because Sat has this remarkable property, we say that it is NP-*hard*. Since the problem happens to be solvable in polynomial time on a nondeterministic Turing machine, it is also in the class NP. A problem which is both in NP and also NP-hard is called NP-complete;[3] recall Figure 1.3 on page 6.

It remains open whether $\mathsf{P} = \mathsf{NP}$, but it is conjectured that it is not — if it is, then every problem in NP, including a wide spectrum of NP-complete problems, will have polynomial-time algorithms. It was estimated that several thousand new problems were proved to be NP-complete each year in the 90's (Papadimitriou, 1997), and investigating historical trends for the term "NP-complete" in scholarly databases suggests that the number might be even higher today. If $\mathsf{P} = \mathsf{NP}$, then all these thousands of problems are solvable significantly faster than what we currently are aware of; showing any one of these problems to be in P will have massive implications.

A more cynical view is that NP-hardness is a useful concept because we do not need to waste our time trying to find polynomial-time algorithms for such problems.

3.4.2 Other complexity classes

There is a multitude of complexity classes which can be defined; for example, given a computational model and a restriction on the runtime (or the amount of space) allowed as a function of the input description size, we obtain such a class. We mention a few of examples that we might refer to later:

[3] Formally speaking, a problem is only NP-complete if it is in NP and is NP-hard with respect to *polynomial-time many-one reductions* (also called *Karp* reductions). In this stricter definition, a problem L is NP-hard if there exist polynomial-time many-one reductions from every problem in NP to L. This holds for Sat. See also Section 3.5 on reductions and hardness.

EXP A decision problem L is in the class EXP if there exists a Turing machine which decides L in $2^{p(n)}$ time where n is the input size, and p is a polynomial function. We know that P is *strictly* contained within EXP, and that NP $\subseteq$ EXP. In other words, if P $=$ NP, then NP $\subset$ EXP, and if NP $=$ EXP, then P $\neq$ NP.

PSPACE A decision problem L is in the class PSPACE if there exists a Turing machine which decides L using only polynomial amount of *space*. We remark that the class PSPACE is known to be contained in EXP, and known to contain NP.

coNP A decision problem L is in the class coNP if it is the complement of a problem in NP; in other words, there is a problem $\overline{L} \in$ NP on the same domain as L such that an instance s is TRUE for L if and only if it is FALSE for $\overline{L}$. An example of a problem which is believed to be in coNP but not in NP, is the TAUTOLOGY problem: in this problem one is given a sentence in propositional logic, and is asked whether all possible structures are models for the formula. This is the complement of the SATISFIABILITY problem, except that the formula is negated; we ask whether all assignments are FALSE rather than asking if there exists an assignment which is TRUE.

While they appear to be the same question (and if we know the answer to one, we immediately know the answer to the other), they are also different in an important regard: verifying the answer. If the answer to an instance of SATISFIABILITY is *yes*, then there exists a small/polynomial size *witness* of this — namely an assignment to the variables which yields the formula TRUE. On the other hand, if the answer is *no*, then we do not know of any small witness.

ZPP A decision problem L is in the class ZPP if there exists a *probabilistic* Turing machine which decides L in polynomial time. We remark that a probabilistic Turing machine is a nondeterministic Turing machine which picks its sequence of steps according to some probability distribution; the definition of *runtime* for such a machine is different from a nondeterministic Turing machine, in that it will be the *expected* runtime according to the probabilities.

We know that P $\subseteq$ ZPP $\subseteq$ NP, however it is unknown whether any of the inclusions are strict.

APX An optimization problem L is in the class APX if there exists a constant-factor approximation algorithm for L which runs in polynomial time on a deterministic Turing machine.

We will see a few more complexity classes in Section 3.6 on parameterized complexity.

Some complexity classes form a *hierarchy* of inclusion. For example, we know that P $\subseteq$ ZPP $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ EXP, but we do not know whether any of the inclusions are *strict*; we know that P $\subset$ EXP, but we do not know which intermediate inclusions are strict and which are not. We usually assume that these inclusions are *all* strict — it the opposite is proved, it will cause a *collapse* in the hierarchy, meaning that some classes of problems are actually the same class.

The complexity classes we have mentioned are defined by which problems *can* be solved under certain restrictions — in order to prove that a given problem L is in a certain complexity

class $\mathcal{C}$, we have to give an *algorithm* for the problem which adheres to the specifications of the given class. For example, we can show that L is in P by providing a polynomial-time algorithm for it.

To show that a problem is *hard*, on the other hand, requires a different tool set which is equally important as algorithms in complexity theory: reductions.

3.5 Reductions and hardness

Building on the result of Cook (1971), Karp (1972) showed that SAT is not the only problem in NP which is NP-hard; in fact, many problems are. Cook's proof was in essence a construction which as input was provided a description of a nondeterministic Turing machine M, and an input string intended for said machine; based on this, a formula φ for SAT is constructed such that it can be satisfied if and only if M will accept the string. This is in a sense to "express" the problem $\mathcal{L}(M)$ as a SATISFIABILITY problem.

Karp took this perspective further, and showed that SAT itself can be expressed as a wide variety of other problems; and just like Cook's construction, the new expression of the problem requires only polynomial time to be computed. To formalize this, Karp introduced the notion of a *reduction*.

- A *polynomial-time many-one reduction* from a decision problem L to a decision problem Q is an algorithm which works in polynomial time with respect to the input size, and takes as input an instance s to the problem L, and returns an instance s' to the problem Q such that s is a *yes*-instance to L if and only if s' is a *yes*-instance to Q.

If there exists such an algorithm for problems L and Q, we write this as $L \leq_P Q$; here, $\leq_P$ reads as "is polynomial-time many-one reducible to." It follows that if Q can be solved in polynomial time, then certainly also L can be solved in polynomial time.

While we will stick with polynomial-time many-one reductions in the current thesis, it is worth being aware that other types of reductions exist. For instance, a natural notion from a programmer's point of view are *Turing* reductions (also called Cook reductions). A (polynomial-time) Turing reduction from a problem A to a problem B is an algorithm for A which call a black box algorithm for B as a subroutine; possibly more than once. Assuming that the calls to B require only polynomial time, then the algorithm as a whole also has this property. A many-one reduction is a special case of a Turing reduction, where there is *one* call to the algorithm for B, and this is the final step of the algorithm.

We say that a problem L is *hard* for a complexity class $\mathcal{C}$ with respect to type x reductions if every problem in $\mathcal{C}$ has a type x reduction to L. We say that the problem is *complete* for $\mathcal{C}$ with respect to type x reductions if additionally the problem is in $\mathcal{C}$. In order to show that a certain problem L is $\mathcal{C}$-hard, it suffices to give a reduction from any $\mathcal{C}$-hard problem to L, since reductions can be chained.

If nothing else is mentioned specifically, we will assume that by a *reduction* we mean a polynomial-time many-one reduction. This is the standard form of reduction with which NP-completeness is most commonly defined with respect to.

3.6 Exploiting structure

We do not expect that NP-hard problems can be solved with a polynomial time algorithm unless there is a collapse of the classes P and NP. However, NP-hard problems exist in the real world, and humanity has come up with algorithms to deal with them as best we can. Some of these algorithms turns out to solve their problem both correctly and quickly on all input instances which occur in practice. To explain this theoretically, we define concepts such as pseudo-polynomial algorithms and parameterized complexity.

3.6.1 Pseudo-polynomial algorithms

The notion of *pseudo-polynomial* algorithms is one example of algorithms which are exploiting the structure of practical instances in order to quickly solve NP-hard problems. These are algorithms which works in polynomial time if the input is encoded in *unary* — that is, if writing down a number x in the input is done using x amounts of space, and not $\log_2 x$ as we would normally expect with a binary encoding. The Subset sum problem is an example of this; while it is NP-hard in general, many instances which occur in practice can be solved efficiently.

Subset sum
Input: A finite set of natural numbers $S = \{a_1, a_2, \ldots, a_n\} \subseteq \mathbb{N}$ and a target $W \in \mathbb{N}$.
Parameter: W
Question: Does there exist a subset $S' \subseteq S$ such that the sum of its elements equals W?

The Subset sum problem can famously be solved by dynamic programming, by creating a 2-dimensional table t with dimension $n \times W$. A cell $t[i, j]$ is supposed to contain the value True if there exists a subset of $\{a_1, a_2, \ldots a_i\}$ such that they sum to j, and False otherwise. For $i = 1$, filling in $t[i, j]$ with the correct value is trivial: it is True for $j = a_1$ and False otherwise. If we fill the table starting from lower values of i, we can assume that $t[i-1, j]$ is correct for all values of j when we want to calculate the value for $t[i, j]$. We can then set the value for $t[i, j]$ as follows: $t[i, j] = t[i-1, j] \vee t[i-1, j-a_i] \vee (j = a_i)$. The final answer is found in $t[n, W]$.

When analyzing the runtime of the algorithm outlined in the previous paragraph, we notice that we fill nW cells, and computing each cell can be done in constant time. Hence, the runtime is $\mathcal{O}(nW)$. Note that this runtime is highly dependent on W, the target value. If W was encoded in unary, then the algorithm would indeed be polynomial — we would simply start by converting the input to binary, and then do the same as before.

The target value W above is a *structural parameter*. By restricting the value of W, we can give better guarantees on the runtime of the algorithm. For instance, if we restrict the instances we are willing to handle to those where W is bounded by some polynomial function of n, then we are also able to give a polynomial algorithm.

An NP-hard problem which has a pseudo-polynomial algorithm is called *weakly* NP-hard. If the problem does not have a pseudo-polynomial algorithm unless P = NP, then the problem is *strongly* NP-hard. The notion of weakly/strongly NP-completeness is defined analogously.

3.6.2 Parameterized complexity

Introduced in the 1990's by Rod Downey and Michael Fellows, parameterized complexity is a formalized framework for describing the computational complexity of problems with respect to *multiple* parameters that affect the runtime required to solve a problem — not only the input size, which is done in traditional complexity. Pseudo-polynomial algorithms are an example of parameterized algorithms which was well-known beforehand; but looking at how large numerical values exist in the input is by no means the only way to find structural parameters.

In the formal definitions by Downey and Fellows (2012), a parameter is simply some additional data (typically a number) which is provided with the problem instance — it does not need to appear in the "unparameterized" part of input instance in any way, though it can be used to describe some property it possesses if wanted; it does not need to appear in the structure of a witness for a *yes* -instance, though it can be used to describe this as well. Formally speaking, it is merely some extra data, a number.

These definitions are from the book by Cygan et al. (2015):

Parameterized decision problem A parameterized decision problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed, finite alphabet. For a problem instance $(s, k) \in \Sigma^* \times \mathbb{N}$, k is called the *parameter*.

FPT A parameterized decision problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called *fixed-parameter tractable* (FPT) if there exists an algorithm $\mathcal{A}$ (called a *fixed-parameter algorithm*), a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant c such that, given a problem instance $(s, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ correctly decides whether $(s, k) \in L$ in time bounded by $f(k) \cdot |(s, k)|^c$. The complexity class containing all fixed-parameter tractable problems is called FPT.

XP A parameterized decision problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called *slice-wise polynomial* (XP) if there exists an algorithm $\mathcal{A}$ and two computable functions $f, g : \mathbb{N} \to \mathbb{N}$ such that, given $(s, k) \in \Sigma^* \times \mathbb{N}$, the Algorithm $\mathcal{A}$ correctly decides whether $(s, k) \in L$ in time bounded by $f(k) \cdot |(s, k)|^{g(k)}$. The complexity class containing all slice-wise polynomial problems is called XP.

We can assume that the functions f and g in the definitions of FPT and XP are non-decreasing functions.

Note that the class FPT clearly is contained in the class XP, since we are free to let the function g always output a constant if we so desire. The converse is not true.

3.6.3 Parameterized reductions

From an algorithm designer's point of view, the classes FPT and XP are important complexity classes in parameterized complexity, since algorithms in these classes can be implemented on normal computers. However, whenever we have a parameterized problem for which we are unable to find an FPT-algorithm, we desire to show that it is *hard* or even *complete* for some class that is not contained in FPT — this will mimic how hardness is showed in the unparameterized setting. Even if it is unknown if the class is contained in FPT, then such a

conditional hardness results will still have value; a later proof that the problem actually is FPT will cause a collapse in the hierarchy, showing that *all* problems in said class actually have an FPT-algorithm.

In order to show such hardness, we need a notion of *reductions* which is appropriate with respect to parameterized problems and the class FPT. In comes the notion of a *parameterized* reduction:

FPT many-one reduction Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized decision problems. A FPT many-one reduction (also called a parameterized reduction) from A to B is a fixed-parameter algorithm that, given an instance (s, k) for A, outputs an instance (s', k') for B such that

- (s, k) is a *yes*-instance of A if and only if (s', k') is a *yes*-instance of B, and
- $k' \leq g(k)$ for some computable function g.

We emphasize that the runtime of an FPT-reduction must be bounded by $f(k) \cdot |(s, k)|^c$ for some computable function f and some constant c, since the reduction is an FPT-algorithm.

Note that if there is such a parameterized reduction from A to B, and B is in FPT, then also A is in FPT.

Our parameterized reductions differ from polynomial-time many-one reductions in two regards: first, we allow the runtime of the algorithm to be FPT; and second, the parameter must be preserved. It turns out that many polynomial-time many-one reductions are also parameterized reductions for some natural parameters. It is, however, not always the case.

Using such reductions, we can show some problems are "at least as difficult" as others; the problem being reduced *to* is at least as hard as the problem being reduced from. This shows a conditional hardness: assuming A is hard (i. e. not in FPT), and that A is reducible to B, then B must also be hard. By chaining reductions, we can obtain an intricate network of problems which can be reduced to each other, and sometimes one goes full circle – there is a sequence of parameterized reductions leading back to the original problem. Then all the problems in that cycle are "equally" hard with respect to obtaining FPT algorithms; either none of them, or all of them, are in FPT.

3.6.4 The W-hierarchy

While many problems are proven to be equally hard with respect to FPT-reductions, it turns out that the digraph showing which parameterized problems we know to be FPT-reducible to each other has *multiple* strongly connected components, even if restricting the universe to those problems which are in XP and unknown to be in FPT. This suggests that there is a hierarchy of complexity classes between FPT and XP.

Downey and Fellows (1992) introduced the W hierarchy, which describes a family of parameterized complexity classes $\mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \mathsf{W}[3] \subseteq \ldots$ which elegantly capture some prominent strongly connected components in the FPT-reducible digraph as the *complete* problems for a certain class in the hierarchy. It holds that FPT is contained in $\mathsf{W}[1]$, and for every

positive integer t, $\mathsf{W}[t]$ is contained in XP. It is not known if any of these classes are distinct, or even if they are different from FPT; however, it is conjectured that the inclusions are all proper.

We will not formally define classes in the W hierarchy here, but refer to Cygan et al. (2015) and Downey and Fellows (2012) for a deeper introduction. However we would like to point out some parameterized problems which are complete with respect to these classes.

$\mathsf{W}[1]$ Examples of $\mathsf{W}[1]$-complete problems are the k-Clique and k-Independent Set problems, including their colored versions. A parameterized reduction from any of these problems will show that the problem reduced to is $\mathsf{W}[1]$-hard.

$\mathsf{W}[2]$ Examples of $\mathsf{W}[2]$-complete problems are the k-Dominating Set, k-Set Cover and k-Hitting Set problems. A parameterized reduction from any of these problems will show that the problem reduced to is $\mathsf{W}[2]$-hard.

$\mathsf{W}[t]$ An example of a $\mathsf{W}[t]$-complete problem (for $t \geq 2$) is that of Weighted t-normalized Satisfiability, where the input is a Satisfiability formula φ which is t-normalized, and the parameter k is an upper bound on the number of variables that can be set to True. A formula is t-normalized if it is the big conjunction (or big disjunction) of subformulae, each of which is a ℓ-normalized formula for some $\ell \leq t - 1$. A 0-normalized formula is a literal, that is, a variable or its negation.

Chapter 4

Graph classes

A *graph class* is a predicate on the class of all graphs, categorizing every graph into one of two categories: those possessing the property, and those who do not.

Consider for instance the property of *connectivity*. We say that a graph G is *connected* if for every non-trivial partition of the vertices into two sets $A, B \subseteq V(G)$, it holds that there exists an edge $uv \in E(G)$ such that $u \in A$ and $v \in B$. This is a *property* a given graph can have or not have — but it also defines a graph class, namely the collection of all graphs which are connected.

Graph classes can be described in various different ways; some graph classes are described by exactly which graphs are in the graph class. Other graph classes are described recursively by how they can be built. Yet other classes are expressed as all graphs which can or can not be modified in a certain way to be something else. Often, a graph class can be described in many ways.

We begin by describing some very narrow graph classes, where each graph in the class even has a special name.

Complete graph A graph G is *complete* if it contains every possible edge, i. e. if $E(G) = \binom{V(G)}{2}$. The complete graph on n vertices is denoted K_n.

Edgeless graph A graph G is an *edgeless* graph if it does not contain any edges, i. e. if $E(G) = \emptyset$. The edgeless graph on n vertices is denoted $\overline{K_n}$.

Path A graph P is a *path* if there exists a permutation σ over $V(G) = \{u_1, u_2, \ldots, u_n\}$, such that the neighborhood of each vertex is exactly its neighboring vertices in the permutation. In other words, the set of edges for P is exactly $E(G) = \{\sigma(u_i)\sigma(u_{i+1}) \mid i \in [n-1]\}$. The *endpoints* of P are $\sigma(u_1)$ and $\sigma(u_n)$.

The path on n vertices, for $n \in \mathbb{N}$, is denoted P_n. The path on n vertices is said to have *length* $n - 1$.

In a graph G, the distance between two vertices $u, v \in V(G)$ is equal to the length of the shortest path with u and v as its endpoints.

Cycle A graph C is a *cycle* if removing *any* edge from the graph yields a path. It follows that a graph C is a cycle if and only if every vertex has degree 2 and C is connected. The cycle on n vertices, for $n \in \mathbb{N}_{\geq 3}$, is denoted C_n.

A cycle on n vertices also have n edges, and we say that the *length* of the cycle is n. If n is odd, we say that C_n is an *odd* cycle.

Complete bipartite A graph G is a *complete bipartite* graph if its vertex set can be partitioned into two sets $A, B \subseteq V(G)$ such that all edges between A and B are present, and no other edges are. In other words, $E(G) = \{uv \mid u \in A \wedge v \in B\}$. The complete bipartite graph with $a = |A|$ vertices in one partition and $b = |B|$ vertices in the other, is denoted by $K_{a,b}$.

Star A graph G is a *star* if is a complete bipartite graph where one side of the bipartition has size 1. Equivalently, there exists a vertex $r \in V(S)$ called the *root*, such that r is universal in S and $S - r$ is edgeless. The vertices in $V(S) \setminus \{r\}$ are called the *leaves* or *S*. The star with k leaves is denoted $K_{1,k}$.

4.1 Forbidden graph characterization

While some graph classes are described by exactly which graphs they contain, other graph classes are described by which graphs they do *not* contain — or more generally, which graphs they can not be transformed into by a certain toolbox of graph modifications.

For example, a graph is said to be *triangle-free* if it does not contain a C_3 as an induced subgraph. In other words, if a graph is such that it is possible to remove some vertices such that the remaining graph is a triangle, then that graph is *not* triangle-free.

This and similar ways of characterizing graph classes has proven incredibly useful, since they can describe a wide variety of graph classes, and it is also possible to prove powerful meta-theorems about such graph classes.

A graph G can be related to a graph H in multiple ways. Most commonly:

Induced subgraph A graph G contains H as an *induced subgraph* if it is possible to obtain H from G by:

- removing vertices.

If a graph G does not contain an induced subgraph isomorphic to H, then G is *H-free*.

Subgraph A graph G contains H as a *subgraph* if it is possible to obtain H from G by:

- removing vertices, and
- removing edges.

If a graph G does not contain a subgraph isomorphic to H, then G is *H-subgraph free*.

Minor A graph G contains H as a *minor* if it is possible to obtain H from G by:

- removing vertices, and
- removing edges, and
- contracting edges.

$\{C_3\}$	IS	Triangle-free
$\{C_3\}$	M	Forest
$\{C_3, C_4, C_5, \ldots\}$	S	Forest
$\{C_3, C_5, C_7, \ldots\}$	S	Bipartite
$\{C_4, C_5, C_6, \ldots\}$	IS	Chordal
$\{K_5, K_{3,3}\}$	M	Planar
$\{K_4, K_{2,3}\}$	M	Outerplanar
$\{C_4, C_5, K_2 + K_2\}$	IS	Split
$\{P_4\}$	IS	Cograph
$\{K_{1,4}\}$	S	Subcubic

Table 4.1: Some well-known graph classes and their characterization in terms of forbidden structures. In the first column is the set of forbidden structures, in the second column the relation (S = subgraph, IS = induced subgraph, M = minor), in the third column is the name of the resulting graph class

If a graph G does not contain a minor isomorphic to H, then G is *H-minor free*.

Induced minor A graph G contains H as an *induced minor* if it is possible to obtain H from G by:

- removing vertices, and
- contracting edges.

If a graph G does not contain an induced minor isomorphic to H, then G is *H-induced minor free*.

We can define plenty of well-known graph classes in this manner; see Table 4.1.

4.2 Central graph classes

Trees

A particularly important graph class is that of *trees*. A graph T is a tree if it is a connected forest — in other words, for every two vertices there is a unique path which connects them, and there are no cycles.

A vertex of degree 1 in a tree is called a *leaf*. The set of all leaves of a tree T is denoted $L(T) = \{u \in V(T) \mid \deg_T(u) = 1\}$. A vertex with higher degree is called an *internal vertex*.

A tree can be *rooted*, in which case a vertex $r \in V(T)$ is designated as the *root*. For every vertex other than the root $u \in V(T) \setminus \{r\}$ there is a unique path with endpoints u and r; the neighboring vertex to u in this path is called the *parent* of u. Every neighbor of a vertex $u \in V(T)$ which is not u's parent, is a *child* of u. A vertex $u \in V(T)$ is an *ancestor* of a vertex $v \in V(T)$ if u is on the path from v to the root r.

The *descendants* of a vertex $u \in V(T)$ is the set of all vertices for which u is an ancestor. The *subtree* of u is the graph induced on the vertex u and all its descendants. It is denoted by T_u, and it is naturally rooted in u.

Chordal graphs

A graph G is *chordal* if every cycle longer than 3 has a *chord*, i. e. at least one edge connecting two non-consecutive vertices in the cycle. An alternative characterization is that chordal graphs are those which admits a *perfect elimination ordering*. This is an ordering of the vertices such that if they are removed from the graph in order, then the neighborhood the vertex being removed will always induce a complete graph (Fulkerson and Gross, 1965). Yet another characterization is that a graph is chordal if and only if every minimal separator is a clique (Dirac, 1961).

Co-comparability graphs

A graph G is a *comparability* graph if there exists a partial order on $V(G)$ such that u and v are comparable in the partial order if and only if there is an edge $uv \in E(G)$. G is a comparability graph if and only if its complement $\overline{G}$ is a *co-comparability* graph.

An alternative characterization of co-comparability graphs is that they are the graphs which admit a *co-comparability ordering*. This is an ordering of the vertices $u_1, u_2, \ldots, u_n$ such that for all integers i, j, k with $1 \leq i < j < k \leq n$, we have that u_iu_k implies u_iu_j or u_ju_k. In other words, for any edge u_iu_k crossing a vertex u_j in the ordering, u_j is adjacent to at least one of the endpoints. Hence, if vertices u_i and u_k are on different sides of a vertex u_j in the order, then u_j is either in or has a neighbor in every path from u_i to u_k.

4.3 Graphs in logic

A graph class is merely a predicate on the class of all graphs. It turns out that formal logic systems such as first- and second order logic is able to capture the world of graphs in a natural way, and we can express many properties of graphs using such formulae.

The way this is done, is by letting a logic structure M_G represent the graph G. There are two common ways to do this, using either $\mathbf{MSO}_1$ or $\mathbf{MSO}_2$. The logic we expect to use is hence monadic second-order logic, though there is nothing preventing us from describing a graph property in first-order logic, for instance assuming the same graph representation as in $\mathbf{MSO}_1$. Regardless of which representation is used, we say that a formula φ representing a graph property is satisfied if M_G models φ; for simplicity we denote this by

$$G \models \varphi$$

since the model M_G is implicitly given by G. If G models φ, then G is a member of the graph class φ describes. For a thorough introduction to formal logic and graphs, we refer to Courcelle and Engelfriet (2012); however, we give a brief overview here.

Representation of graphs in $\mathbf{MSO}_1$

A natural way to let a structure M_G represent a (directed) graph G, is to let the domain of discourse U be the set of vertices $V(G)$, and let the signature have a single binary relation **adj** which M_G gives the following interpretation:

- The relation **adj** is a binary relation indicating whether one vertex is an (out-) neighbor of another. More precisely, $\mathbf{adj}(u, v)$ is $\textsc{True}$ if and only if $v \in N_G(u)$.

Note that an *undirected* graph G can be seen as a directed graph where for every edge $uv \in E(G)$, we also have $vu \in E(G)$. In other words, G is undirected if

$$M_G \models \forall u \forall v(\mathbf{adj}(u, v) \rightarrow \mathbf{adj}(v, u))$$

Both quantifiers in this formula are first-order; we can thus describe the property "the graph is undirected" in first-order logic.

More generally, the formula

$$\textsc{sym}(P) = \forall u \forall v(P(u, v) \rightarrow P(v, u))$$

indicates whether the binary relation P is *symmetric*. For a graph G to be *simple*, then the adjacency relation must be both symmetric and *irreflexive* (so it has no self-loops). Irreflexiveness can also be described as a first-order formula:

$$\textsc{irref}(P) = \forall u \neg P(u, u)$$

It follows that a graph G is simple if M_G is a model for the formula:

$$\textsc{Simple}_1 = \textsc{sym}(\mathbf{adj}) \wedge \textsc{irref}(\mathbf{adj})$$

So far we have used the perspective where we begin with a graph G check it against a graph property described as a formula φ. However, it is also possible to have the perspective where we start from the formula φ, and want to find a graph G (or a class of graphs) which satisfies φ. Since the signature of φ contains only a single relation, the number of different models for φ is limited — for each positive integer n, there is a direct correspondence between directed graphs on n vertices and structures with a universe of size $|U| = n$. In this way, every unique model M_G for φ represents a unique graph G.

Examples of $\mathbf{MSO}_1$ properties

We can express a variety of different properties of graphs using the $\mathbf{MSO}_1$ signature. For example, in a (simple) graph G a set $I \subseteq V(G)$ is *independent* if there are no edges between vertices of I, and a set $K \subseteq V(G)$ is a *clique* if every edge between distinct vertices of K is present. We can define the formulae

$$\textsc{Ind}(X) = \forall u_{\in X} \forall v_{\in X} \neg \mathbf{adj}(u, v)$$

$$\textsc{Clq}(X) = \forall u_{\in X} \forall v_{\in X} (\neg \mathbf{adj}(u, v) \rightarrow u = v)$$

which respectively indicate whether a set of vertices of X is an independent set or a clique. We can use these sub-formulae to describe a graph class. For example, a simple graph is *split* if its vertices can be divided into two sets K and I such that K is a clique and I is an independent set. The class of (simple) split graphs may thus be described by the formula

$$\textsc{Split} = \textsc{Simple}_1 \wedge \exists K_{\subseteq V(G)} \exists I_{\subseteq V(G)} \langle \forall u_{\in V(G)} \neg (u \in K \leftrightarrow u \in I) \wedge \textsc{Ind}(I) \wedge \textsc{Clq}(K) \rangle$$

A *vertex cover* for a simple graph G is a set of vertices $X \subseteq V(G)$ such that $V(G) \setminus X$ is an independent set. Equivalently, X is a vertex cover if every edge has an endpoint in X. For a given graph G and non-negative integer k, one might be interested to know whether there exists a set X of size $\leq k$ such that X is a vertex cover. We construct the formula

$$\begin{aligned}\textsc{vc}_k = \ & \textsc{Simple} \wedge \exists x_1 \exists x_2 \ldots \exists x_k \forall u \forall v(\\ & \mathbf{adj}(u,v) \rightarrow u = x_1 \vee v = x_1 \vee u = x_2 \vee v = x_2 \vee \ldots \vee u = x_k \vee v = x_k \\ &)\end{aligned}$$

which describes the class of all graphs which has a vertex cover of size at most k.

Representing graphs in $\mathbf{MSO}_2$

A limitation of $\mathbf{MSO}_1$, is that we may not quantify over edges. For example, we might want to express the property that a graph has a *Hamiltonian cycle* — that is, it contains a set of edges such that every vertex is incident to exactly two edges, and such that the graph induced by the edges is connected. Then we probably want to describe the graph property as a formula $\exists F_{\subseteq E(G)} \textsc{Ham}(F)$, where $\textsc{Ham}(F)$ is a formula checking that the edge set indeed is a Hamiltonian cycle. Unfortunately, this is not possible in $\mathbf{MSO}_1$, since edges are not elements of the universe, but rather encoded in the relation **adj**.

A way to remedy this is to let the universe also contain edges, for example by letting M_G represent the *incidence graph* of G. It turns out that $\mathbf{MSO}_2$ lends itself well for this purpose: for a graph G, a structure M_G representing G will have as its universes the sets $U_1 = V(G)$ and $U_2 = E(G)$. Moreover, the interpretation of M_G will have the following relation:

- The binary relation **inc** where the first argument is of type vertex and the second argument is of type edge. For $u \in V(G)$ and $e \in E(G)$, $\mathbf{inc}(u, e)$ indicate whether the vertex u is incident to the edge e.

Observe that the type of graph which this structure naturally capture, is a undirected multigraph with hyper-edges; however we can provide formulae which, if satisfied, restricts the graph to simpler graphs. We can also introduce direction by adding binary relation **tail** which takes as its arguments a vertex u and an edge e, and is $\textsc{True}$ if u is the (destination) endpoint of e.

As before, we can give a formula ensuring that the graph is simple:

$$\begin{aligned}\textsc{Simple}_2 = \ & \forall e_{\in E(G)} \exists u, v_{\in V(G)} [u \neq v \wedge \mathbf{inc}(u,e) \wedge \mathbf{inc}(v,e)] \\ & \wedge \forall e_{\in E(G)} \neg \exists u, v, w_{\in V(G)} [(u \neq v) \wedge (v \neq w) \wedge (u \neq w) \\ & \qquad \wedge \mathbf{inc}(u,e) \wedge \mathbf{inc}(v,e) \wedge \mathbf{inc}(w,e)] \\ & \wedge \forall e, f_{\in E(G)} \exists u_{\in V(G)} [\mathbf{inc}(u,e) \wedge \neg \mathbf{inc}(u,f)]\end{aligned}$$

The first line of the formula ensures that each edge is incident to at least two vertices; the second line ensure it is incident to at most two vertices. On the last line it is ensured that all edges are distinct.

While adjacency is not in the signature of this representation, we can define the open formula

$$\textsc{Adj}(u,v) = (u \neq v) \wedge \exists e_{\in E(G)} \mathbf{inc}(u,e) \wedge \mathbf{inc}(v,e)$$

which provides the same function. We can also annotate the formula by the argument E, the set of edges to consider. We get

$$\textsc{Adj}_E(u, v) = u \neq v \wedge \exists e_{\in E} \mathbf{inc}(u, e) \wedge \mathbf{inc}(v, e)$$

Example of $\mathbf{MSO}_2$ property: Hamiltonian cycle

Let us return to our example of a Hamiltonian cycle. Let us begin by defining a formula $\textsc{Ham}(V, E)$ which determines whether a subgraph given by $V \subseteq V(G)$ and $E \subseteq E(G)$ contains a Hamiltonian cycle. Then substituting in $V(G)$ and $E(G)$ for V and E respectively will give our answer. We let

$$\textsc{Ham}(V, E) = \exists F_{\subseteq E} \textsc{IsCycle}(V, F)$$

where $\textsc{IsCycle}(V, E)$ is a formula indicating whether the subgraph given by the vertices V and the edges E represents a cycle. The formula indicating a cycle is

$$\textsc{IsCycle}(V, E) = \forall u_{\in V} \textsc{twoInc}(u, E) \wedge \textsc{Conn}(V, E)$$

where $\textsc{twoInc}(u, E)$ indicates whether a vertex u is incident to exactly two edges of E, and $\textsc{Conn}(V, E)$ indicates whether the subgraph given by vertices V and edges F is connected. We omit defining twoInc here, as it is analogous to how we ensured that each edge is incident to two vertices in the formula $\textsc{Simple}_2$ above. We proceed to give the formula for connectivity:

$$\begin{aligned}\textsc{Conn}(V, E) = \; & \forall X_{\subseteq V}[\langle \exists u, v_{\in V}(u \in X \wedge v \notin X)\rangle \rightarrow \\ & \langle \exists u, v_{\in V}(\textsc{Adj}_E(u, v) \wedge u \in X \wedge v \notin X)\rangle]\end{aligned}$$

The formula for connectivity is basically checking that every way of partitioning the vertex set into two non-empty parts will lead to some edge crossing between the sets — there will exist two adjacent vertices where one is in X and the other is not. Note that this formulae for a cycle and for connectivity is also expressible for graphs in $\mathbf{MSO}_1$. The only location where quantification over edges was actually used, was in the very beginning of the formula for Hamiltonicity. This concludes the formula for Hamiltonian path.

4.4 Graph problems

A *graph problem* is a computational task involving a question about graphs. We will only deal with problems which takes a single graph as input. Since we deal with decision problems, we can look at such graph problems from a few different perspectives:

- A graph decision problem is a yes/no question we can ask about an input graph G.
- A graph decision problem is a predicate P on the class of all graphs; in other words, we want to compute the truth value $P(G)$ for some input graph G. For example, if the predicate can be described in $\mathbf{MSO}_1$ or $\mathbf{MSO}_2$, then the graph problem can be understood as asking whether the corresponding model satisfies the formula.
- A graph decision problem takes as input a graph G, and ask for a certain graph class $\mathcal{G}$ whether $G \in \mathcal{G}$.

Some of our favorite problems to reference are:

Independent set, vertex cover and clique

For a graph G, a vertex set $I \subseteq V(G)$ is called *independent* if there is no edge in $E(G)$ with both endpoints in I. Given an integer k and a graph G, we can ask whether the graph G contains an independent set of size (at least) k.

Independent Set
Input: A graph G and an integer k.
Question: Does there exist an independent set $I \subseteq V(G)$ of size k?

We can think of this problem as computing a function $\text{IS}_k : \mathcal{G} \to \{\text{True}, \text{False}\}$ defined as

$$\text{IS}_k(G) = \begin{cases} \text{True} & \text{if } \exists I \subseteq V(G) \text{ s.t. } |I| = k \text{ and } \forall e \in E(G)[e \not\subseteq I] \\ \text{False} & \text{otherwise} \end{cases}$$

where $\mathcal{G}$ is the class of all simple graphs. We can also think of the problem as deciding whether an input graph G belongs to the graph class $\mathcal{IS}_k = \{G \in \mathcal{G} \mid \exists I \subseteq V(G) \text{ s.t. } |I| = k \text{ and } \forall e \in E(G)[e \not\subseteq I]\}$.

The dual problem to Independent Set is Vertex Cover. A vertex set $X \subseteq V(G)$ is a *vertex cover* if every edge has at least one endpoint in X. Note that if X is a vertex cover, then $V(G) \setminus X$ is an independent set and vice versa. Given an integer k and a graph G, we can ask whether the graph G contains a vertex cover of size (at most) k.

Vertex Cover
Input: A graph G and an integer k.
Question: Does there exist a vertex cover $X \subseteq V(G)$ of size k?

A *clique* in a graph G is a vertex set $K \subseteq V(G)$ such that every two distinct vertices of K are neighbors. Given an integer k and a graph G, we can ask whether the graph G contains a clique of size (at least) k.

Clique
Input: A graph G and an integer k.
Question: Does there exist a clique $K \subseteq V(G)$ of size k?

We remark that a graph G contains a clique of size k if and only if the complement graph $\overline{G}$ contains an independent set of size k.

Dominating set

If G is a graph and $D \subseteq V(G)$ is a vertex set, then D is called a *dominating set* if every vertex in G is either in D or has a neighbor in D. Given an integer k and a graph G, we can ask whether the graph G contains a dominating set of size (at most) k.

Dominating Set
Input: A graph G and an integer k.
Question: Does there exist a dominating set $D \subseteq V(G)$ of size k?

4.4.1 Graph modification problems

Given a certain set of graph modification operations and a graph class $\mathcal{G}$, is it possible to modify an input graph G with at most k operations such that the modified graph belongs to $\mathcal{G}$? For example, in the Vertex Cover problem our operation of choice is vertex removal, and we ask: can we remove k vertices from the input graph such that the graph is edgeless?

A multitude of problems can be described this way. Some examples include:

Edit operation	**Graph class**	k	**Problem name**
Vertex deletion	Edgeless	k	Vertex Cover
Vertex deletion	Complete	$n-k$	Clique
Vertex deletion	Forest	k	Feedback Vertex Set
Vertex deletion	Bipartite	k	Odd Cycle Transversal
Edge insertion	Chordal	k	Minimum Fill-in
Edge insertion, edge deletion	Union of cliques	k	Cluster Editing
Vertex deletion, edge deletion, edge contraction	$\{H\}$	∞	H-minor

4.4.2 Locally checkable vertex subset and vertex partitioning problems

Let $\sigma, \rho \subseteq \mathbb{Z}_{\geq 0}$ be finite or co-finite[1] subsets of non-negative integers. For a graph G and a vertex set $X \subseteq V(G)$, we say X is a (σ, ρ)-*dominating set* if

- for every vertex $u \in X$, σ contains the cardinality of its neighborhood in X, i. e. $|N(u) \cap X| \in \sigma$; and
- for every vertex $u \in V(G) \setminus X$, ρ contains the cardinality of its neighborhood in X, i. e. $|N(u) \cap X| \in \rho$.

For example, a $(\{0\}, \mathbb{Z}_{\geq 0})$-dominating set is an independent set; and a $(\mathbb{Z}_{\geq 0}, \mathbb{Z}_{\geq 1})$-dominating set is a (standard) dominating set. For each (σ, ρ), we can formulate the minimization, maximization and existence questions. Recall that the decision version of a minimization (or maximization) problem provides some target value k along with the input and ask whether the optimal solution is at most (or at least) the target value. Some examples of (σ, ρ)-problems are:

σ	ρ	?	**Problem name**
$\{0\}$	$\mathbb{Z}_{\geq 0}$	max	Independent Set
$\mathbb{Z}_{\geq 0}$	$\mathbb{Z}_{\geq 1}$	min	Dominating Set
$\{0\}$	$\mathbb{Z}_{\geq 1}$	min	Maximal Independent Set/Independent Dominating Set
$\{0\}$	$\{1\}$	$\exists$	Perfect Code
$\{1\}$	$\mathbb{Z}_{\geq 0}$	max	Induced Matching

[1] A co-finite subset of non-negative integers is a set such that there exists a largest integer which is not in the set.

The (σ, ρ) problems were introduced by Telle and Proskurowski (1997), and are called *locally checkable vertex subset* problems. They can be further generalized to *locally checkable vertex partitioning* (LCVP) problems, where one considers a *degree constraint matrix* D, which is a $q \times q$ matrix whose entries are finite or co-finite subsets of $\mathbb{Z}_{\geq 0}$. A D-partition of a graph G is a partition of the vertex set into parts $V_1, V_2, \ldots V_q$ such that for all $i, j \in [q]$ and all $u \in V_i$, we have $|N(u) \cap V_j| \in D[i, j]$. For example, if there exists a D_3-partitioning of a graph for the matrix

$$D_3 = \begin{bmatrix} \{0\} & \mathbb{Z}_{\geq 0} & \mathbb{Z}_{\geq 0} \\ \mathbb{Z}_{\geq 0} & \{0\} & \mathbb{Z}_{\geq 0} \\ \mathbb{Z}_{\geq 0} & \mathbb{Z}_{\geq 0} & \{0\} \end{bmatrix}$$

then the graph is 3-colorable. Furthermore, if we have a degree constraint matrix D, a graph G, and an integer k, we can ask whether there exists a D-partition of G such that the first partition V_1 has cardinality at least (or at most) k. Then the (σ, ρ) problems, including their maximization and minimization version, can be expressed by the degree constraint matrix:

$$D_{(\sigma,\rho)} = \begin{bmatrix} \sigma & \mathbb{Z}_{\geq 0} \\ \rho & \mathbb{Z}_{\geq 0} \end{bmatrix}$$

Each (σ, ρ) and LCVP problem has a d-*value*. This is defined as the maximum value contained in a finite set in the degree constraint matrix, or the maximum value not included in a co-finite set, whichever is highest.

Chapter 5

Graph decompositions and width parameters

A *graph decomposition* is a way of dividing different "sections" of a graph into separate containers, such that the combination of these containers in the appropriate way will form the full graph. The intuition behind decomposing graphs in this manner, is that saying something interesting about each part might be easier than saying something about the whole graph — and if we are able to say something about each part, perhaps we can combine this knowledge to say something about the whole afterwards.

The idea of breaking a graph into smaller parts, say something about the smaller parts and then combine the knowledge to say something about the whole, can be applied recursively. In this case, the decomposition will naturally obtain a tree-like structure.

For example, say that we want to design a divide-and-conquer-algorithm for one of our old favorites, the Independent Set problem based on the following idea:

1. Partition the vertices of G into two parts $A, B \subseteq V(G)$.
2. Let X be the set of vertices that has a neighbor in the opposite partition of itself. Then no vertex in $A \setminus X$ has a neighbor in B and vice versa.
3. For each independent set $I_X \in 2^X$:
   (a) Recursively find the best independent sets in A and B whose intersections with I_X is respectively $I_X \cap A$ and $I_X \cap B$.
   (b) Take the union of found independent sets and remember the largest one found.

If $f(n)$ is the runtime of this algorithm on a graph with n vertices, then $f(n+1)$ can be bounded by something like $\mathcal{O}(2^{|X|} \cdot f(n) + \mathsf{poly}(n))$. At the outset this does not seem very promising, with a horrendous total runtime of $2^{\mathcal{O}(n^2)}$ lurking at us. Imagine, though, that we are able to (by a stroke of magic) always partition the graph such that the set X has cardinality at most some constant w — then at least we bound the runtime to $2^{\mathcal{O}(wn)}$. Still, less impressive than the naïve $\mathcal{O}(2^n \cdot \mathsf{poly}(n))$ algorithm.

We can, however, employ the powerful concept of *memoization* in order to reduce the runtime significantly. By memoizing the answers, we only need to compute recursive calls with the same arguments once. In order for this to significantly speed up computation time,

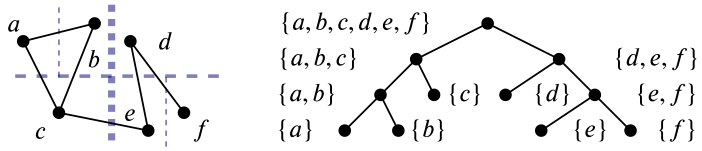


Figure 5.1: Example of a binary decomposition tree for a graph.

we need the number of distinct recursive calls to be relatively small. To obtain this, we require that the graph is partitioned in exactly the same manner in every recursive call. The way we consistently partition the graph forms a *binary decomposition tree*, where each node in the tree represents a subset of vertices; the two children of a node represents a non-trivial partition of that vertex set. See Figure 5.1.

By doing this, our binary decomposition tree will have $2n - 1$ nodes. Each such node can be recursively called with at most 2^w distinct arguments, and thus the number of distinct recursive calls is bounded by $2n \cdot 2^w$. The runtime local to each such recursive call is $\mathcal{O}(4^w \cdot \mathsf{poly}(n))$ time, leading us to a total runtime of at most $\mathcal{O}(8^w \cdot \mathsf{poly}(n))$. If w is constant, this runtime is *polynomial*! And if w is a parameter, then the algorithm is fixed parameter tractable.

Of course, finding good decompositions will not always be possible (unless $\mathsf{P} = \mathsf{NP}$), and the structure required to easily combine information about the parts to the whole can differ across different problems; but in those instances when we *can* do something like this, we can get very useful algorithms.

5.1 Branch decompositions and cut functions

In the context of a universal set X, the *complement* of a subset $A \subseteq X$ is the set $\overline{A} = X \setminus A$.

A *branch decomposition* of a set X is a pair (T, δ) where T is a subcubic tree called a *decomposition tree* and $\delta : X \to L(T)$ is a bijection from X to leaves of the tree. Each edge $e \in E(T)$ naturally defines a partition of X; let T_1 and T_2 be the two connected components in $T - e$, and define the part $A_e = \{u \in V(G) \mid \delta(u) \in V(T_1)\}$. It follows that $\overline{A}_e = \{u \in X \mid \delta(u) \in V(T_2)\}$, and $\{A_e, \overline{A}_e\}$ is a non-trivial partition of X. We call this partition the *cut* according to e. The set of all possible branch decompositions of X is denoted $\textsc{bd}(X)$.

A *cut function* on a set X is a symmetric[1] set function $f : 2^X \to \mathbb{R}$, i. e. it holds that $f(X') = f(X \setminus X')$ for every $X' \in 2^X$. For a branch decomposition (T, δ) on X, and an edge $e \in E(T)$, we let $f(e)$ be a shorthand notation for $f(A_e)$, where A_e is one part in the cut according to e. This is well-defined, since $f(A_e) = f(\overline{A}_e)$ as f is symmetric.

For a cut function f and a branch decomposition (T, δ) on a set X, the *f-width* of the branch decomposition is the maximum value of f across all edges of T, denoted $fw(T, \delta) = \max_{e \in E(T)} f(e)$.

[1] The requirement that the cut function is symmetric is not absolute; for example, the cut function for *module-width* is not symmetric. For non-symmetric cut functions, it is important that the branch decomposition is rooted in order for the width to be well-defined.

For a set X, the *f-width* of X is the minimum f-width of any branch decomposition of X, denoted $fw(X) = \min_{(T,\delta)\in \text{BD}(X)} fw(T,\delta)$.

In order to distinguish vertices of a decomposition tree from vertices of a general graph, we will call them *nodes*. This distinction is most useful when the set X originates from a graph.

5.2 Graph algorithms on branch decompositions: an example

In the context of graphs, we can consider branch decompositions and cut functions over either the set of vertices or the set of edges. In order to comply with our independent set story from the beginning of the chapter, we define as an illustrative example the cut function $cp : 2^{V(G)} \to \mathbb{Z}_{\geq 0}$. Let $cp(A)$ denote the cardinality of the smallest set X which contains all vertices in A with a neighbor in $V(G) \setminus A$ and also all vertices in $V(G) \setminus A$ with a neighbor in A. This set function is clearly symmetric. Let us name it the *cosmopolity* function, in honor of those who befriend others across borders; the function describes the number of "cosmopolitan" vertices.

By this definition of a cut function, we immediately obtain a corresponding notion of *cosmopolity-width* of a graph G, denoted as $cpw(G)$.

Given a branch decomposition (T,δ) for a graph G (that is, for the vertex set $V(G)$) with $cpw(T,\delta) = w$, we can solve Independent Set by dynamic programming on the decomposition tree; this corresponds to the recursive approach we broadly described earlier, but is done in a bottom-up fashion starting from the leaves.

In order to know which order this *is*, precisely, we want to make the decomposition tree *rooted*. It is also convenient to make each internal vertex have exactly two children. Thus we pick an internal vertex of degree 2 as the root; if there are none, we subdivide an edge and make the new vertex the root. We then smooth all other internal degree-2 vertices of T. Note that none of these operations will add or remove any partition, since the edges incident to a degree-2 vertex will both yield the same partition as the resulting edge after smoothing the vertex. The resulting decomposition tree is a *binary decomposition tree*.

In the binary decomposition tree, each node u represents a set of vertices in G, namely those δ maps to leaf nodes of T_u; let $V_u \subseteq V(G)$ denote these vertices. Each node (other than the root) also represents a *cut*, namely $\{V_u, \overline{V}_u\}$ corresponding to the edge to its parent node. Let X_u denote the set of vertices in V_u with a neighbor in $\overline{V}_u$; then $|X_u| \leq w$, where w is the cosmopolity-width of the decomposition. In our dynamic programming for Independent Set, we store in each node u:

- For each independent set I of $G[X_u]$, we store in the memory location $dp_u[I]$ the largest independent set of $G[V_u]$ that contains I and moreover does not contain any vertex of $X_u \setminus I$.

This is trivial to calculate for the leaves, since the graph is then only a singleton vertex. For an internal node u with children a and b, note that V_a and V_b forms a two-partition of V_u. See Figure 5.2. For an independent set $I \subset X_u$, we calculate $dp_u[I]$ as follows:

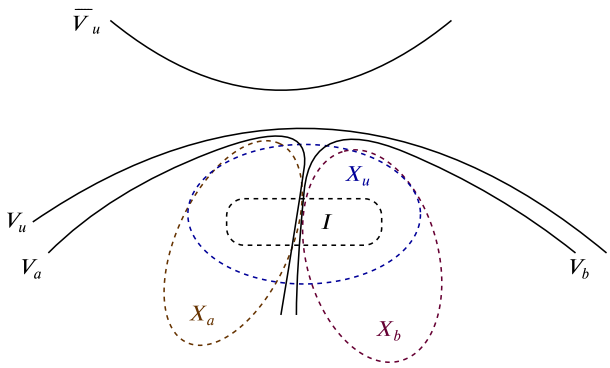


Figure 5.2: Venn diagram of relevant vertex sets when calculating the entry $dp_u[I]$ for an internal node u with children a and b.

- Let $dp_u[I]$ be equal to the largest union $dp_a[I_a] \cup dp_b[I_b]$ where I_a and I_b are subset of respectively X_a and X_b such that $I_a \cup I_b$ is independent and $(I_a \cup I_b) \cap X_u = I$.

We remark that calculating $dp_u[I]$ takes no more than $\mathcal{O}(4^w \cdot \mathsf{poly}(n))$ time. With less than $2n \cdot 2^w$ such entries to fill, the runtime of the dynamic programming algorithm described here requires $\mathcal{O}(8^w \cdot \mathsf{poly}(n))$ time. Note that this runtime can be improved to $\mathcal{O}(4^w \cdot \mathsf{poly}(n))$ by iterating over I_a and I_b first and then calculate I in polynomial time.

5.3 Width parameters for branch decompositions

Similarly to how we defined the cosmopolity cut function on the vertex set of graphs, there are multiple other cut functions which can be defined. We will describe a few such parameters here.

Carving width

The first width parameter based on branch decompositions of graphs was *carving-width* introduced by Seymour and Thomas (1994). Here, the cut function yields simply the number of edges crossing the cut. It is easy to see that the cosmopolity-width as defined in Section 5.2 is upper bounded by twice the carving-width — each edge is incident to only two vertices. Conversely, if the cosmopolity-width is w_{cp}, then the carving-width is upper bounded by $(\frac{w_{cp}}{2})^2$, the maximum number of edges in a bipartite graph with w_{cp} vertices. In other words, cosmopolity-width is bounded on the same graph classes as carving-width.

MM-width

Cosmopolity-width (and carving-width) is good for designing algorithms, and as such it has a strong *analytic power*. However, its *modelling power* is not very good. For instance, a graph as simple as a star will have a maximum possible cosmopolity-width, which is n;

any graph class which allow stars, or in fact any graph with a universal vertex, will have unbounded cosmopolity-width. Since we know how to solve independent set for stars, this suggest that there might be parameters with a stronger modeling power that also allow us to solve Independent Set in FPT time.

The crucial property of the set X in the context of our Independent Set algorithm parameterized by cosmopolity-width, is that it is a *separator* between the vertices $A \setminus X$ and $\overline{A} \setminus X$. However, it is overkill to require both $X \cap A$ and $X \cap \overline{A}$ to be separators; what we really want is the smallest set X that is a vertex cover for all edges crossing the cut — then that set is a separator.

If we refine the definition to be this, we end up with *maximum matching width*, as introduced by Vatshelle (2012). The cut function mm is defined over vertex subsets, $mm : 2^{V(G)} \to \mathbb{Z}_{\geq 0}$, and it gives the size of a maximum matching in the graph induced on the edges crossing the cut — equivalently, this is the size of a minimum set X which covers all edges crossing the cut.

This parameter is never further than a factor 3 from the well-known decomposition parameter *treewidth*, which we will briefly introduce later. Thus, it has the same modeling power and analytic power as treewidth: graph classes such as trees, cactuses, outerplanar graphs and series-parallel graphs all have mm-width bounded by a constant, and problems such as independent set, dominating set, feedback vertex set, Hamiltonian path and graph isomorphism can be solved in FPT time when mm-width is the parameter.

Rank-width

The *rank-width* of a graph is defined as the cut function which returns the $\mathrm{GF}[2]$-rank of the graph $G[A, \overline{A}]$ where A and $\overline{A}$ are the partitions of the cut, and $G[A, \overline{A}]$ is the graph induced on the edges crossing the cut (Oum and Seymour, 2006). This parameter is shown to be bounded on the same graph classes as *clique-width*; this class of graphs is strictly larger than the class of graphs with bounded mm-width; for example, cliques have constant rank-width, but unbounded mm-width.

MIM-width

The mim*-width* of a graph G, is a branch decomposition parameter defined as a cut function on vertex subsets $mim : 2^{V(G)} \to \mathbb{Z}_{\geq 0}$, where the cut value is defined as the size of a maximum induced matching in the graph induced on the edges crossing the cut. Introduced by Vatshelle (2012), this parameter has an even better modeling power than rank-width; graph classes such as interval graphs, permutation graphs, trapezoid graphs and many others have constant mim-width, even though their rank-width is unbounded.

On the analytic side, mim-width is as expected weaker than mm-width and the other width parameters higher up in the hierarchy. The most pressing issue is that no XP algorithm for determining mim-width is currently known, and a constant-factor polynomial-time approximation algorithm is unlikely to exist (unless NP $=$ ZPP, see Sæther and Vatshelle (2016)). However, for some graph classes it is possible to find good decompositions; the classes mentioned above are just a few such examples.

SIM-width

In Chapter 7, we introduce the width parameter *sim*-width. This cut function is defined as the maximum induced matching of edges across the cut, but where we have the additional constraint that no two edges in the matching can be connected by an edge on the same side of the cut. This parameter has even stronger modeling power than mim-width, and graph classes such as split, chordal and co-comparability graphs have constant sim-width, even though their mim-width is unbounded. Moreover, a decomposition with constant sim-width can be found in polynomial time for the mentioned classes.

The analytic power of sim-width is unfortunately not really strong; since Dominating Set is NP-complete on chordal graphs, we can for example not expect to have an XP-algorithm for this problem with sim-width as the parameter.

However, we show that certain classes of graphs with bounded sim-width also have bounded mim-width, thereby extending the realm of graphs for which we can find a bounded mim-width decompositions.

5.4 Treewidth

Treewidth is a width parameter for graphs that goes longer back than the parameters based on branch decompositions. It was discovered independently at multiple occasions, most famously by Robertson and Seymour (1984), and it is arguably the most popular and well-known width parameter.

The treewidth of a graph is an integer aiming to describe how "tree-like" a graph is; for example, say that you create a graph by first making a tree on n nodes, and then replace each node in the tree with a clique of size k; each vertex in such a clique is made neighbor with each vertex in the "neighboring" cliques (in other words, the vertices in two neighboring cliques forms a larger clique of size $2k$). The intuition is that if n is much larger than k, then the graph will be quite tree-like at a distance, even though it actually contains hordes of cycles. The treewidth of this strange graph is exactly $2k - 1$. And more importantly: every subgraph of such a graph will also have treewidth at most $2k - 1$.

Another way that a graph can be "tree-like," is if the graph is made by first having a normal tree on n vertices, and then adding k vertices with an arbitrary neighborhood — for example, adding k universal vertices. This graph is also quite similar to a tree, since such a huge part of the graph is only a tree. The treewidth of such a graph, and all its subgraphs, is at most $k + 1$.

We describe a third way to be "tree-like". Observe first that a tree is a graph which can be constructed as follows: start with a single vertex, and repeatedly add one vertex to the graph; the new vertex must pick exactly one neighbor from the older vertices to connect to. Now, imagine that instead of starting with a single vertex, we start with a complete graph of size $k + 1$; we then add one vertex at a time, and the new vertex must choose a clique of size k as its neighborhood. The graphs which are constructed like this are called k-trees, and have treewidth k. In fact, the class of subgraphs of k-trees (called partial k-trees) is exactly the class of graphs with treewidth at most k.

More commonly, treewidth is defined as follows. A *tree decomposition* for a graph G is a

pair (T, χ) where T is a tree and $\chi = \{X_t\}_{t \in V(T)}$ is a set of *bags*, where each bag $X_t \subseteq V(G)$ is a set of vertices related to a node $t \in V(T)$. Furthermore, the following requirements hold:

- For each vertex $u \in V(G)$, there exists a node $t \in T$ such that its bag X_t contains u.
- For each edge $uv \in E(G)$, there exists a node $t \in T$ such that its bag X_t contains both u and v.
- For each vertex $u \in V(G)$, the set $T_u = \{t \in V(T) \mid u \in X_t\}$ induce a connected subgraph in T. In other words, the nodes which an arbitrary vertex u is related to are connected.

The *width* of a tree decomposition is the cardinality of its largest bag minus one, i. e. $w(T, \chi) = \max_{X_t \in \chi} |X_t| - 1$.[2] The *treewidth* of a graph G is the smallest width of any tree decomposition for G.

5.5 Graph grammars and clique-width

The width measure that became known as clique-width was introduced by Courcelle et al. (1993). The context in which this parameter is born, is that of *grammars* for graphs; in this area, a class of graphs is that which can be generated by a certain set of rules for combining and transforming graphs.

Consider for example the class of *cographs*. This class is defined by three rules as follows:

- The graph K_1, i. e. an isolated vertex u, is a cograph.
- If G and H are cographs, then their disjoint union $G + H$ is a cograph.
- If G and H are cographs, then the *complete join* $G \otimes H$ is a cograph. The complete join of two graphs G and H is obtained by first taking their disjoint union, and then adding every edge between vertices of G and vertices of H.

Graphs which are generated by grammars like this, have a natural decomposition tree, where the leafs are the "basic" graphs defined to belong in the class (in the case of cographs, each leaf represents a K_1). Internal nodes will have a type, depending on which rule for combining or transformation was used (for cographs, each internal node is either of type disjoint union or of type complete join). They also have an *expression*. See Figure 5.3.

[2] 1 is subtracted in order for trees to have tree-width 1

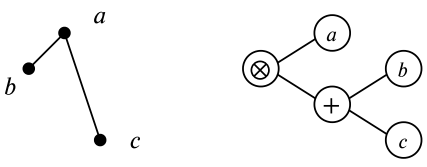


Figure 5.3: A cograph decomposition tree depicting the expression $a \otimes (b + c)$.

The class of graphs with clique-width at most k are generated similarly; they have a k-*expression* and a k-*decomposition tree*. The generation rules are slightly different, however, and during the creation process, each vertex is annotated with one of k *labels*; these labels are only significant in the construction and decomposition of the graph. We call a graph whose vertices are labeled by one of k labels, and which is constructed according to the rules below a k-*graph*. The rules are:

- An isolated vertex u with the label $i \in [k]$ is a k-graph. The k-expression for this graph is $i(u)$.
- If G and H are k-graphs, then their disjoint union $G + H$ is a k-graph.
- If G is a k-graph and $i, j \in [k]$ are two distinct labels, then the graph where all edges between vertices labeled i and vertices labeled j are added to G is also a k-graph. This graph is denoted $\eta_{i,j}(G)$.
- If G is a k-graph and $i, j \in [k]$ are two distinct labels, then the graph where all vertices with label i are relabeled to label j is a k-graph. This graph is denoted $\rho_{i,j}(G)$.

The class of cographs is exactly the class of graphs with clique-width at most 2. As such, the class of graphs with clique-width at most k is a generalization of cographs, similarly to how treewidth is a generalization of trees.

5.6 Algorithmic meta-theorems

5.6.1 Courcelle's theorem

Width parameters treewidth and clique-width have been shown to have a good analytical power; there are FPT parameters for many problems when these width measures are the parameters. A very powerful meta-theorem in this regard is *Courcelle's theorem*, which comes in two versions: one for treewidth (shown by Courcelle (1990); here in the words of Downey and Fellows (2013)) and one for clique-width (Courcelle et al., 2000):

- Given a graph G and a formula φ in (counting) $\mathbf{MSO}_2$ logic describing a graph property of interest, and parameterizing by the combination of $tw(G)$ (the treewidth of G) and the size of the formula φ, it can be determined in time $f(tw(G) + |\varphi|) \cdot n^{\mathcal{O}(1)}$ whether G has the property of interest.
- Given a graph G and a formula φ in (counting) $\mathbf{MSO}_1$ logic describing a graph property of interest, and parameterizing by the combination of $cw(G)$ (the clique-width of G) and the size of the formula φ, it can be determined in time $f(cw(G)+|\varphi|)\cdot n^{\mathcal{O}(1)}$ whether G has the property of interest.

While the dependency on the graph size in both cases is actually *linear* when the decomposition is given, the function f contains a tower of exponentials whose height is bounded by the width and φ (Frick and Grohe, 2004).

5.6.2 Locally checkable vertex subset and vertex partitioning problems

Telle and Proskurowski (1997) gave an FPT-algorithm for LCVP problems (recall Section 4.4.2) on graphs whose treewidth are bounded by k, with k, q and d as structural parameters; the runtime is of the form $\mathcal{O}(f(k,q,d)\cdot n)$. This runtime is linear in n, the number of vertices in an input graph. Their algorithm also works for minimizing or maximizing a certain partition. Since LCVP problems usually can be expressed in (counting) $\mathbf{MSO}_1$, the main contribution of this result is that the dependency on k is far better than what Courcelle's theorem can provide; f has complexity $(qd^q)^{\mathcal{O}(k)}$, which is singly exponential in k.

Continuing the work on LCVP problems, Bui-Xuan et al. (2013) gives an algorithm for graphs with bounded *Boolean-width*, whose runtime is $\mathcal{O}(n^4 \cdot 2^{\mathcal{O}(qdw^2)})$ where w is the structural parameter Boolean-width. We have not introduced Boolean-width here, but Vatshelle (2012) (see also Belmonte and Vatshelle (2013)) showed that it is upper bounded by $mimw \cdot \log_2 n$. It follows that LCVP problems can be solved in XP time when mim-width is the parameter.

In his thesis, Vatshelle (2012) strengthens the result by refining the runtime analysis for mim-width, finding that it is actually $n^{\mathcal{O}(w)}$ when w is the mim-width of a provided branch decomposition.

Chapter 6

Overview of Part II

In the upcoming part of the thesis, we present four papers which are approaching NP-hard graph problems with the perspective of exploiting structures in order to provide polynomial-time algorithms.

A width parameter useful for chordal and co-comparability graphs

In Chapter 7 we find the article (Kang et al., 2017). The background for this paper is that LCVP problems are known to have XP-algorithms when a branch decomposition of bounded mim-width is provided (Vatshelle, 2012). It follows that any algorithm which takes as input a graph from a graph class $\mathcal{G}$ and produce a branch decomposition for that graph with mim-width t in XP time, will also yield an XP-algorithm for any LCVP problem for that graph class, with t as the parameter.

This is exactly what we do in this paper. First, we consider $\mathcal{G}$ as the class of chordal graphs, and a structural parameter t defined as the smallest integer such that a graph is $K_t \boxminus \overline{K_t}$-free. The graph $K_t \boxminus \overline{K_t}$ is the disjoint union of a complete graph and an edgeless graph both on k vertices which are then joined by a perfect matching. For $K_t \boxminus \overline{K_t}$-free chordal graphs, we show how to provide a branch decomposition of mim-width at most $t-1$; this yields a $n^{\mathcal{O}(t)}$ algorithm for any fixed LCVP problem on the class of chordal graphs.

Similarly, we show that any LCVP problem can be solved in $n^{\mathcal{O}(t)}$ time on the class of co-comparability graphs, where the parameter t is defined as the smallest integer such that a graph is $K_t \boxminus K_t$-free (the graph $K_t \boxminus K_t$ is the disjoint union of two complete graphs of size k joined by a perfect matching).

To generalize these results further, we introduce a new width parameter called *special induced matching*-width, abbreviated to sim-width. We show that this parameter is constant for the class of chordal graphs and the class of co-comparability graphs, and for these classes we can also quickly find a branch decomposition with sim-width 1. Since the LCVP problem Dominating Set is NP-hard on chordal graphs, we can not hope to solve LCVP problems in polynomial time on the class of graphs with bounded sim-width. However, bounding it might allow us to use other parameters which would otherwise be insufficient — for instance the parameter t as described above (note that planar graphs are both $K_5 \boxminus K_5$-free and $K_5 \boxminus \overline{K_5}$-free, yet Dominating Set is still NP-hard; hence, being $\{K_t \boxminus K_t, K_t \boxminus \overline{K_t}\}$-free is not itself sufficient to yield XP-algorithms for LCVP problems with the parameter t).

We show that given a branch decomposition with sim-width w for a graph G which is $K_t \boxminus K_t$ and $K_t \boxminus \overline{K_t}$-free, then the same decomposition has mim-width $\mathcal{O}(wt^3)$.

Mim-width III. Graph powers and generalized distance domination problems

Chapter 8 contains the article (Jaffke et al., 2019). In this paper we investigate the generalization of LCVP problems to their *distance* versions. For example, in the distance-r version of Independent Set, the distance between two "independent" vertices must be at least r, and in the distance-r version of Dominating Set, a dominator can dominate all vertices at distance r or closer. We show that the distance-r versions of the LCVP problems are still in XP when the mim-width of a provided branch decomposition is the parameter. The main result is a purely structural one, namely that taking an arbitrary large *graph power* does not increase the mim-width by more than a factor 2. Since a distance-r LCVP problem is equivalent to the non-distance version on the r^{th} graph power, the claim follows.

We complement these findings by showing that many (σ, ρ) problems are W[1]-hard parameterized by mim-width + solution size, even in their non-distance versions.

In chapters 7 and 8, our common theme is that we show how a certain graph class is contained in a different graph class for which we have good algorithms; we show that the class of $\{K_t \boxminus K_t, K_t \boxminus \overline{K_t}\}$-free graphs with bounded t and constant sim-width is contained in the class of graphs with bounded mim-width, and we show that the class of graph powers of graphs with constant mim-width also has constant mim-width (albeit for different constants). In the final two chapters we obtain algorithmic consequences slightly differently, by providing algorithms directly.

Subgraph Complementation

Chapter 9 contains the paper (Fomin et al., 2020). Here, we investigate a tool for modifying graphs called a subgraph complement. A *subgraph complement* for a graph G entails picking a vertex set $X \subseteq V(G)$ and complementing all adjacencies within the set X. The graph $G \oplus X$ denotes the graph where a subgraph complement has been performed on the vertex set $X \subseteq V(G)$ in the graph G.

Given as input a graph G and for a constant graph class $\mathcal{G}$, we ask whether we can modify G with a subgraph complement to become a member of $\mathcal{G}$. Subgraph complementation is a generalization of adding or removing an edge, so clearly we can create any graph on the same number of vertices by applying this modification repeatedly; however, it is not obvious how many applications are needed. Is it sufficient with only a single application of the modification?

We show that this question can be answered in polynomial time for many choices of graph class $\mathcal{G}$, such as triangle-free, d-degenerate, cographs, and split graphs. For triangle-free and d-degenerate graph classes we provide novel algorithms.

We also show that the *clique-width* of a graph does not increase by more than a factor 3 when applying a subgraph complement, and that any $\mathbf{MSO}_1$ property can be extended to encompass those graphs which has a subgraph complement with the original property — this yields algorithmic consequences due to Courcelle's theorem for all classes $\mathcal{G}$ that has bounded clique-width and can be expressed in $\mathbf{MSO}_1$; for example cographs.

Similarly, we show that for every graph property P which can be described as a $k \times k$ M-partition, there exists a $2k \times 2k$ M-partition describing the property that a graph has a subgraph complement with property P. Due to existing algorithmic results for 4×4 M-partitions, this yields polynomial algorithms for any property which can be described as a 2×2 M-partition; for example split graphs.

Finally, we give a NP-hardness proof for the case when $\mathcal{G}$ is the class of regular graphs.

Time-inconsistent planning: simple motivation is hard to find

In Chapter 10 we provide an extended version of (Fomin and Strømme, 2020). The background for this paper is the *time-inconsistent planning model* due to Kleinberg and Oren (2018). In this model, an agent exhibiting present bias is moving through a weighted acyclic digraph, hoping to pick up a reward at the end of their path; they will move along the cheapest perceived path towards the target. Edge weights indicate how costly it is for the agent to move along the given edge. However, the agent continuously evaluate the path with a present bias, giving undue salience to the immediate next edge compared to later edges in the path; this cause the agent to behave irrationally, and change their path as they go. The model capture behaviors such as procrastination and abandonment.

If the agent at some point perceives the cheapest path to the target as costlier than the reward, they give up. In order to motivate the agent to finish, Kleinberg and Oren demonstrate that it in some cases is possible to remove edges and vertices from the graph such that the agent will not go to the point where they give up in the first place. This operation can be interpreted as a form of choice reduction, which psychological research also confirms to be beneficial with respect to completing tasks and making good choices.

This begs the question, which edges and vertices should we remove in order that a biased agent will reach the target? This is the problem of finding a *motivating subgraph*. Whilst shown to be NP-hard in the general case (Tang et al., 2017; Albers and Kraft, 2019), we consider the problem from a structural perspective: are there any structural properties of the *target* subgraph which we can exploit to find good algorithms?

As a first step, we show that deciding whether there exists a *path* which is motivating for the agent can be done in *linear time*; we give a dynamic programming algorithm over the topological order of the input graph for this. The requirement that the motivating subgraph is a path is, however, a very strong structural requirement; it is the very simplest scenario we can possibly imagine. We therefore try to allow the subgraph we look for to be slightly more complicated: we allow it to contain a single vertex with out-degree 2, and a single vertex with in-degree 2. Unfortunately, we show that the problem is NP-hard already in this case.

However, we do not give up; since the NP-hardness reduction was from Subset sum, which is only weakly NP-complete, there is still the hope that there exists a *pseudo-polynomial* algorithm when we allow a single vertex with out-degree 2. We show that this is indeed the case: if we allow the motivating subgraph to contain a constant k vertices whose out-degree is two or more, we scale up all weights and the reward to become integers, and we let W denote the reward, then we give an algorithm which solves the motivating subgraph problem in $(nW)^{\mathcal{O}(k)}$ time. Hence, the pseudo-polynomial algorithm is XP parameterized by the number of vertices with out-degree 2 or more in the solution.

Bibliography

Albers, S. and D. Kraft (2019). Motivating time-inconsistent agents: A computational approach. *Theory of Computing Systems 63*(3), 466–487. 55

Belmonte, R. and M. Vatshelle (2013). Graph classes with structured neighborhoods and algorithmic applications. *Theoretical Computer Science 511*, 54 – 65. Exact and Parameterized Computation. 51

Bui-Xuan, B., J. A. Telle, and M. Vatshelle (2013). Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoretical Computer Science 511*, 66–76. 51

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, New York, NY, USA, pp. 151–158. ACM. 6, 26, 28

Cook, S. A. and R. A. Reckhow (1973). Time bounded random access machines. *Journal of Computer and System Sciences 7*(4), 354 – 375. 7

Copeland, B. J. (2019). The church-turing thesis. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Spring 2019 ed.). Metaphysics Research Lab, Stanford University. 23

Courcelle, B. (1990). The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and computation 85*(1), 12–75. 50

Courcelle, B. and J. Engelfriet (2012). *Graph structure and monadic second-order logic: a language-theoretic approach*, Volume 138. Cambridge University Press. 36

Courcelle, B., J. Engelfriet, and G. Rozenberg (1993). Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences 46*(2), 218 – 270. 49

Courcelle, B., J. A. Makowsky, and U. Rotics (2000). Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems 33*(2), 125–150. 50

Cygan, M., F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh (2015). *Parameterized algorithms*, Volume 4. Springer. 30, 32

De Mol, L. (2019). Turing machines. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Winter 2019 ed.). Metaphysics Research Lab, Stanford University. 24

Diestel, R. (2017). *Graph theory* (5rd ed.), Volume 173 of *Graduate Texts in Mathematics*. Springer. 19

Dirac, G. A. (1961). On rigid circuit graphs. In *Treatises from the Mathematical Seminar at the University of Hamburg*, Volume 25, pp. 71–76. Springer. 36

Downey, R. G. and M. R. Fellows (1992). Fixed-parameter tractability and completeness. *Congressus Numerantium*, 161–161. 31

Downey, R. G. and M. R. Fellows (2012). *Parameterized complexity*. Springer Science & Business Media. 30, 32

Downey, R. G. and M. R. Fellows (2013). *Fundamentals of parameterized complexity*, Volume 4. Springer. 50

Fomin, F. V., P. A. Golovach, T. J. F. Strømme, and D. M. Thilikos (2020). Subgraph complementation. *Algorithmica*. 54

Fomin, F. V. and T. J. F. Strømme (2020). Time-inconsistent planning: simple motiavtion is hard to find. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*. 55

Fredman, M. L. (1975). On computing the length of longest increasing subsequences. *Discrete Mathematics 11*(1), 29 – 35. 13

Frick, M. and M. Grohe (2004). The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic 130*(1), 3 – 31. Papers presented at the 2002 IEEE Symposium on Logic in Computer Science (LICS). 50

Fulkerson, D. and O. Gross (1965). Incidence matrices and interval graphs. *Pacific journal of mathematics 15*(3), 835–855. 36

Jaffke, L., O. Kwon, T. J. F. Strømme, and J. A. Telle (2019). Mim-width iii. graph powers and generalized distance domination problems. *Theoretical Computer Science 796*, 216 – 236. 54

Kang, D. Y., O. Kwon, T. J. F. Strømme, and J. A. Telle (2017). A width parameter useful for chordal and co-comparability graphs. *Theoretical Computer Science 704*, 1–17. 53

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, pp. 85–103. Springer. 28

Kleinberg, J. and S. Oren (2018, February). Time-inconsistent planning: A computational problem in behavioral economics. *Communications of the ACM 61*(3), 99–107. 55

Levin, L. A. (1973). Universal sequential search problems. *Problemy peredachi informatsii 9*(3), 115–116. 6, 26

Neumann, J. v. (1945). First draft of a report on the edvac. Technical report. 7

Oum, S. and P. Seymour (2006). Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B 96*(4), 514–528. 47

Papadimitriou, C. H. (1997). Np-completeness: A retrospective. In *International Colloquium on Automata, Languages, and Programming*, pp. 2–6. Springer. 26

Robertson, N. and P. Seymour (1984). Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B 36*(1), 49 – 64. 48

Sæther, S. H. and M. Vatshelle (2016). Hardness of computing width parameters based on branch decompositions over the vertex set. *Theoretical Computer Science 615*, 120–125. 47

Sedgewick, R. and K. Wayne (2011). *Algorithms*. Addison-wesley professional. 9

Seymour, P. D. and R. Thomas (1994). Call routing and the ratcatcher. *Combinatorica 14*(2), 217–241. 46

Sipser, M. (2012). *Introduction to the Theory of Computation*. Cengage Learning. 23, 24

Stoll, R. R. (1979). *Set theory and logic*. Courier Corporation. 17

Tang, P., Y. Teng, Z. Wang, S. Xiao, and Y. Xu (2017). Computational issues in time-inconsistent planning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*. 55

Telle, J. A. and A. Proskurowski (1997). Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal on Discrete Mathematics 10*(4), 529–550. 42, 51

Turing, A. M. (1937). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society 2*(1), 230–265. 24

Turing, A. M. (1950, 10). I.—Computing Machinery and Intelligence. *Mind LIX*(236), 433–460. 4

Vatshelle, M. (2012). New width parameters of graphs. 47, 51, 53

Walicki, M. (2016). *Introduction to Mathematical Logic: Extended Edition*. World Scientific Publishing Company. 18

Part II

Publications

Chapter 7

A width parameter useful for chordal and co-comparability graphs

Dong Yeap Kang, O-joung Kwon, Torstein J. F. Strømme and Jan Arne Telle. In *Theoretical Computer Science*, Volume 704, 2017, Pages 1-17.

A preliminary version appeared in WALCOM 2017, LNCS 10167, Pages 93–105, 2017, and is available online with DOI 10.1007/978-3-319-53925-6_8.

A width parameter useful for chordal and co-comparability graphs

Dong Yeap Kang*[1], O-joung Kwon†[2], Torstein J. F. Strømme[3], and Jan Arne Telle[3]

[1]Department of Mathematical Sciences, KAIST, Daejeon, South Korea.
[2]Logic and Semantics, Technische Universität Berlin, Berlin, Germany.
[3]Department of Informatics, University of Bergen, Norway.

Abstract

Belmonte and Vatshelle (TCS 2013) used mim-width, a graph width parameter bounded on interval graphs and permutation graphs, to explain existing algorithms for many domination-type problems on those graph classes. We investigate new graph classes of bounded mim-width, strictly extending interval graphs and permutation graphs. The graphs $K_t \boxminus K_t$ and $K_t \boxminus S_t$ are graphs obtained from the disjoint union of two cliques of size t, and one clique of size t and one independent set of size t respectively, by adding a perfect matching. We prove that :

- interval graphs are $(K_3 \boxminus S_3)$-free chordal graphs; and $(K_t \boxminus S_t)$-free chordal graphs have mim-width at most $t-1$,
- permutation graphs are $(K_3 \boxminus K_3)$-free co-comparability graphs; and $(K_t \boxminus K_t)$-free co-comparability graphs have mim-width at most $t-1$,
- chordal graphs and co-comparability graphs have unbounded mim-width in general.

We obtain several algorithmic consequences; for instance, while MINIMUM DOMINATING SET is NP-complete on chordal graphs, it can be solved in time $n^{\mathcal{O}(t)}$ on $(K_t \boxminus S_t)$-free chordal graphs. The third statement strengthens a result of Belmonte and Vatshelle stating that either those classes do not have constant mim-width or a decomposition with constant mim-width cannot be computed in polynomial time unless $P = NP$.

We generalize these ideas to bigger graph classes. We introduce a new width parameter *sim-width*, of stronger modelling power than mim-width, by making a small change in the

*Supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2017R1A2B4005020) and also by TJ Park Science Fellowship of POSCO TJ Park Foundation.

†Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant DISTRUCT, agreement No. 648527).

[0]E-mail addresses: dynamical@kaist.ac.kr (D. Kang), ojoungkwon@gmail.com (O. Kwon), torstein.stromme@ii.uib.no (T. Strømme), Jan.Arne.Telle@uib.no (J. A. Telle).

An extended abstract appeared in the proceedings of 11th International Workshop on Algorithms and Computation, 2017.

definition of mim-width. We prove that chordal graphs and co-comparability graphs have sim-width at most 1. We investigate a way to bound mim-width for graphs of bounded sim-width by excluding $K_t \boxminus K_t$ and $K_t \boxminus S_t$ as induced minors or induced subgraphs, and give algorithmic consequences. Lastly, we show that circle graphs have unbounded sim-width, and thus also unbounded mim-width.

1 Introduction

The study of structural graph "width" parameters like tree-width and clique-width have been ongoing since at least the 1990's, and their algorithmic use has been steadily increasing [9]. A parallel and somewhat older field of study gives algorithms for special graph classes, such as chordal graphs and permutation graphs. Since the introduction of algorithms based on tree-width, which generalized algorithms for trees and series-parallel graphs, these two fields have been connected. Recently the fields became even more intertwined with the introduction of mim-width in 2012 [32], which yielded generalized algorithms for quite large graph classes. In the context of parameterized complexity a negative relationship holds between the modelling power of graph width parameters, i. e. what graph classes have bounded parameter values, and their analytical power, i. e. what problems become FPT or XP. For example, the family of graph classes of bounded width is strictly larger for clique-width than for tree-width, while under standard complexity assumptions the class of problems solvable in FPT time on a decomposition of bounded width is strictly larger for tree-width than for clique-width. For a parameter like mim-width its algorithmic use must therefore be carefully evaluated against its modelling power, which is much stronger than clique-width. A common framework for defining graph width parameters is by branch decompositions over the vertex set. This is a natural hierarchical clustering of a graph G, represented as an unrooted binary tree T with the vertices of G at its leaves. Any edge of the tree defines a cut of G given by the leaves of the two subtrees that result from removing the edge from T. Judiciously choosing a cut-function to measure the complexity of such cuts, or rather of the bipartite subgraphs of G given by the edges crossing the cuts, this framework then defines a graph width parameter by a minmax relation, minimum over all trees and maximum over all its cuts. Restricting T to a path with added leaves we get a linear variant. The first graph parameter defined this way was carving-width [29], whose cut-function is simply the number of edges crossing the cut, and whose modelling power is weaker than tree-width. The carving-width of a graph G is thus the minimum, over all such branch decomposition trees, of the maximum number of edges crossing a cut given by an edge of the tree. Several graph width parameters have been defined this way. For example the mm-width [32], whose cut function is the size of the maximum matching and has modelling power equal to tree-width; and the rank-width [25], whose cut function is the GF[2]-rank of the adjacency matrix and has modelling power equal to clique-width.

This framework was used by Vatshelle in 2012 [32] to define the parameter mim-width whose cut function is the size of the maximum induced matching of the graph crossing the cut. Note that low mim-width allows quite complex cuts. Carving-width one allows just a single edge, mm-width one a star graph, and rank-width one a complete bipartite graph with some isolated vertices. In contrast, mim-width one allows any cut where the neighborhoods of the vertices in a color class can be ordered linearly w.r.t. inclusion. The modelling power of mim-width is much stronger than clique-width. Belmonte and Vatshelle showed that interval graphs and permutation graphs have linear mim-width at most one, and circular-arc graphs and trapezoid

Figure 1: $K_5 \boxminus S_5$ and $K_5 \boxminus K_5$.

graphs have linear mim-width at most two [3] [1], whereas the clique-width of such graphs can be proportional to the square root of the number of vertices. With such strong modelling power it is clear that the analytical power of mim-width must suffer. The cuts in a decomposition of constant mim-width are too complex to allow FPT algorithms for interesting NP-hard problems. Instead, what we can get is XP algorithms, for the class of LC-VSVP problems [8] - locally checkable vertex subset and vertex partitioning problems - defined in Section 2.4. For classes of bounded mim-width this gives a common explanation for many classical results in the field of algorithms on special graph classes.

In this paper we extend these results on mim-width in several ways. Firstly, we show that chordal graphs and co-comparability graphs have unbounded mim-width, thereby answering a question by Belmonte and Vatshelle [3], and giving evidence to the intuition that mim-width is useful for large graph classes having a linear structure rather than those having a tree-like structure. Secondly, by excluding certain subgraphs, like $K_t \boxminus S_t$ obtained from the disjoint union of a clique of size t and an independent set of size t by adding a perfect matching, we find subclasses of chordal graphs and co-comparability graphs for which we can nevertheless compute a bounded mim-width decomposition in polynomial time and thereby solve LC-VSVP problems. See Figure 1 for illustrations of $K_t \boxminus S_t$. Note that already for $K_3 \boxminus S_3$-free chordal graphs the tree-like structure allowed by mim-width is necessary, as they have unbounded linear mim-width. Thirdly, we introduce sim-width, of modelling power even stronger than mim-width, and start a study of its properties.

The graph width parameter sim-width is defined within the same framework as mim-width with only a slight change to its cut function, simply requiring that a special induced matching across a cut cannot contain edges between any pair of vertices on the same side of the cut. This exploits the fact that a cut function for branch decompositions over the vertex set of a graph need not be a parameter of the bipartite graph on edges crossing the cut. The cuts allowed by sim-width are more complex than for mim-width, making sim-width applicable to well-known graph classes having a tree-like structure. We show that chordal graphs and co-comparability graphs have sim-width one and that these decompositions can be found in polynomial time. See Figure 2 for an inclusion diagram of some well-known graph classes illustrating these results. Since LC-VSVP problems like MINIMUM DOMINATING SET are NP-complete on chordal graphs we cannot expect XP algorithms parameterized by sim-width. However, for graphs of sim-width w, when excluding subgraphs $K_t \boxminus K_t$ or $K_t \boxminus S_t$, either as induced subgraphs or induced minors, we get graphs of bounded mim-width. The induced minor relation is natural since graphs of bounded sim-width are closed under induced minors, which might be of independent interest when taking the structural point of view. We show that by the alternate parametrization $w+t$ the LC-VSVP problems are solvable in XP time on graphs excluding $K_t \boxminus K_t$ or $K_t \boxminus S_t$, when given with a decomposition of sim-width w. The class of circle graphs is one of the classes listed

[1] In [3], all the related results are stated in terms of d-neighborhood equivalence, but in the proof, they actually gave a bound on mim-width or linear mim-width.

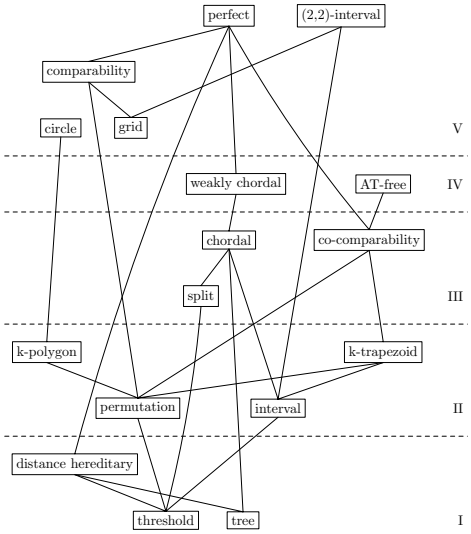


Figure 2: Inclusion diagram of some well-known graph classes. (I) Classes where clique-width and rank-width are constant. (II) Classes where mim-width is constant. (III) Classes where sim-width is constant. (IV) Classes where it is unknown if sim-width is constant. (V) Classes where sim-width is unbounded.

in [3] where either graphs in the class do not have constant mim-width, or it is NP-complete to find such a decomposition. Using a technique recently introduced by Mengel [24] we prove that sim-width of circle graphs is unbounded, which implies that mim-width of circle graphs is also unbounded.

Let us mention some more related work. Golovach et al. [18] applied linear mim-width in the context of enumeration algorithms, showing that all minimal dominating sets of graphs of bounded linear mim-width can be enumerated with polynomial delay using polynomial space. The graph parameter mim-width has also been used for knowledge compilation and model counting in the field of satisfiability [7, 27, 15]. The LC-VSVP problems include the class of domination-type problems known as (σ,ρ)-DOMINATION problems, whose intractability on chordal graphs is well known [6]. For two subsets of natural numbers σ,ρ a set S of vertices is called (σ,ρ)-dominating if for every vertex $v \in S$, $|S \cap N(v)| \in \sigma$, and for every $v \notin S$, $|S \cap N(v)| \in \rho$. Golovach et al. [17] showed that for fixed σ,ρ the problem of deciding if a given chordal graph has a (σ,ρ)-dominating set, is NP-complete whenever there exists at least one chordal graph containing more than one (σ,ρ)-dominating set, as this graph can then be used as a gadget in a reduction. Golovach, Kratochvíl, and Suchý [19] extended these results to the parameterized setting, showing that the existence of a (σ,ρ)-dominating set of size k, and at most k, are W[1]-complete problems when parameterized by k for any pair of finite sets σ and ρ. In contrast, combining our bounds on mim-width and algorithms of Bui-Xuan, Telle, and Vatshelle [8] we obtain the following.

Theorem 1.1. *Let $t \geq 2$ be an integer. Given an n-vertex $(K_t \boxminus S_t)$-free chordal graph or an n-vertex $(K_t \boxminus K_t)$-free co-comparability graph, every fixed LC-VSVP problem can be solved in time $n^{O(t)}$.*

As a specific example, we show that MINIMUM DOMINATING SET can be solved in time $O(n^{3t+4})$ and q-COLORING can be solved in time $O(qn^{3qt+4})$. We note that given an n-vertex graph, one can test in time $\mathcal{O}(n^{2t})$ whether it contains an induced subgraph isomorphic to $K_t \boxminus S_t$ or not. Therefore, a membership testing for $(K_t \boxminus S_t)$-free chordal graphs and the algorithm in Theorem 1.1 can be applied in time $n^{\mathcal{O}(t)}$. The same argument holds for $(K_t \boxminus K_t)$-free co-comparability graphs.

The remainder of this paper is organized as follows. Section 2 contains all the necessary notions required for our results. In Section 3, we prove that chordal graphs have sim-width at most 1 and mim-width at most $t-1$ if they are $(K_t \boxminus S_t)$-free. Similarly we show that co-comparability graphs have linear sim-width at most 1 and linear mim-width at most $t-1$ if they are $(K_t \boxminus K_t)$-free. We provide polynomial-time algorithms to find such decompositions, and discuss their algorithmic consequences for LC-VSVP problems. We also show that chordal graphs and co-comparability graphs have unbounded mim-width. In Section 4, we bound the mim-width of graphs of sim-width w that do not contain $K_t \boxminus K_t$ and $K_t \boxminus S_t$ as induced subgraphs or induced minors, and give algorithmic consequences. We also show that graphs of bounded sim-width are closed under induced minors. Lastly, we show in Section 5 that circle graphs have unbounded sim-width. We finish with some research questions related to sim-width in Section 6.

After our result appeared on arXiv June 2016, Mengel [24] obtained an asymptotically tight lower bound of mim-width on chordal graphs.

2 Preliminaries

In this paper, all graphs are finite and simple. We denote the vertex set and edge set of a graph G by $V(G)$ and $E(G)$, respectively. Let G be a graph. For a vertex v of G, we denote by $N_G(v)$ the set of neighbors of v in G. For $v \in V(G)$ and $X \subseteq V(G)$, we denote by $G-v$ the graph obtained from G by removing v, and denote by $G-X$ the graph obtained from G by removing all vertices in X. For $e \in E(G)$, we denote by $G-e$ the graph obtained from G by removing e, and denote by G/e the graph obtained from G by contracting e. For a vertex v of G with exactly two neighbors v_1 and v_2 that are non-adjacent, the operation of removing v and adding the edge v_1v_2 is called *smoothing* the vertex v. For $X \subseteq V(G)$, we denote by $G[X]$ the subgraph of G induced by X. A *clique* in G is a set of vertices of G that are pairwise adjacent, and an *independent set* in G is a set of vertices that are pairwise non-adjacent. A set of edges $\{v_1w_1, v_2w_2, \ldots, v_mw_m\}$ of G is called an *induced matching* in G if there are no other edges in $G[\{v_1, \ldots, v_m, w_1, \ldots, w_m\}]$.

For sets R and C, an (R,C)*-matrix* is a matrix whose rows and columns are indexed by R and C, respectively. For an (R,C)-matrix M, $X \subseteq R$, and $Y \subseteq C$, let $M[X,Y]$ be the submatrix of M whose rows and columns are indexed by X and Y, respectively. For a graph G, a $(V(G),V(G))$-matrix M is called the *adjacency matrix* of G if for $v,w \in V(G)$, $M[v,w] = 1$ if v and w are adjacent in G, and $M[v,w] = 0$ otherwise.

A pair of vertex subsets (A,B) of a graph G is called a *vertex bipartition* if $A \cap B = \emptyset$ and $A \cup B = V(G)$. For a vertex bipartition (A,B) of a graph G, we denote by $G[A,B]$ the bipartite graph on the bipartition (A,B) where for $a \in A$ and $b \in B$, a and b are adjacent in $G[A,B]$ if and only if they are adjacent in G. In other words, it is the graph obtained from G by removing edges whose both end vertices are contained in A or contained in B. For a vertex bipartition (A,B) of G and an induced matching $\{v_1w_1, v_2w_2, \ldots, v_mw_m\}$ in G where $v_1, \ldots, v_m \in A$ and $w_1, \ldots, w_m \in B$, we say that it is an induced matching in G between A and B.

We denote by $\mathbb{N}$ the set of all non-negative integers, and let $\mathbb{N}^+ := \mathbb{N} \setminus \{0\}$.

2.1 Graph classes

A tree is called *subcubic* if every internal node has exactly 3 neighbors. A tree T is called a *caterpillar* if it contains a path P such that every vertex in $V(T) \setminus V(P)$ has a neighbor in P. The complete graph on n vertices is denoted by K_n.

A graph is *chordal* if it contains no induced subgraph isomorphic to a cycle of length 4 or more. A graph is a *split* graph if it can be partitioned into two vertex sets C and I where C is a clique and I is an independent set. A graph is an *interval graph* if it is the intersection graph of a family of intervals on the real line. All split graphs and interval graphs are chordal. An ordering $v_1, \ldots, v_n$ of the vertex set of a graph G is called a *co-comparability ordering* if for every integers i, j, k with $1 \leq i < j < k \leq n$,

- if v_i is adjacent to v_k, then v_j is adjacent to v_i or v_k.

It is known that this condition is equivalent to the following: for every integers i, j, k with $1 \leq i < j < k \leq n$, v_j has a neighbor in every path from v_i to v_k avoiding v_j. A graph is a *co-comparability graph* if it admits a co-comparability ordering. Every co-comparability graph is the complement of some comparability graph, where comparability graphs are graphs that can be obtained from some partial order by connecting pairs of elements that are comparable to each other. A graph is a *permutation graph* if it is the intersection graph of line segments whose endpoints lie on two parallel lines. Permutation graphs are co-comparability graphs [11]. A graph is a *circle graph* if it is the intersection graph of chords in a circle.

For positive integer n, let $K_n \boxminus K_n$ be the graph on $\{v_1^1, \ldots, v_n^1, v_1^2, \ldots, v_n^2\}$ such that for all $i, j \in \{1, \ldots, n\}$,

- $\{v_1^1, \ldots, v_n^1\}$ and $\{v_1^2, \ldots, v_n^2\}$ are cliques,
- v_i^1 is adjacent to v_j^2 if and only if $i = j$,

and let $K_n \boxminus S_n$ be the graph on $\{v_1^1, \ldots, v_n^1, v_1^2, \ldots, v_n^2\}$ such that for all $i, j \in \{1, \ldots, n\}$,

- $\{v_1^1, \ldots, v_n^1\}$ is a clique, $\{v_1^2, \ldots, v_n^2\}$ is an independent set,
- v_i^1 is adjacent to v_j^2 if and only if $i = j$.

Since $K_2 \boxminus K_2$ is an induced cycle of length 4, chordal graphs do not contain $K_2 \boxminus K_2$ as an induced subgraph. We observe that $K_3 \boxminus S_3$ is not a co-comparability graph.

Lemma 2.1. *The graph $K_3 \boxminus S_3$ is not a co-comparability graph.*

Proof. Let G be a graph on $\{v_1, v_2, v_3\} \cup \{w_1, w_2, w_3\}$ such that

- $\{v_1, v_2, v_3\}$ is a clique, $\{w_1, w_2, w_3\}$ is an independent set, and
- v_i is adjacent to w_j if and only if $i = j$.

It is clear that G is isomorphic to $K_3 \boxminus S_3$. Suppose G admits a co-comparability ordering. By relabeling if necessary, we may assume w_1, w_2, w_3 appear in the co-comparability ordering in that order. However $w_1 v_1 v_3 w_3$ is a path from w_1 to w_3 that contains no neighbors of w_2, and thus, it contradicts the assumption that the given ordering is a co-comparability ordering. We conclude that $K_3 \boxminus S_3$ is not a co-comparability graph. □

2.2 Graph relations

Let G be a graph. A graph H is a *subgraph* of G if H can be obtained from G by removing some vertices and edges. A graph H is an *induced subgraph* of G if $H = G[X]$ for some $X \subseteq V(G)$. A graph H is an *induced minor* of G if H can be obtained from G by a sequence of removing vertices and contracting edges. A graph H is a *minor* of G if H can be obtained from G by a sequence of removing vertices, removing edges, and contracting edges. For a graph H, we say a graph is *H-free* if it contains no induced subgraph isomorphic to H.

A *minor model* of a graph H in G is a function η with the domain $V(H) \cup E(H)$, where

- for every $v \in V(H)$, $\eta(v)$ is a non-empty connected subgraph of G, all pairwise vertex-disjoint
- for every edge e of H, $\eta(e)$ is an edge of G, all distinct
- if $e \in E(H)$ and $v \in V(H)$ then $\eta(e) \notin E(\eta(v))$,
- for every edge $e = uv$ of H, $\eta(e)$ has one end in $V(\eta(u))$ and the other in $V(\eta(v))$.

It is well known that H is a minor of G if and only if there is a minor model of H in G. A minor model η of a graph H in G is an *induced minor model* if for every distinct vertices v_1 and v_2 in H that are non-adjacent, there are no edges between $V(\eta(v_1))$ and $V(\eta(v_2))$. A graph H is an induced minor of G if and only if there is an induced minor model of H in G.

2.3 Width parameters

For sets A and B, a function $f : 2^A \to B$ is *symmetric* if for every $Z \subseteq A$, $f(Z) = f(A \setminus Z)$.

For a graph G and a vertex set $A \subseteq V(G)$, we define functions cutrk_G, mim_G, and sim_G from $2^{V(G)}$ to $\mathbb{N}$ such that

- $\mathrm{cutrk}_G(A)$ is the rank of the matrix $M[A, V(G) \setminus A]$ where M is the adjacency matrix of G and the rank is computed over the binary field,
- $\mathrm{mim}_G(A)$ is the maximum size of an induced matching of $G[A, V(G) \setminus A]$,
- $\mathrm{sim}_G(A)$ is the maximum size of an induced matching between A and $V(G) \setminus A$ in G.

Remark that in $K_n \boxminus K_n$, $\mathrm{mim}_{K_n \boxminus K_n}(\{v_1^1, \ldots, v_n^1\}) = n$ and $\mathrm{sim}_{K_n \boxminus K_n}(\{v_1^1, \ldots, v_n^1\}) = 1$.

For a graph G, a pair (T, L) of a subcubic tree T and a bijection L from $V(G)$ to the set of leaves of T is called a *branch-decomposition*. For each edge e of T, let T_1^e and T_2^e be the two connected components of $T - e$, and let (A_1^e, A_2^e) be the vertex bipartition of G such that for each $i \in \{1, 2\}$, A_i^e is the set of all vertices in G mapped to leaves contained in T_i^e by L. We call (A_1^e, A_2^e) the vertex bipartition of G associated with e. For a branch-decomposition (T, L) of a graph G and an edge e in T and a symmetric function $f : 2^{V(G)} \to \mathbb{N}$, the *$f$-width* of e is define as $f(A_1^e)$ where (A_1^e, A_2^e) is the vertex bipartition associated with e. The *f-width* of (T, L) is the maximum f-width over all edges in T, and the *f-width* of G is the minimum f-width over all branch-decompositions of G. If $|V(G)| \leq 1$, then G does not admit a branch-decomposition, and the f-width of G is defined to be 0.

The *rank-width* of a graph G, denoted by $\mathrm{rw}(G)$, is the cutrk_G-width of G, and the *mim-width* of a graph G, denoted by $\mathrm{mimw}(G)$, is the mim_G-width of G, and the *sim-width* of a graph

G, denoted by $\mathrm{simw}(G)$, is the sim_G-width of G. For convenience, the f-width of a branch-decomposition is also called a rank-width, mim-width, or sim-width depending on the function f.

If T is a subcubic caterpillar tree, then a branch-decomposition (T,L) is called a *linear branch-decomposition*. The *linear f-width* of G is the minimum f-width over all linear branch-decompositions of G. The *linear mim-width* of a graph G, denoted by $\mathrm{lmimw}(G)$, is the linear mim_G-width of G, and the *linear sim-width* of a graph G, denoted by $\mathrm{lsimw}(G)$, is the linear sim_G-width of G.

By definitions we have the following.

Lemma 2.2. *For a graph G, we have* $\mathrm{simw}(G) \leq \mathrm{mimw}(G) \leq \mathrm{rw}(G)$.

We frequently use the following fact.

Lemma 2.3. *Let G be a graph, let w be a positive integer, and let $f : 2^{V(G)} \to \mathbb{N}$ be a symmetric function. If G has f-width at most w, then G admits a vertex bipartition (A_1, A_2) where $f(A_1) \leq w$ and for each $i \in \{1,2\}$, $\frac{|V(G)|}{3} < |A_i| \leq \frac{2|V(G)|}{3}$.*

Proof. Let (T,L) be a branch-decomposition of G of f-width at most w. We subdivide an edge of T, and regard the new vertex as a root node. For each node $t \in V(T)$, let $\mu(t)$ be the number of leaves of T that are descendants of t. Now, we choose a node t that is farthest from the root node such that $\mu(t) > \frac{|V(G)|}{3}$. By the choice of t, for each child t' of t, $\mu(t') \leq \frac{|V(G)|}{3}$. Therefore, $\frac{|V(G)|}{3} < \mu(t) \leq \frac{2|V(G)|}{3}$. Let e be the edge connecting the node t and its parent. By the construction, the vertex bipartition associated with e is a required bipartition. □

We also use tree-decompositions in Section 3. A *tree-decomposition* of a graph G is a pair $(T, \mathscr{B} = \{B_t\}_{t \in V(T)})$ such that

(1) $\bigcup_{t \in V(T)} B_t = V(G)$,

(2) for every edge in G, there exists B_t containing both end vertices,

(3) for $t_1, t_2, t_3 \in V(T)$, $B_{t_1} \cap B_{t_3} \subseteq B_{t_2}$ whenever t_2 is on the path from t_1 to t_3.

Each vertex subset B_t is called a *bag* of the tree-decomposition. The *width* of a tree-decomposition is $w-1$ where w is the maximum size of a bag in the tree-decomposition, and the *tree-width* of a graph is the minimum width over all tree-decompositions of the graph. It is well known that a graph has tree-width at most w if and only if it is a subgraph of a chordal graph with maximum clique size at most $w+1$; see for instance [5]. Furthermore, chordal graphs admit a tree-decomposition where each bag induces a maximal clique of the graph. We will use this fact in Section 3.

2.4 LC-VSVP problems

Telle and Proskurowski [31] classified a class of problems called *locally checkable vertex subset and vertex partitioning problems*, which is a subclass of MSO_1 problems. These problems generalize problems like MAXIMUM INDEPENDENT SET, MINIMUM DOMINATING SET, q-COLORING etc.

Let σ and ρ be finite or co-finite subsets of $\mathbb{N}$. For a graph G and $S \subseteq V(G)$, we call S a (σ,ρ)-*dominating set* of G if

- for every $v \in S$, $|N_G(v) \cap S| \in \sigma$, and
- for every $v \in V(G) \setminus S$, $|N_G(v) \cap S| \in \rho$.

For instance, a $(0, \mathbb{N})$-set is an independent set as there are no edges inside of the set, and we do not care about the adjacency between S and $V(G) \setminus S$. Another example is that an $(\mathbb{N}, \mathbb{N}^+)$-set is a dominating set as we require that for each vertex in $V(G) \setminus S$, it has at least one neighbor in S. See [31, Table 4.1] for more examples. The MIN-(OR MAX-)(σ, ρ)-DOMINATION problem is a problem to find a minimum (or maximum) (σ, ρ)-dominating set in an input graph G, and possibly on vertex-weighted graphs. These problems also called as *locally checkable vertex subset* problems.

For a positive integer q, a $(q \times q)$-matrix D_q is called a *degree constraint* matrix if each element is either a finite or co-finite subset of $\mathbb{N}$. A partition $\{V_1, V_2, \ldots, V_q\}$ of the vertex set of a graph G is called a *D_q-partition* if

- for every $i, j \in \{1, \ldots, q\}$ and $v \in V_i$, $|N_G(v) \cap V_j| \in D_q[i, j]$.

For instance, if we take a matrix D_q where all diagonal entries are 0, and all other entries are $\mathbb{N}$, then a D_q-partition is a partition into q independent sets, which corresponds to a q-coloring of the graph. The D_q-PARTITIONING problem is a problem deciding if an input graph admits a D_q-partition or not. These problems are also called as *locally checkable vertex partitioning* problems.

All these problems will be called *locally checkable vertex subset and vertex partitioning problems*, shortly LC-VSVP problems. As shown in [8] the runtime solving an LC-VSVP problem by dynamic programming relates to the finite or co-finite subsets of $\mathbb{N}$ used in its definition. The following function d is central.

1. Let $d(\mathbb{N}) = 0$.
2. For every finite or co-finite set $\mu \subseteq \mathbb{N}$, let $d(\mu) = 1 + \min(\max\{x \in \mathbb{N} : x \in \mu\}, \max\{x \in \mathbb{N} : x \notin \mu\})$.

For example, for MINIMUM DOMINATING SET and q-COLORING problems we plug in $d = 1$ because $\max(d(\mathbb{N}), d(\mathbb{N}^+)) = 1$ and $\max(d(0), d(\mathbb{N})) = 1$.

Belmonte and Vatshelle [3] proved the following application of mim-width.

Theorem 2.4 (Belmonte and Vatshelle [3] and Bui-Xuan, Telle, and Vatshelle [8])**.** *(1) Given an n-vertex graph and its branch-decomposition (T, L) of mim-width w and $\sigma, \rho \subseteq \mathbb{N}$, one can solve* MIN-(OR MAX-)(σ, ρ)-DOMINATION *in time $\mathcal{O}(n^{3dw+4})$ where $d = \max(d(\sigma), d(\rho))$.*

*(2) Given an n-vertex graph and its branch-decomposition (T, L) of mim-width w and a $(q \times q)$-matrix D_q, one can solve D_q-*PARTITIONING *in time $\mathcal{O}(qn^{3dwq+4})$ where $d = \max_{i,j} d(D_q[i, j])$.*

3 Mim-width of chordal graphs and co-comparability graphs

In this section, we show that chordal graphs admit a branch-decomposition (T, L) such that

1. (T, L) has sim-width at most 1, and

2. (T,L) has mim-width at most $t-1$ if the given graph is $(K_t \boxminus S_t)$-free for some $t \geq 2$.

We combine the second statement with Theorem 2.4, and show that LC-VSVP problems can be efficiently solved for $(K_t \boxminus S_t)$-free chordal graphs (Corollary 3.2). In the same context, we show that co-comparability graphs admit a linear branch-decomposition such that

1. (T,L) has linear sim-width at most 1, and
2. (T,L) has linear mim-width at most $t-1$ if the given graph is $(K_t \boxminus K_t)$-free for some $t \geq 2$.

We combine the second statement with Theorem 2.4, and show that LC-VSVP problems can be efficiently solved for $(K_t \boxminus K_t)$-free co-comparability graphs (Corollary 3.7). We also prove that chordal graphs and co-comparability graphs have unbounded mim-width in general.

3.1 Mim-width of chordal graphs

We prove the following.

Proposition 3.1. *Given a chordal graph G, one can in time $\mathcal{O}(|V(G)|+|E(G)|)$ output a branch-decomposition (T,L) with the following property:*

- *(T,L) has sim-width at most 1,*
- *(T,L) has mim-width at most $t-1$ if G is $(K_t \boxminus S_t)$-free for some integer $t \geq 2$.*

To show Proposition 3.1, we use the fact that chordal graphs admit a tree-decomposition whose bags are maximal cliques. Such a tree-decomposition can be easily transformed from a *clique-tree* of a chordal graph. A *clique-tree* of a chordal graph G is a pair $(T, \{C_t\}_{t \in V(T)})$, where

(1) T is a tree,

(2) each C_t is a maximal clique of G, and

(3) for each $v \in V(G)$, the vertex set $\{t \in V(T) : v \in C_t\}$ induces a subtree of T.

Gavril [16] showed that chordal graphs admit a clique-tree, and it is known that given a chordal graph G, its clique-tree can be constructed in time $\mathcal{O}(|V(G)|+|E(G)|)$ [4, 21].

Proof of Proposition 3.1. Let G be a chordal graph. We compute a tree-decomposition $(F, \mathscr{B} = \{B_t\}_{t \in V(F)})$ of G whose bags induce maximal cliques of G in time $\mathcal{O}(|V(G)|+|E(G)|)$. Let us choose a root node r of F.

We construct a tree (T,L) from F as follows.

1. We attach a leaf r' to the root node r of F and regard it as the parent of r and let $B_{r'} := \emptyset$.
2. For every $t \in V(F)$ with its parent t', we subdivide the edge tt' into a path $tv^t_1 \cdots v^t_{|B_t \setminus B_{t'}|}t'$, and for each $j \in \{1,\ldots,|B_t \setminus B_{t'}|\}$, we attach a leaf z^t_j to v^t_j and assign the vertices of $B_t \setminus B_{t'}$ to $L(z^t_1),\ldots,L(z^t_{|B_t \setminus B_{t'}|})$ in any order. Then remove r'.

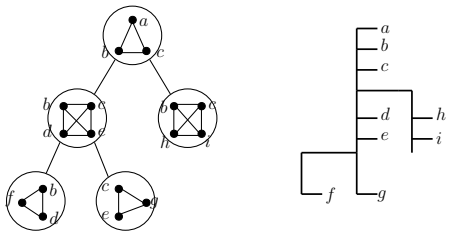


Figure 3: Constructing a branch-decomposition (T,L) of a chordal graph G of sim-width at most 1 from its tree-decomposition.

3. We transform the resulting tree into a tree of maximum degree at most 3. For every $t \in V(F)$, we do the following. Let $t_1,\ldots,t_m$ be the children of t in F. We remove t and introduce a path $w^t_1 w^t_2 \cdots w^t_m$. If t is a leaf, then we just remove it. We add an edge $w^t_1 v^t_1$, and for each $i \in \{1,\ldots,m\}$, add an edge $w^t_i v^{t_i}_{|B_{t_i} \setminus B_t|}$.

4. Let T' be the resulting tree, and we obtain a tree T from T' by smoothing all nodes of degree 2. Let (T,L) be the resulting branch-decomposition. See Figure 3 for an illustration of the construction.

We can construct (T,L) in linear time as the number of nodes in T is $\mathcal{O}(|V(G)|)$. We consider T as a rooted tree with the root $z^r_{B_r \setminus B_{t'}}$.

We claim that (T,L) has sim-width at most 1. We prove a stronger result that for every edge e of T with a vertex bipartition (A,B) associated with e, either $N_G(A) \cap B$ or $N_G(B) \cap A$ is a clique.

Claim 1. *Let e be an edge of T and let (A,B) be the vertex bipartition of G associated with e. Then either $N_G(A) \cap B$ or $N_G(B) \cap A$ is a clique.*

Proof. For convenience, we prove for T', which is the tree before smoothing. We may assume that both end nodes of e are internal nodes of T', otherwise, it is trivial. There are four types of e:

1. $e = v^t_i v^t_{i+1}$ for some $t \in V(F)$, its parent t', and $i \in \{1,\ldots,|B_t \setminus B_{t'}| - 1\}$.
2. $e = w^t_1 v^t_1$ for some $t \in V(F)$.
3. $e = v^{t_i}_{|B_{t_i} \setminus B_t|} w^t_i$ for some $t \in V(F)$ and its child t_i.
4. $e = w^t_i w^t_{i+1}$ for some $t \in V(F)$ and its child t_i.

Suppose $e = v^t_i v^t_{i+1}$ for some t and i, and let t' be the parent of t. We may assume that A is the set of all vertices assigned to the descendants of v^t_i. Note that $B_t \cap B_{t'}$ separates A and $B \setminus B_t$ in G. Therefore, for each $v \in A$, $N_G(v) \cap B$ is contained in B_t, and $N_G(A) \cap B$ is a clique. We can similarly prove for Cases 2 and 3.

Now, let $t_1,\ldots,t_m$ be the children of t, and let $w^t_1,\ldots,w^t_m$ be the vertices that were replaced from t in the algorithm. We assume $e = w^t_i w^t_{i+1}$ for some $i \in \{1,\ldots,m-1\}$. For each $j \in \{1,\ldots,m\}$, let L_j be the set of all vertices assigned to the descendants of $v^{t_j}_{|B_{t_j} \setminus B_t|}$. Without loss of generality, we may assume that $B_t \subseteq B$. We can observe that $A = \bigcup_{i+1 \le j \le m} L_j$.

Let j_1 and j_2 be integers such that $1 \leq j_1 \leq i < j_2 \leq m$. Note that B_t separates L_{j_1} and L_{j_2} in G. Therefore, for every $v \in A$, $N_G(v) \cap B$ is contained in B_t, and $N_G(A) \cap B$ is a clique, as required. $\Diamond$

We prove that when G is $(K_t \boxminus S_t)$-free for some $t \geq 2$, (T,L) has mim-width at most $t-1$.

Claim 2. *Let $t \geq 2$ be an integer. If G is $(K_t \boxminus S_t)$-free, then (T,L) has mim-width at most $t-1$.*

Proof. We show that (T,L) has mim-width at most $t-1$. Suppose for contradiction that there is an edge e of T with the vertex bipartition (A,B) of G associated with e such that $\mathrm{mim}_G(A) \geq t$. We may assume both end nodes of e are internal nodes of T.

By Claim 1, one of $N_G(A) \cap B$ and $N_G(B) \cap A$ is a clique. Without loss of generality we assume $N_G(B) \cap A$ is a clique C. Since $\mathrm{mim}_G(A) \geq t$, there is an induced matching $\{a_1b_1, \ldots, a_tb_t\}$ in $G[A,B]$ where $a_1, \ldots, a_t \in A$. Since $N_G(B) \cap A$ is a clique C, there are no edges between vertices in $\{b_1, \ldots, b_t\}$, otherwise, it creates an induced cycle of length 4. Thus, we have an induced subgraph isomorphic to $K_t \boxminus S_t$, which contradicts our assumption. We conclude that (T,L) has mim-width at most $t-1$. $\Diamond$

This concludes the proposition. $\square$

As a corollary of Proposition 3.1, we obtain the following.

Corollary 3.2. *Let $t \geq 2$ be an integer.*

(1) Given an n-vertex $(K_t \boxminus S_t)$-free chordal graph and $\sigma, \rho \subseteq \mathbb{N}$, one can solve MIN-(OR MAX-) (σ,ρ)-DOMINATION *in time $\mathcal{O}(n^{3d(t-1)+4})$ where $d = \max(d(\sigma), d(\rho))$.*

*(2) Given an n-vertex $(K_t \boxminus S_t)$-free chordal graph and a $(q \times q)$-matrix D_q, one can solve D_q-*PARTITIONING *in time $\mathcal{O}(qn^{3dq(t-1)+4})$ where $d = \max_{i,j} d(D_q[i,j])$.*

This generalizes the algorithms for interval graphs, as interval graphs are $(K_3 \boxminus S_3)$-free chordal graphs [22].

We now prove the lower bound on the mim-width of general chordal graphs. We show this for split graphs, which form a subclass of the class of chordal graphs. The Sauer-Shelah lemma [28, 30] is essential in the proof.

Theorem 3.3 (Sauer-Shelah lemma [28, 30])**.** *Let t be a positive integer and let M be an $X \times Y$ $(0,1)$-matrix such that $|Y| \geq 2$ and any two row vectors of M are distinct. If $|X| \geq |Y|^t$, then there are $X' \subseteq X$, $Y' \subseteq Y$ such that $|X'| = 2^t$, $|Y'| = t$, and all possible row vectors of length t appear in $M[X',Y']$.*

Proposition 3.4. *Let $m \geq 8$ be an integer and let $n := m + (2^m - 1)$. There is a split graph on n vertices having mim-width at least $\frac{\log_2 n}{5\log_2(\log_2 n)} - 1$.*

Proof. Let G be a split graph on the vertex bipartition (C,I) where C is a clique of size m, I is an independent set of size $2^m - 1$, and all vertices in I have pairwise distinct non-empty neighborhoods on C. We claim that every branch-decomposition of G has mim-width at least $\frac{\log_2 n}{5\log_2(\log_2 n)} - 1$.

Let (T,L) be a branch-decomposition of G. By Lemma 2.3, there is a vertex bipartition (A_1,A_2) of G associated with e satisfies that for each $i \in \{1,2\}$, $\frac{n}{3} < |A_i| \leq \frac{2n}{3}$. Without loss of generality, we may assume that $|A_1 \cap C| \geq |A_2 \cap C|$, and thus we have $\frac{m}{2} \leq |A_1 \cap C| \leq m$ and $|A_2 \cap C| \leq \frac{m}{2}$.

We prove that there are many vertices in $A_2 \cap I$ having pairwise distinct neighborhoods in $A_1 \cap C$.

Claim 3. *There are at least $|A_1 \cap C|^{\frac{m}{4\log_2 m}}$ vertices in $A_2 \cap I$ that have pairwise distinct neighborhoods in $A_1 \cap C$.*

Proof. Since $m \geq 8$, we have $2^m \geq 4 + 2m$. Therefore, we have

$$\begin{aligned} |A_2 \cap I| = |A_2| - |A_2 \cap C| &> \frac{n}{3} - \frac{m}{2} \\ &\geq \frac{m + (2^m - 1)}{3} - \frac{m}{2} \\ &\geq \frac{2^m}{3} - \frac{2+m}{6} \\ &\geq \frac{2^m}{4}. \end{aligned}$$

Observe that there are exactly $2^{|A_2 \cap C|}$ vertices in A_2 having the same set of neighbors in $A_1 \cap C$, because such vertices should have distinct neighborhoods in $A_2 \cap C$. Since $|A_2 \cap C| \leq \frac{m}{2}$, there are at least $\frac{2^m}{4} \cdot \frac{1}{2^{|A_2 \cap C|}} \geq \frac{1}{4} 2^{\frac{m}{2}}$ vertices in $A_2 \cap I$ having pairwise distinct neighborhoods on $A_1 \cap C$. As $m \geq 8$, we have $\frac{m}{2} - 2 \geq \frac{m}{4}$ and

$$2^{\frac{m}{2}-2} \geq m^{\frac{m}{4\log_2 m}}.$$

Thus, we have

$$\frac{1}{4} 2^{\frac{m}{2}} = 2^{\frac{m}{2}-2} \geq m^{\frac{m}{4\log_2 m}} \geq |A_1 \cap C|^{\frac{m}{4\log_2 m}},$$

as required. ◊

By Claim 3, there are at least $|A_1 \cap C|^{\frac{m}{4\log_2 m}}$ vertices in $A_2 \cap I$ that have pairwise distinct neighborhoods in $A_1 \cap C$. Therefore, by the Sauer-Shelah lemma, there exist $A' \subseteq A_1 \cap C$ and $B' \subseteq A_2 \cap I$ such that

- $|A'| = \lfloor \frac{m}{4\log_2 m} \rfloor$ and $|B'| = 2^{\lfloor \frac{m}{4\log_2 m} \rfloor}$ and
- all vertices in B' have pairwise distinct neighborhoods in A'.

In particular, there is an induced matching of size $\lfloor \frac{m}{4\log_2 m} \rfloor$ between A' and B'. It implies that (T,L) has mim-width at least $\frac{m}{4\log_2 m} - 1$. Since $n = 2^m + m - 1$, we have

- $\log_2 n - 1 \leq m \leq \log_2 n$.

As (T,L) was chosen arbitrary, the mim-width of G is at least

$$\frac{m}{4\log_2 m} - 1 \geq \frac{(\log_2 n) - 1}{4\log_2(\log_2 n)} - 1 \geq \frac{\log_2 n}{5\log_2(\log_2 n)} - 1. \qquad \square$$

3.2 Mim-width of co-comparability graphs

We observe similar properties for co-comparability graphs.

Theorem 3.5 (McConnell and Spinrad [23])**.** *Given a co-comparability graph G, one can output a co-comparability ordering in time $\mathcal{O}(|V(G)|+|E(G)|)$.*

Proposition 3.6. *Given a co-comparability graph G, one can in time $\mathcal{O}(|V(G)|+|E(G)|)$ output a linear branch-decomposition (T,L) with the following property:*

- *(T,L) has linear sim-width at most 1,*
- *(T,L) has linear mim-width at most $t-1$ if G is $(K_t \boxminus K_t)$-free for some integer $t \geq 2$.*

Proof. Let G be a co-comparability graph. Using Theorem 3.5, we can obtain its co-comparability ordering $v_1, v_2, \ldots, v_n$ in time $\mathcal{O}(|V(G)|+|E(G)|)$.

From this, we take a linear branch-decomposition following the co-comparability ordering. More precisely, let Q be the path $q_1 q_2 \cdots q_n$, and let T be the tree obtained from Q by adding a leaf p_i adjacent to q_i for each $i \in \{2,3,\ldots,n-1\}$. Let $p_1 = q_1$ and $p_n = q_n$. We define a function $L : V(G) \to \{p_1, p_2, \ldots, p_n\}$ such that $L(v_i) = p_i$ for each $i \in \{1,2,\ldots,n\}$. Note that (T,L) is a linear branch-decomposition of G. We show that (T,L) is a linear branch-decomposition of linear sim-width at most 1. Clearly for each leaf edge of T, its sim_G-width is at most 1. We focus on non-leaf edges in T.

We claim that for each $i \in \{2,\ldots,n-1\}$, there is no induced matching of size 2 between $\{v_1,\ldots,v_i\}$ and $\{v_{i+1},\ldots,v_n\}$. Suppose there are $i_1, i_2 \in \{1,\ldots,i\}$ and $j_1, j_2 \in \{i+1,\ldots,n\}$ such that $\{v_{i_1}v_{j_1}, v_{i_2}v_{j_2}\}$ is an induced matching of G. Without loss of generality we may assume that $i_1 < i_2$. Then we have $i_1 < i_2 < j_1$, and thus by the definition of the co-comparability ordering, v_{i_2} should be adjacent to one of v_{i_1} and v_{j_1}, which contradicts our assumption. Therefore, there is no induced matching of size 2 between $\{v_1,\ldots,v_i\}$ and $\{v_{i+1},\ldots,v_n\}$ in G. It implies that (T,L) has linear sim-width at most 1.

Now, suppose that G is $(K_t \boxminus K_t)$-free for $t \geq 2$. We prove that for every $i \in \{1,\ldots,n\}$, there are no induced matchings of size t in $G[\{v_1,\ldots,v_i\},\{v_{i+1},\ldots,v_n\}]$. For contradiction, suppose there is an induced matching $\{v_{i_1}v_{j_1},\ldots,v_{i_t}v_{j_t}\}$ in $G[\{v_1,\ldots,v_i\},\{v_{i+1},\ldots,v_n\}]$ where $i_1,\ldots,i_t \in \{1,\ldots,i\}$. We claim that $\{v_{i_1},\ldots,v_{i_t}\}$ and $\{v_{j_1},\ldots,v_{j_t}\}$ are cliques.

Claim 4. *$\{v_{i_1},\ldots,v_{i_t}\}$ and $\{v_{j_1},\ldots,v_{j_t}\}$ are cliques.*

Proof. Let $x,y \in \{1,\ldots,t\}$. Assume that $i_x < i_y$. Then $i_x < i_y < j_x$, and therefore, v_{i_y} is adjacent to either v_{i_x} or v_{j_x}. Since v_{i_y} is not adjacent to v_{j_x}, v_{i_y} is adjacent to v_{i_x}. In case when $i_y < i_x$, we also have that v_{i_y} is adjacent to v_{i_x} by the same reason. It implies that $\{v_{i_1},\ldots,v_{i_t}\}$ is a clique. By the symmetric argument, we have that $\{v_{j_1},\ldots,v_{j_t}\}$ is a clique. ◊

By Claim 4, $\{v_{i_1},\ldots,v_{i_t}\}$ and $\{v_{j_1},\ldots,v_{j_t}\}$ are cliques, and therefore, G contains $K_t \boxminus K_t$ as an induced subgraph, a contradiction. We conclude that (T,L) is a linear branch-decomposition of linear mim-width at most $t-1$. □

Corollary 3.7. *Let $t \geq 2$ be an integer.*

(1) Given an n-vertex $(K_t \boxminus K_t)$-free co-comparability graph and $\sigma,\rho \subseteq \mathbb{N}$, one can solve Min-(or Max-) (σ,ρ)-Domination *in time $\mathcal{O}(n^{3d(t-1)+4})$ where $d = \max(d(\sigma),d(\rho))$.*

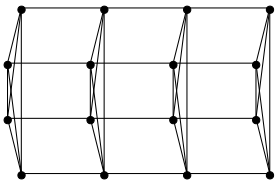


Figure 4: The (4×4) column-clique grid.

*(2) Given an n-vertex $(K_t \boxminus K_t)$-free co-comparability graph and a $(q \times q)$-matrix D_q, one can solve D_q-*PARTITIONING *in time $\mathcal{O}(qn^{3dq(t-1)+4})$ where $d = \max_{i,j} d(D_q[i,j])$.*

Note that permutation graphs do not contain $K_3 \boxminus K_3$ as induced subgraphs, because $K_3 \boxminus K_3$ is the complement of C_6, which is not a permutation graph. Therefore, Corollary 3.7 generalizes the algorithms for permutation graphs.

In the next, we show that co-comparability graphs have unbounded mim-width. For positive integers p and q, the $(p \times q)$ *column-clique grid* is the graph on the vertex set $\{v_{i,j} : 1 \leq i \leq p, 1 \leq j \leq q\}$ where

- for every $j \in \{1,\ldots,q\}$, $\{v_{1,j},\ldots,v_{p,j}\}$ is a clique,
- for every $i \in \{1,\ldots,p\}$ and $j_1, j_2 \in \{1,\ldots,q\}$, v_{i,j_1} is adjacent to v_{i,j_2} if and only if $|j_2 - j_1| = 1$,
- for $i_1, i_2 \in \{1,\ldots,p\}$ and $j_1, j_2 \in \{1,\ldots,q\}$ with $i_1 \neq i_2$ and $j_1 \neq j_2$, v_{i_1,j_1} is not adjacent to v_{i_2,j_2}.

We depict an example in Figure 4. For each $1 \leq i \leq p$, we call $\{v_{i,1},\ldots,v_{i,h}\}$ *the i-th row* of G, and define its columns similarly.

Lemma 3.8. *For integers $p,q \geq 12$, the $(p \times q)$ column-clique grid has mim-width at least* $\min(\frac{p}{4}, \frac{q}{3})$.

Proof. Let G be the $(p \times q)$ column-clique grid, and let (T,L) be a branch-decomposition of G. It is enough to show that (T,L) has mim-width at least $\min(\frac{p}{4}, \frac{q}{3})$. Let w be the mim-width of (T,L). By Lemma 2.3, G admits a vertex bipartition (A,B) where $\mathrm{mim}_G(A) \leq w$ and for each $U \in \{A,B\}$, $\frac{|V(G)|}{3} < |U| \leq \frac{2|V(G)|}{3}$. It is sufficient to show that $\mathrm{mim}_G(A) \geq \min(\frac{p}{4}, \frac{q}{3})$.

Firstly, assume that for each row R of G, $R \cap A \neq \emptyset$ and $R \cap B \neq \emptyset$. Then there is an edge between $R \cap A$ and $R \cap B$, as $G[R]$ is connected. For each i-th row R_i, we choose a pair of vertices $v_{i,a_i} \in R \cap A$ and $v_{i,b_i} \in R \cap B$ that are adjacent. We choose a subset $X \subseteq \{1,\ldots,p\}$ such that $|X| \geq \frac{p}{2}$ and every pair (v_{i,a_i}, v_{i,b_i}) in $\{(v_{i,a_i}, v_{i,b_i}) : i \in X\}$ satisfies that $a_i + 1 = b_i$. By taking the same parity of a_i's, we choose a subset $Y \subseteq X$ such that $|Y| \geq \frac{p}{4}$ and all integers in $\{a_i : i \in Y\}$ have the same parity. Then we can observe that $\{v_{i,a_i} v_{i,b_i} : i \in Y\}$ is an induced matching in $G[A,B]$, as all integers in $\{a_i : i \in Y\}$ have the same parity. Therefore, we have $\mathrm{mim}_G(A) \geq \frac{p}{4}$.

Now, we assume that there exists a row R such that R is fully contained in one of A and B. Without loss of generality, we may assume that R is contained in A. Since $|B| > \frac{|V(G)|}{3}$, there is a subset $X \subseteq \{1,\ldots,q\}$ such that $|X| > \frac{q}{3}$ and for each $i \in X$, the i-th column contains a vertex of B. For each i-th column where $i \in X$, we choose a vertex $v_{a_i,i}$ in B. It is not hard to verify

that the edges between $\{v_{a_i,t} : i \in X\}$ and the rows in R form an induced matching of size $\frac{q}{3}$ in $G[A,B]$.

Therefore, we have $d \geq \min(\frac{p}{4}, \frac{q}{3})$. □

Corollary 3.9. *For every large enough n, there is a co-comparability graph on n vertices having mim-width at least* $\sqrt{\frac{n}{12}}$.

Proof. Let $p \geq 4$ be an integer, and let $n := 12p^2$. Let G be the $(4p \times 3p)$ column-clique grid. It is not hard to see that

$$v_{1,1}, v_{2,1}, \ldots, v_{4p,1}, v_{1,2}, v_{2,2}, \ldots, v_{4p-1,3p}, v_{4p,3p}$$

is a co-comparability ordering, as each column is a clique. Thus, G is a co-comparability graph. By Lemma 3.8, $\mathrm{mimw}(G) \geq p = \sqrt{\frac{n}{12}}$. □

We remark that the two classes, $(K_t \boxminus S_t)$-free chordal graphs and $(K_t \boxminus K_t)$-free co-comparability graphs, are subclasses of the class of graphs of sim-width at most 1 and having no induced subgraph isomorphic to $K_t \boxminus K_t$ and $K_t \boxminus S_t$ for $t \geq 3$. This is because chordal graphs have no $K_2 \boxminus K_2$ induced subgraph, and co-comparability graphs have no $K_3 \boxminus S_3$ induced subgraphs by Lemma 2.1. Motivated from it, we extend these classes to classes graphs of bounded sim-width and having no $K_t \boxminus K_t$ and $K_t \boxminus S_t$ as induced subgraphs or induced minors, in Section 4.

4 Graphs of bounded sim-width

In Section 3, we proved that graphs of sim-width at most 1 contain all chordal graphs and all co-comparability graphs. A classical result on chordal graphs is that MINIMUM DOMINATING SET is NP-complete on chordal graphs [6]. So, even for this kind of locally-checkable problems, we cannot expect efficient algorithms on graphs of sim-width at most w. Therefore, to obtain a meta-algorithm for graphs of bounded sim-width encompassing many locally-checkable problems, we must impose some restrictions. We approach this problem in a way analogous to what has previously been done in the realm of rank-width [13].

It is well known that complete graphs have rank-width at most 1, but they have unbounded tree-width. Fomin, Oum, and Thilikos [13] showed that if a graph G is K_r-minor free, then its tree-width is bounded by $c \cdot \mathrm{rw}(G)$ where c is a constant depending on r. This can be utilized algorithmically, to get a result for graphs of bounded rank-width when excluding a fixed minor, as the class of problems solvable in FPT time is strictly larger when parameterized by tree-width than rank-width [20].

We will do something similar by focusing on the distinction between mim-width and sim-width. However, K_r-minor free graphs are too strong, as one can show that on K_r-minor free graphs, the tree-width of a graph is also bounded by some constant factor of its sim-width. To see this, one can use Lemma 4.5 and the result on contraction obstructions for graphs of bounded tree-width [12].

Instead of using minors, we exclude $K_t \boxminus K_t$ and $K_t \boxminus S_t$ as induced subgraphs or induced minors. The induced minor operation is rather natural because sim-width does not increase when taking induced minors; we prove this property in Subsection 4.2. In Subsection 4.3, we prove that chordal graphs having no induced minor isomorphic to $K_3 \boxminus S_3$ have unbounded rank-width. As chordal graphs have no induced minor isomorphic to $K_3 \boxminus K_3$ and they have sim-width 1,

this implies that graphs that have bounded sim-width and have no induced minor isomorphic to $K_t \boxminus K_t$ or $K_t \boxminus S_t$ for fixed t have unbounded rank-width. Therefore, such classes still extend classes of bounded rank-width.

We denote by $R(k,\ell)$ the *Ramsey number*, that is the minimum integer satisfying that every graph with at least $R(k,\ell)$ vertices contains either a clique of size k or an independent set of size ℓ. By Ramsey's Theorem [26], $R(k,\ell)$ exists for every pair of positive integers k and ℓ.

4.1 Bounding mim-width

We show the following.

Proposition 4.1. *Every graph with sim-width w and no induced minor isomorphic to $K_t \boxminus K_t$ and $K_t \boxminus S_t$ has mim-width at most $8(w+1)t^3-1$.*

Proposition 4.2. *Every graph with sim-width w and no induced subgraph isomorphic to $K_t \boxminus K_t$ and $K_t \boxminus S_t$ has mim-width at most $R(R(w+1,t),R(t,t))$.*

We remark that sometimes this Ramsey number can go down to a polynomial function depending on the underlying graphs. See discussions in Belmonte et al [2].

We first prove Proposition 4.1. We use the following result. Notice that the optimal bound of Theorem 4.3 has been slightly improved by Fox [14], and then by Balogh and Kostochka [1].

Theorem 4.3 (Duchet and Meyniel [10])**.** *For positive integers k and n, every n-vertex graph contains either an independent set of size k or a K_t-minor where $t \geq \frac{n}{2k-1}$.*

Proof of Proposition 4.1. Let G be a graph with sim-width w and no induced minor isomorphic to $K_t \boxminus K_t$ and $K_t \boxminus S_t$. Let (T,L) be a branch-decomposition of G of sim-width w. Let e be an edge of T and (A,B) be the vertex bipartition of G associated with e. We claim that $\mathrm{mim}_G(A) \leq 8(w+1)t^3-1$.

Suppose for contradiction that there is an induced matching $\{v_1w_1,\ldots,v_mw_m\}$ in $G[A,B]$ where $v_1,\ldots,v_m \in A$, $w_1,\ldots,w_m \in B$, and $m \geq 8(w+1)t^3$. Let f be the bijection from $\{v_1,\ldots,v_m\}$ to $\{w_1,\ldots,w_m\}$ such that $f(v_i)=w_i$ for each $i \in \{1,\ldots,m\}$. As $m \geq 8(w+1)t^3$, by Theorem 4.3, the subgraph $G[\{v_1,\ldots,v_m\}]$ contains either an independent set of size $2(w+1)t$, or a K_{2t^2}-minor.

First assume that $G[\{v_1,\ldots,v_m\}]$ contains a K_{2t^2}-minor. Then there is a minor model of K_{2t^2} in $G[\{v_1,\ldots,v_m\}]$, that is, there exist pairwise disjoint subsets $S_1,\ldots,S_{2t^2}$ of $\{v_1,\ldots,v_m\}$ such that

- for each $i \in \{1,\ldots,2t^2\}$, $G[S_i]$ is connected, and
- for two distinct integers $i,j \in \{1,\ldots,2t^2\}$, there is an edge between S_i and S_j.

For each $i \in \{1,\ldots,2t^2\}$, we choose a representative d_i in each $f(S_i)$ and contract each S_i to a vertex c_i. Let G_1 be the resulting graph. Then $G_1[\{c_1,\ldots,c_{2t^2}\},\{d_1,\ldots,d_{2t^2}\}]$ is an induced matching of size $2t^2$, and $\{c_1,\ldots,c_{2t^2}\}$ is a clique in G_1. By the same procedure on $\{d_1,\ldots,d_{2t^2}\}$, the subgraph $G_1[\{d_1,\ldots,d_{2t^2}\}]$ contains either an independent set of size t, or a K_t-minor. In both cases, one can observe that G_1 contains an induced minor isomorphic to $K_t \boxminus K_t$ or $K_t \boxminus S_t$, hence we obtain a contradiction.

Now assume that $G[\{v_1,\ldots,v_m\}]$ contains an independent set $\{c_1,\ldots,c_{2(w+1)t}\}$, and for each $i \in \{1,\ldots,2(w+1)t\}$, let $d_i := f(c_i)$. By Theorem 4.3, $G[\{d_1,\ldots,d_{2(w+1)t}\}]$ contains either an independent set of size $w+1$ or a K_t-minor. In the former case, we obtain an induced matching of size $w+1$ between A and B in G, contradicting to the assumption that $\mathrm{sim}_G(A) \leq w$. In the latter case, we obtain an induced minor isomorphic to $K_t \boxminus S_t$, contradiction.

We conclude that $\mathrm{mim}_G(A) \leq 8(w+1)t^3 - 1$. Since e is arbitrary, G has mim-width at most $8(w+1)t^3-1$. □

In a similar manner we can prove Proposition 4.2.

Proof of Proposition 4.2. One can easily modify from the proof of Proposition 4.1 by replacing the application of Theorem 4.3 with the Ramsey's Theorem to find a clique or an independent set. □

As a corollary, we obtain the following. In general, we do not have a generic algorithm to find a decomposition, and thus we assume that the decomposition is given as an input.

Corollary 4.4. *Let $w \geq 1$ and $t \geq 2$ be integers.*

(1) Given an n-vertex graph of sim-width at most w and having induced minor isomorphic to neither $K_t \boxminus K_t$ nor $K_t \boxminus S_t$, and $\sigma,\rho \subseteq \mathbb{N}$, one can solve MIN-(OR MAX-) (σ,ρ)-DOMINATION *in time $\mathcal{O}(n^{3dt'+4})$ where $d = \max(d(\sigma),d(\rho))$ and $t' = 8(w+1)t^3-1$.*

*(2) Given an n-vertex graph of sim-width at most w and having induced minor isomorphic to neither $K_t \boxminus K_t$ nor $K_t \boxminus S_t$, and a $(q \times q)$-matrix D_q, one can solve D_q-*PARTITIONING *in time $\mathcal{O}(qn^{3dqt'+4})$ where $d = \max_{i,j} d(D_q[i,j])$ and $t' = 8(w+1)t^3-1$.*

(3) Given an n-vertex graph of sim-width at most w and having induced subgraph isomorphic to neither $K_t \boxminus K_t$ nor $K_t \boxminus S_t$, and $\sigma,\rho \subseteq \mathbb{N}$, one can solve MIN-(OR MAX-) (σ,ρ)-DOMINATION *in time $\mathcal{O}(n^{3dt'+4})$ where $d = \max(d(\sigma),d(\rho))$ and $t' = R(R(w+1,t),R(t,t))$.*

*(4) Given an n-vertex graph of sim-width at most w and having induced subgraph isomorphic to neither $K_t \boxminus K_t$ nor $K_t \boxminus S_t$, and a $(q \times q)$-matrix D_q, one can solve D_q-*PARTITIONING *in time $\mathcal{O}(qn^{3dqt'+4})$ where $d = \max_{i,j} d(D_q[i,j])$ and $t' = R(R(w+1,t),R(t,t))$.*

4.2 Induced minors

We show that the sim-width of a graph does not increase when taking an induced minor. This is one of the main motivations to consider this parameter.

Lemma 4.5. *The sim-width of a graph does not increase when taking an induced minor.*

Proof. Clearly, the sim-width of a graph does not increase when removing a vertex. We prove for contractions.

Let G be a graph, $v_1v_2 \in E(G)$, and let (T,L) be a branch-decomposition of G of sim-width w for some positive integer w. Let z be the contracted vertex in G/v_1v_2. We claim that G/v_1v_2 admits a branch-decomposition of G of sim-width at most w. We may assume that G is connected and has at least 3 vertices. For G/v_1v_2, we obtain a branch-decomposition (T',L') as follows:

- Let T' be the tree obtained from T by removing $L(v_2)$, and smoothing its neighbor.

- Let L' be the function from $V(G/v_1v_2)$ to the set of leaves of T' such that $L'(v) = L(v)$ for $v \in V(G/v_1v_2) \setminus \{z\}$ and $L'(z) = L(v_1)$.

Let e_1 and e_2 be the two edges of T incident with the neighbor of $L(v_2)$, but not incident with $L(v_2)$. Let e_{cont} be the edge of T' obtained by smoothing. By construction, all edges of $E(T') \setminus \{e_{cont}\}$ are contained in T.

Claim 5. *For each $e \in E(T') \setminus \{e_{cont}\}$, the sim_{G/v_1v_2}-width of e in (T',L') is at most the sim_G-width of e in (T,L).*

Proof. Let $e \in E(T') \setminus \{e_{cont}\}$. Let (A,B) be the vertex bipartition of G/v_1v_2 associated with e. Without loss of generality, we may assume $z \in A$. Suppose there exists an induced matching $\{a_1b_1,\ldots,a_mb_m\}$ in G/v_1v_2 with $a_1,\ldots,a_m \in A$ and $b_1,\ldots,b_m \in B$. Let (A',B') be the vertex bipartition of G associated with e where $v_1 \in A'$. We will show that there is also an induced matching in G of same size between A' and B'.

We have either $v_2 \in A'$ or $v_2 \in B'$. If $z \notin \{a_1,\ldots,a_m\}$, then $\{a_1b_1,\ldots,a_mb_m\}$ is also an induced matching between A' and B' in G. Without loss of generality, we may assume that $z = a_1$.

Case 1. $v_2 \in A'$**.**

Proof. In G, one of v_1 and v_2, say v', is adjacent to b_1. Also, v_1 and v_2 are not adjacent to any of $\{a_2,\ldots,a_m,b_1,\ldots,b_m\}$. Therefore, $\{v'b_1,a_2b_2,\ldots,a_mb_m\}$ is an induced matching in G between A' and B', as required. □

Case 2. $v_2 \in B'$**.**

Proof. In this case, $\{v_1v_2,a_2b_2,\ldots,a_mb_m\}$ is an induced matching between A' and B', because v_1 and v_2 are not adjacent to any of $\{a_2,\ldots,a_m,b_2,\ldots,b_m\}$. □

It shows that the sim_{G/v_1v_2}-width of e in (T',L') is at most the sim_G-width of e in (T,L). ◊

In a similar manner, we can show that the sim_{G/v_1v_2}-width of e_{cont} in (T',L') is at most the minimum of the sim_{G/v_1v_2}-width of e_1 and e_2 in (T,L). Thus, we conclude that $\mathrm{simw}(G/v_1v_2) \leq \mathrm{simw}(G)$. □

4.3 Unbounded rank-width

We show that the Hsu-clique chain graph depicted in Figure 5 is chordal, but does not contain $K_2 \boxminus K_2$ or $K_3 \boxminus S_3$ as an induced minor. Belmonte and Vatshelle [3, Lemma 16] showed that a $(p \times q)$ Hsu-clique chain graph has rank-width at least $\frac{p}{3}$ when $q = 3p+1$. So, our algorithmic applications based on Proposition 4.1 or Proposition 4.2 are beyond algorithmic applications of graphs of bounded rank-width.

We formally define Hsu-clique chain graphs. For positive integers p,q, the $(p \times q)$ *Hsu-clique chain grid* is the graph on the vertex set $\{v_{i,j} : 1 \leq i \leq p, 1 \leq j \leq q\}$ where

- for every $j \in \{1,\ldots,q\}$, $\{v_{1,j},\ldots,v_{p,j}\}$ is a clique,

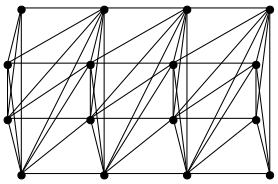


Figure 5: The (4×4) Hsu-clique chain graph.

- for every $i_1, i_2 \in \{1, \ldots, p\}$ and $j \in \{1, \ldots, q-1\}$, $v_{i_1,j}$ is adjacent to $v_{i_2,j+1}$ if and only if $i_1 \leq i_2$,
- for $i_1, i_2 \in \{1, \ldots, p\}$ and $j_1, j_2 \in \{1, \ldots, q\}$, v_{i_1,j_1} is not adjacent to v_{i_2,j_2} if $|j_1 - j_2| > 1$.

Proposition 4.6. *Let $w \geq 1$ and $t \geq 2$ be integers. The class of graphs of sim-width w and having induced minor isomorphic to neither $K_t \boxminus S_t$ nor $K_t \boxminus K_t$ has unbounded rank-width.*

Proof. For this, we provide that Hsu-clique chain grids are chordal and having no induced minor isomorphic to $K_t \boxminus S_t$ or $K_t \boxminus K_t$. Since Hsu-clique chain grids have unbounded rank-width [3], it proves the proposition.

Let G be the $(p \times q)$ Hsu-clique chain graph for some positive integers p and q.

First show that G is chordal. Suppose for contradiction that G contains an induced cycle $C = c_1 c_2 \cdots c_m c_1$ where $m \geq 4$. Since each column of G is a clique, all vertices of C are contained in two consecutive columns. But also, since each column contains at most 2 vertices, we have $m = 4$, and two consecutive vertices of C are contained in one column. Without loss of generality, we assume c_1, c_2 are in the i-th column for some $i \in \{1, \ldots, q-1\}$, and c_3, c_4 are in the $(i+1)$-th column. This implies that there is an induced matching of size 2 between two columns if we ignore edges in each column. However, this is not possible from the construction. Therefore, G is chordal.

If G contains an induced minor isomorphic to $K_2 \boxminus K_2$, which is an induced cycle of length 4, then G contains an induced subgraph isomorphic to an induced cycle of length ℓ for some $\ell \geq 4$. This contradicts the fact that G is chordal. So, G has no induced minor isomorphic to $K_2 \boxminus K_2$.

We claim that G has no induced minor isomorphic to $K_3 \boxminus S_3$. For contradiction, suppose there is an induced minor model η of $K_3 \boxminus S_3$ in G. Let $H := K_3 \boxminus S_3$. For each $v \in V(H)$, let $I_v = \{j : v_{i,j} \in V(\eta(v))\}$. Let $I := I_{v_1^1} \cup I_{v_2^1} \cup I_{v_3^1}$, and let ℓ and r be the smallest and greatest integers in I, respectively. Let $x, y \in \{v_1^1, v_2^1, v_3^1\}$ such that

1. $V(\eta(x))$ contains a vertex in the ℓ-th column, but for $z \in \{v_1^1, v_2^1, v_3^1\} \setminus \{x\}$, $V(\eta(z))$ has no vertex whose row index is higher than all vertices in $V(\eta(x))$ in the ℓ-th column,
2. similarly, $V(\eta(y))$ contains a vertex in the r-th column, but for $z \in \{v_1^1, v_2^1, v_3^1\} \setminus \{x\}$, $V(\eta(z))$ has no vertex whose row index is lower than all vertices in $V(\eta(y))$ in the r-th column.

As $\{v_1^1, v_2^1, v_3^1\}$ is a clique, it is easy to observe that $I_x \cup I_y = I$. Let ℓ be the integer such that $v_\ell^1 \in \{v_1^1, v_2^1, v_3^1\} \setminus \{x, y\}$. By the choice of x and y, every vertex in $V(\eta(v_\ell^2))$ has a neighbor in $V(\eta(x)) \cup V(\eta(y))$. Thus, it contradicts the assumption that G contains H as an induced minor. □

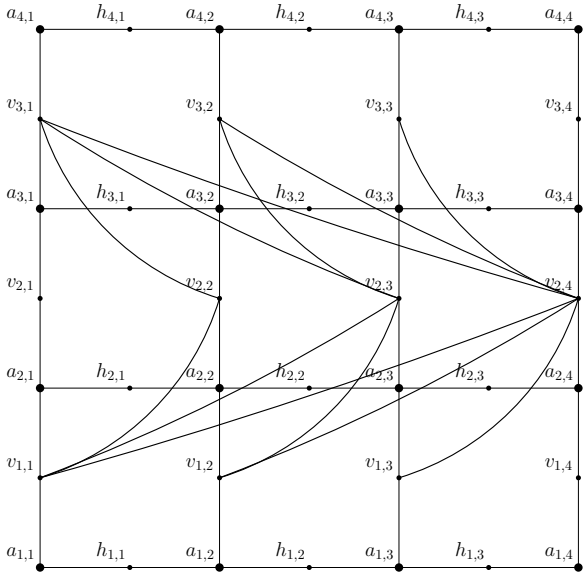


Figure 6: The circle graph G_4 of Lemma 5.4

5 Lower bound of sim-width on circle graphs

In this section, we construct a set of circle graphs with unbounded sim-width and no triangle, using the approach similar to the one by Mengel [24]. As $\mathrm{simw}(G) \leq \mathrm{mimw}(G)$ for every graph G, the example also implies that the class of circle graphs has unbounded mim-width.

Theorem 5.1. *Circle graphs have unbounded sim-width, and thus have unbounded mim-width.*

For an integer $d \geq 0$, a graph G is *d-degenerate* if every subgraph of G contains a vertex of degree at most d. The following lemma asserts that every d-degenerate graph with large treewidth also has a large mim-width.

Lemma 5.2 (Vatshelle's Thesis [32]; See also Mengel [24])**.** *For every d-degenerate graph G,* $\mathrm{mimw}(G) \geq \frac{\mathrm{tw}(G)+1}{3(d+1)}$.

Given a bipartite graph with large mim-width, we do not lose much of mim-width after adding edges to one part of the bipartition.

Lemma 5.3 (Mengel [24])**.** *Let H be a bipartite graph with a bipartition (A,B), and let G be a graph obtained from H by adding some edges in $H[A]$. Then* $\mathrm{mimw}(G) \geq \frac{\mathrm{mimw}(H)}{2}$.

For an integer $k \geq 1$ and a graph H, a graph G is a *k-subdivision* of H if G can be built from H by replacing every edge of H with a path of length $k+1$ such that those paths replacing edges of H are internally vertex-disjoint.

Now we prove the main lemma of this section.

Lemma 5.4. *For every integer $k \geq 2$, there is a circle graph G_k satisfying the following.*

1. *G_k contains no K_3 as a subgraph.*
2. *G_k contains a 1-subdivision of a $(k \times k)$-grid as a subgraph. In particular,* $\mathrm{tw}(G_k) \geq k$.
3. $\mathrm{mimw}(G_k) \geq \frac{k+1}{18}$.

Proof. Let H be a 1-subdivision of a $(k \times k)$-grid. It is easy to see that H is 2-degenerate and has tree-width at least k, since it contains a $(k \times k)$-grid as a minor. By Lemma 5.2, $\mathrm{mimw}(H) \geq \frac{k+1}{9}$.

For $i, j \in \{1, \ldots, k\}$, let $a_{i,j}$ be the vertex on the i-th row and j-th column of the $(k \times k)$-grid. For $i \in \{1, \ldots, k\}$ and $j \in \{1, \ldots, k-1\}$, let $h_{i,j}$ be the vertex of degree 2 adjacent to $a_{i,j}$ and $a_{i,j+1}$ in H. For $i \in \{1, \ldots, k-1\}$ and $j \in \{1, \ldots, k\}$, let $v_{i,j}$ be the vertex of degree 2 adjacent to $a_{i,j}$ and $a_{i+1,j}$ in H. Let us define $A := \{a_{i,j} : i, j \in \{1, \ldots, k\}\}$, $B_h := \{h_{i,j} : i \in \{1, \ldots, k\}, j \in \{1, \ldots, k-1\}\}$, and $B_v := \{v_{i,j} : i \in \{1, \ldots, k-1\}, j \in \{1, \ldots, k\}\}$. Then H is a bipartite graph with the bipartition $(A, B_h \cup B_v)$. Let $B = B_h \cup B_v$.

Before we describe in detail, we briefly explain how we will construct a circle representation of H. For two points a and b on a circle, we denote by $\overline{ab}$ the chord whose end points are a and b. Let $p_1, \ldots, p_{2k^2}$ be distinct $2k^2$ points on the circle in clockwise order. We regard each vertex in A as a chord joining two points p_{2i} and p_{2i-1} for some $i \in \{1, \ldots, k^2\}$. Since every vertex in B has a degree 2 in H, the chord representing a vertex in B will be represented by a chord intersecting two disjoint chords $\overline{p_{2i}p_{2i-1}}$ and $\overline{p_{2j}p_{2j-1}}$ for some $i \neq j$ that represents its neighbors. Since the chords representing vertices in B may cross each other, the resulting circle graph will be a graph obtained from H by adding some edges in B. By Lemma 5.3, this graph has mim-width at least $\frac{\mathrm{mimw}(H)}{2} \geq \frac{k+1}{18}$.

We explain how we can assign chords representing vertices in A not to induce K_3 in B.

Let C be a circle and $c_1, \ldots, c_{6k^2}$ be distinct $6k^2$ points on C in clockwise order. For $i, j \in \{1, \ldots, k\}$, $a_{i,j}$ is represented as follows:

- if i is odd, then $a_{i,j}$ is represented by a chord intersecting $c_{6(i-1)k+6j-5}$ and $c_{6(i-1)k+6j}$,
- if i is even, then $a_{i,j}$ is represented by a chord intersecting $c_{6ik-6j+1}$ and $c_{6ik-6j+6}$.

Remark that we assign chords representing vertices $a_{1,1}, \ldots, a_{1,k}, a_{2,k}, \ldots, a_{2,1}, \ldots$ in clockwise order. We assign chords for vertices in B_h as follows.

- For an odd integer $i \in \{1, \ldots, k\}$ and an integer $j \in \{1, \ldots, k-1\}$, a chord connecting $c_{6(i-1)k+6j-1}$ and $c_{6(i-1)k+6j+2}$ represents a vertex $h_{i,j} \in B_h$.
- For an even integer $i \in \{1, \ldots, k\}$ and an integer $j \in \{1, \ldots, k-1\}$, a chord connecting $c_{6ik-6j+2}$ and $c_{6ik-6j-1}$ represents a vertex $h_{i,j} \in B_h$.

We assign chords for vertices in B_v as follows. Note that we assign chords to vertices in B_v so that for $i \in \{1, \ldots, k-2\}$ and $j_1, j_2 \in \{1, \ldots, k\}$, chords representing two vertices v_{i,j_1} and v_{i+1,j_2} cross if and only if either i is odd and $j_1 < j_2$ or i is even and $j_1 > j_2$.

- For an odd integer $i \in \{1, \ldots, k-1\}$ and an integer $j \in \{1, \ldots, k\}$, a chord connecting $c_{6(i-1)k+6j-2}$ and $c_{6ik+6(k-j)+3}$ represents a vertex $v_{i,j} \in B_v$.
- For an even integer $i \in \{1, \ldots, k-1\}$ and an integer $j \in \{1, \ldots, k\}$, a chord connecting $c_{6ik-6j+4}$ and $c_{6ik+6j-3}$ represents a vertex $v_{i,j} \in B_v$.

Let G_k be the circle graph obtained by this procedure. See Figure 6 which describes G_4. Since every new edge of G_k joins v_{i,j_1} and v_{i+1,j_2} for some $i \in \{1,\ldots,k-2\}$ and $j_1, j_2 \in \{1,\ldots,k\}$, it is easy to see that G_k contains no K_3. Note that G_k is a graph obtained from H by adding some edges in B. By Lemma 5.3, the resulting circle graph has mim-width at least $\frac{\mathrm{mimw}(H)}{2} \geq \frac{k+1}{18}$. □

Proof of Theorem 5.1. By Proposition 4.2, there is an increasing function $f : \mathbb{N} \to \mathbb{N}$ such that every graph G with no induced subgraph isomorphic to $K_3 \boxminus K_3$ and $K_3 \boxminus S_3$ has mim-width at most $f(\mathrm{simw}(G))$. Since the circle graph G_k of Lemma 5.4 contains no K_3 as a subgraph, we have $f(\mathrm{simw}(G_k)) \geq \mathrm{mimw}(G_k) \geq \frac{k+1}{18}$. It follows that $\mathrm{simw}(G_k)$ should be large for sufficiently large k. □

6 Concluding remarks

In this paper, we showed that every fixed LC-VSVP problem can be solved in XP time parameterized by t on $(K_t \boxminus S_t)$-free chordal graphs and $(K_t \boxminus K_t)$-free co-comparability graphs. We further give a mim-width bound on the class of graphs that have sim-width at most w and do not contain $K_t \boxminus S_t$ and $K_t \boxminus K_t$ as induced minors or induced subgraphs.

Vatshelle [32] asked whether there is a function f such that given a graph G, one can in XP time parameterized by $\mathrm{mimw}(G)$ compute a branch-decomposition of mim-width at most $f(k)$. This still remains an open problem. We asked the same question for sim-width.

Question 1. *Does there exist a function f such that, given a graph G, one can in XP time parameterized by the sim-width k of G compute a decomposition of sim-width $f(k)$?*

We observed that one cannot expect XP algorithms for MINIMUM DOMINATING SET parameterized by sim-width. However, we know that one can solve MAXIMUM INDEPENDENT SET or 3-COLORING in polynomial time on both chordal graphs and co-comparability graphs, which are all known classes of constant sim-width. We ask whether those problems are NP-complete on graphs of sim-width 1 or not.

Question 2. *Is Maximum Independent Set NP-complete on graphs of sim-width at most* 1*? Also, is* 3*-Coloring NP-complete on graphs of sim-width at most* 1*?*

It would be interesting to find more classes having constant sim-width, but unbounded mim-width. We propose some possible classes, that are also presented in Figure 2.

Question 3. *Do weakly chordal graphs or AT-free graphs have constant sim-width?*

We showed that MINIMUM DOMINATING SET can be solved in time $n^{\mathcal{O}(t)}$ on $(K_t \boxminus S_t)$-free chordal graphs, but we could not obtain an FPT algorithm. We ask whether it is W[1]-hard or not.

Question 4. *Is* MINIMUM DOMINATING SET *on chordal graphs W[1]-hard parameterized by the maximum t such that the given graph has an $K_t \boxminus S_t$ induced subgraph?*

References

[1] J. Balogh and A. Kostochka. Large minors in graphs with given independence number. *Discrete Math.*, 311(20):2203 – 2215, 2011.

[2] R. Belmonte, P. Heggernes, P. van t Hof, A. Rafiey, and R. Saei. Graph classes and ramsey numbers. *Discrete Appl. Math.*, 173:16 – 27, 2014.

[3] R. Belmonte and M. Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theoret. Comput. Sci.*, 511:54–65, 2013.

[4] J. R. S. Blair and B. Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation*, volume 56 of *IMA Vol. Math. Appl.*, pages 1–29. Springer, New York, 1993.

[5] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Comput. Sci.*, 209:1–45, 1998.

[6] K. S. Booth and J. H. Johnson. Dominating sets in chordal graphs. *SIAM J. Comput.*, 11(1):191–199, 1982.

[7] S. Bova, F. Capelli, S. Mengel, and F. Slivovsky. On compiling cnfs into structured deterministic dnnfs. In *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, pages 199–214, 2015.

[8] B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theoret. Comput. Sci.*, 511:66–76, 2013.

[9] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

[10] P. Duchet and H. Meyniel. On hadwiger's number and the stability number. In B. Bollobs, editor, *Graph Theory Proceedings of the Conference on Graph Theory, Cambridge*, volume 62 of *North-Holland Mathematics Studies*, pages 71 – 73. North-Holland, 1982.

[11] B. Dushnik and E. W. Miller. Partially ordered sets. *Amer. J. Math.*, 63:600–610, 1941.

[12] F. V. Fomin, P. Golovach, and D. M. Thilikos. Contraction obstructions for treewidth. *J. Combin. Theory Ser. B*, 101(5):302–314, 2011.

[13] F. V. Fomin, S. Oum, and D. M. Thilikos. Rank-width and tree-width of H-minor-free graphs. *European J. Combin.*, 31(7):1617–1628, 2010.

[14] J. Fox. Complete minors and independence number. *SIAM J. Discrete Math.*, 24(4):1313–1321, 2010.

[15] S. Gaspers, C. H. Papadimitriou, S. H. Sther, and J. A. Telle. On Satisfiability Problems with a Linear Structure. In J. Guo and D. Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[16] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Combin. Theory Ser. B*, 16:47–56, 1974.

[17] P. Golovach and J. Kratochvíl. Computational complexity of generalized domination: a complete dichotomy for chordal graphs. In *Graph-theoretic concepts in computer science*, volume 4769 of *Lecture Notes in Comput. Sci.*, pages 1–11. Springer, Berlin, 2007.

[18] P. A. Golovach, P. Heggernes, M. M. Kanté, D. Kratsch, S. H. Sæther, and Y. Villanger. Output-polynomial enumeration on graphs of bounded (local) linear mim-width. *Algorithmica*, To appear, 2017.

[19] P. A. Golovach, J. Kratochvíl, and O. Suchý. Parameterized complexity of generalized domination problems. *Discrete Appl. Math.*, 160(6):780–792, 2012.

[20] P. Hliněný, S. Oum, D. Seese, and G. Gottlob. Width parameters beyond tree-width and their applications. *The Computer Journal*, 51(3):326–362, 2008.

[21] W. L. Hsu and T. H. Ma. Substitution decomposition on chordal graphs and applications. In *ISA '91. Algorithms (Taipei, 1991)*, volume 557 of *Lecture Notes in Comput. Sci.*, pages 52–60. Springer, Berlin, 1991.

[22] C. G. Lekkerkerker and J. C. Boland. Representation of a finite graph by a set of intervals on the real line. *Fund. Math.*, 51:45–64, 1962/1963.

[23] R. M. McConnell and J. P. Spinrad. Modular decomposition and transitive orientation. *Discrete Math.*, 201(1-3):189–241, 1999.

[24] S. Mengel. Lower bounds on the mim-width of some graph classes. *Discrete Appl. Math.*, To appear, 2017.

[25] S. Oum and P. Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

[26] F. P. Ramsey. On a problem of formal logic. *Proc. London Math. Soc.*, s2-30:264–286, 1930.

[27] S. H. Sæther, J. A. Telle, and M. Vatshelle. Solving #sat and MAXSAT by dynamic programming. *J. Artif. Intell. Res. (JAIR)*, 54:59–82, 2015.

[28] N. Sauer. On the density of families of sets. *J. Combin. Theory Ser. A*, 13(1):145 – 147, 1972.

[29] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

[30] S. Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific J. Math.*, 41(1):247–261, 1972.

[31] J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Discrete Math.*, 10(4):529–550, 1997.

[32] M. Vatshelle. New width parameters of graphs. *Ph.D. thesis, University of Bergen*, 2012.

Chapter 8

Mim-width III. Graph powers and generalized distance domination problems.

Lars Jaffke, O-joung Kwon, Torstein J. F. Strømme and Jan Arne Telle. In *Theoretical Computer Science*, Volume 796, 2019, Pages 216-236.

A preliminary version appeared in IPEC 2018, LIPIcs 115, Pages 6:1–6:14, 2018, and is available online with DOI 10.4230/LIPIcs.IPEC.2018.6.

Mim-Width III. Graph Powers and Generalized Distance Domination Problems*

Lars Jaffke†[1], O-joung Kwon‡[2,3], Torstein J. F. Strømme[1], and Jan Arne Telle[1]

[1]Department of Informatics, University of Bergen, Norway.
{lars.jaffke, torstein.stromme, jan.arne.telle}@uib.no
[2]Department of Mathematics, Incheon National University, South Korea.
[3]Discrete Mathematics Group, Institute for Basic Science (IBS), Daejeon, South Korea.
ojoungkwon@gmail.com

Abstract

We generalize the family of (σ,ρ) problems and locally checkable vertex partition problems to their distance versions, which naturally captures well-known problems such as DISTANCE-r DOMINATING SET and DISTANCE-r INDEPENDENT SET. We show that these distance problems are in XP parameterized by the structural parameter *mim-width*, and hence polynomial-time solvable on graph classes where mim-width is bounded and quickly computable, such as k-trapezoid graphs, Dilworth k-graphs, (circular) permutation graphs, interval graphs and their complements, convex graphs and their complements, k-polygon graphs, circular arc graphs, complements of d-degenerate graphs, and H-graphs if given an H-representation. We obtain these results by showing that taking any power of a graph never increases its mim-width by more than a factor of two. To supplement these findings, we show that many classes of (σ,ρ) problems are W[1]-hard parameterized by mim-width + solution size.

We show that powers of graphs of tree-width $w-1$ or path-width w and powers of graphs of clique-width w have mim-width at most w. These results provide new classes of bounded mim-width. We prove a slight strengthening of the first statement which implies that, surprisingly, LEAF POWER graphs which are of importance in the field of phylogenetic studies have mim-width at most 1.

*The paper is based on extended abstracts that appeared in STACS 2018 [20] and IPEC 2018 [16]. The first part of the series, 'Mim-Width I. Induced Path Problems' [18], and the second part of the series, 'Mim-Width II. The Feedback Vertex Set Problem' [19], are based on extended abstracts that appeared in IPEC 2017 [17] and STACS 2018 [20].

†Supported by the Bergen Research Foundation (BFS).

‡Supported by IBS-R029-C1, and the National Research Foundation of Korea (NRF) grant funded by the Ministry of Education (No. NRF-2018R1D1A1B07050294), and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant DISTRUCT, agreement No. 648527). Part of the research took place while Kwon was at Logic and Semantics, Technische Universität Berlin, Berlin, Germany.

1 Introduction

Telle and Proskurowski [30] defined the (σ,ρ)-domination problems, or (σ,ρ) problems for short, and the more general *locally checkable vertex partitioning* problems (LCVP). In (σ,ρ)-domination problems, feasible solutions are vertex sets with constraints on how many neighbors each vertex of the graph has in the set. The framework generalizes important and well-studied problems such as MAXIMUM INDEPENDENT SET and MINIMUM DOMINATING SET, as well as PERFECT CODE, MINIMUM SUBGRAPH WITH MINIMUM DEGREE d and a multitude of other problems. See Table 1. Bui-Xuan, Telle and Vatshelle [7] showed that (σ,ρ)-domination and locally checkable vertex partitioning problems can be solved in time XP parameterized by *mim-width*, if we are given a corresponding decomposition tree. Roughly speaking, the structural parameter mim-width measures how easy it is to decompose a graph along vertex cuts inducing a bipartite graph with small maximum induced matching size [31].

In this paper, we consider distance versions of problems related to independence and domination, such as DISTANCE-r INDEPENDENT SET and DISTANCE-r DOMINATING SET. The DISTANCE-r INDEPENDENT SET problem, also studied under the names r-SCATTERED SET and r-DISPERSION (see e.g. [2] and the references therein), asks to find a set of at least k vertices whose vertices have pairwise distance strictly longer than r. Agnarsson et al. [1] pointed out that it is identical to the original INDEPENDENT SET problem on the r-th power[1] graph G^r of the input graph G, and also showed that for fixed r, it can be solved in linear time for interval graphs, and circular arc graphs. The DISTANCE-r DOMINATING SET problem was introduced by Slater [29] and Henning et al. [15]. They as well observed that it is identical to solve the original DOMINATING SET problem on the r-th power graph. Slater presented a linear-time algorithm to solve DISTANCE-r DOMINATING SET problem on forests.

We generalize all of the (σ,ρ)-domination and LCVP problems to their distance versions, which naturally captures DISTANCE-r INDEPENDENT SET and DISTANCE-r DOMINATING SET. While the original problems put constraints on the size of the immediate neighborhood of a vertex, we consider the constraints to be applied to the ball of radius r around it. Consider for instance the MINIMUM SUBGRAPH WITH MINIMUM DEGREE d problem; where the original problem is asking for the smallest (in terms of number of vertices) subgraph of minimum degree d, we are instead looking for the smallest subgraph such that for each vertex there are at least d vertices at distance at least 1 and at most r. In the PERFECT CODE problem, the target is to choose a subset of vertices such that each vertex has exactly one chosen vertex in its closed neighborhood. In the distance-r version of the problem, we replace the closed neighborhood by the closed r-neighborhood. This problem is known as PERFECT r-CODE, and was introduced by Biggs [4] in 1973.

We show that all these distance problems are in XP parameterized by mim-width if a decomposition tree is given. One of the main results of the paper is of structural nature, namely that for any positive integer r the mim-width of a graph power G^r is at most twice the mim-width of G. It follows that we can reduce the distance-r version of a (σ,ρ)-domination problem to its non-distance variant by taking the graph power G^r, while preserving small mim-width.

One negative aspect of the mim-width parameter is that we do not know an XP algorithm computing mim-width. The problem of deciding whether a given graph has a decomposition tree of mim-width w is NP-complete in general and W[1]-hard parameterized by w. Determining the

[1] For a graph G and an integer r, the *r-th power of G*, denoted by G^r, is the graph that has the same vertex set as G, with each pair of distinct vertices being adjacent if their distance in G is at most r.

optimal mim-width is not in APX unless NP = ZPP, making it unlikely to have a polynomial-time constant-factor approximation algorithm [28].

However, for several graph classes we are able to find a decomposition tree of constant mim-width in polynomial time, using the results of Belmonte and Vatshelle [3]. These include; permutation graphs, convex graphs and their complements, interval graphs and their complements (all of which have *linear* mim-width 1); (circular k-) trapezoid graphs, circular permutation graphs, Dilworth-k graphs, k-polygon graphs, circular arc graphs and complements of d-degenerate graphs. Fomin, Golovach and Raymond [13] showed that we can find linear decomposition trees of constant mim-width for the very general class of H-graphs in polynomial time, *given* an H-representation of the input graph.[2] For all of the above graph classes, our results imply that the distance-r (σ,ρ)-domination and LCVP problems are polynomial time solvable.

Graphs represented by intersections of objects in some model are often closed under taking powers. For instance, interval graphs, and generally d-trapezoid graphs [1, 12], circular arc graphs [1, 27], and leaf power graphs (by definition) are such graphs. We refer to [5, Chapter 10.6] for a survey of such results. For these classes, we already know that the distance-r version of a (σ,ρ)-domination problem can be solved in polynomial time. However, this closure property does not always hold; for instance, permutation graphs are not closed under taking powers. Our result implies that to obtain such algorithmic results, we do not need to know that these classes are closed under taking powers; it is sufficient to know that classes have bounded mim-width. To the best of our knowledge, for the most well-studied distance-r (σ,ρ)-domination problem, DISTANCE-r DOMINATING SET, we obtain the first polynomial time algorithms on Dilworth k-graphs, convex graphs and their complements, complements of interval graphs, k-polygon graphs, H-graphs (given an H-representation of the input graph), and complements of d-degenerate graphs.

We give results that expand our knowledge of the expressive power of mim-width. We show that powers of graphs of tree-width $w-1$ or path-width w and powers of graphs of clique-width w have mim-width at most w. In fact, the statement we prove is a slight strengthening, namely: Given a nice tree decomposition of width w, all of whose join bags have size at most w, or a clique-width w-expression of a graph, one can output a decomposition tree of mim-width w of its k-th power in polynomial time. The former implies that leaf power graphs, of importance in the field of phylogenetic studies [8], have mim-width 1. These graphs are known to be strongly chordal and there has recently been interest in delineating the difference between strongly chordal graphs and leaf power graphs, on the assumption that this difference was not very big [22, 24]. Our result actually implies a large difference, as it was recently shown by Mengel that there are strongly chordal split graphs of mim-width linear in the number of vertices [23].

The natural question to ask after obtaining an XP algorithm is whether we can do better, e. g. can we show that for all fixed r, the distance-r (σ,ρ)-domination problems are in FPT? Fomin et al. [13] answered this in the negative by showing that (the standard, i.e. distance-1 variants of) MAXIMUM INDEPENDENT SET, MINIMUM DOMINATING SET and MINIMUM INDEPENDENT DOMINATING SET problems are W[1]-hard parameterized by (linear) mim-width + solution size. We modify their reductions to extend these results to several families of (σ,ρ)-domination problems, including the maximization variants of INDUCED MATCHING, INDUCED d-REGULAR SUBGRAPH and INDUCED SUBGRAPH OF MAX DEGREE $\leq d$, the

[2] For a formal definition of H-graphs, see Definition 18. We would like to remark that it is NP-complete to decide whether a graph is an H-graph whenever H is not a cactus [9].

minimization variants of TOTAL DOMINATING SET and d-DOMINATING SET and both the maximization and the minimization variant of DOMINATING INDUCED MATCHING.

The remainder of the paper is organized as follows. In Section 2 we introduce the (σ,ρ)-domination problems and define their distance-r generalization, and we give a short introduction to mim-width. In Section 3 we show that the mim-width of a graph grows by at most a factor 2 when taking (arbitrary large) powers and prove bounds on the mim-width of powers of graphs of bounded tree-width and clique-width. We discuss algorithmic consequences for (σ,ρ) problems and more generally, LCVP problems, in Section 4, and in Section 5 we present the above mentioned lower bounds. Finally, we give some concluding remarks in Section 6.

2 The Main Concepts

In this section, we introduce the main concepts of the paper. That is, in Section 2.2 we introduce the family of distance-r (σ,ρ)-domination problems, and in Section 2.3 we give a short introduction to the graph parameter mim-width and several of its aspects that are of importance to this work. Before we proceed, we introduce the basic terminology and notation used throughout the paper.

2.1 Preliminaries

We let the set of natural numbers be $\mathbb{N} = \{0,1,2,\ldots\}$, and the positive natural numbers be $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. For a set S and a given property ψ, we denote by S_ψ the biggest subset of S where ψ is satisfied for all elements. For instance, $\mathbb{N}^+_{\leq k}$ denotes the set $\{1,2,\ldots k\}$. For this particular property, we also use the shorthand $[k] \coloneqq \mathbb{N}^+_{\leq k}$. For a set X, we denote by $\binom{X}{k}$ the set of all size-k subsets of X.

Basic Notation for Graphs. A graph G is a pair of a vertex set $V(G)$ and an edge set $E(G) \subseteq \binom{V(G)}{2}$. For an edge $\{u,v\} \in E(G)$, we use the shorthand notation 'uv'. For a set of edges $F \subseteq E(G)$, we denote by $V(F)$ the set of vertices that are contained in the edges of F, i.e. $V(F) \coloneqq \bigcup_{uv \in F} \{u,v\}$. A *cut* of a graph is a 2-partition of its vertex set.

(Induced) Subgraphs. For graphs G and H we say that H is a *subgraph* of G, denoted by $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For a vertex set X, we denote by $G[X]$ the subgraph of G *induced* by X, i.e. $G[X] \coloneqq (X, E(G) \cap \binom{X}{2})$. We use the notation $G - X \coloneqq G[V(G) \setminus X]$ and for a set of edges $F \subseteq E(G)$, $G - F \coloneqq (V(G), E(G) \setminus F)$. For a vertex $v \in V(G)$ (an edge $e \in E(G)$), we use the shorthand $G - x \coloneqq G - \{x\}$ ($G - e \coloneqq G - \{e\}$).

Neighborhoods. Let G be a graph. For a vertex $v \in V(G)$, we denote by $N_G(v)$ the *open neighborhood* of v, i.e. $N_G(v) \coloneqq \{w \mid vw \in E(G)\}$, and by $N_G[v]$ the *closed neighborhood* of v, i.e. $N_G[v] \coloneqq \{v\} \cup N_G(v)$. The *degree* of v is the size of its open neighborhood, i.e. $\deg_G(v) \coloneqq |N_G(v)|$. For a set of vertices $X \subseteq V(G)$, we let $N_G(X) \coloneqq \bigcup_{v \in X} N_G(x) \setminus X$ and $N_G[X] \coloneqq N_G(X) \cup X$. We use the shorthand notations 'N' and 'deg' for 'N_G' and '$\deg_G$', respectively, if G is clear from the context.

Connectivity/Distance. A graph G is called *connected* if for each 2-partition (X,Y) of $V(G)$ with $X \neq \emptyset$ and $Y \neq \emptyset$, there is an edge $xy \in E(G)$ such that $x \in X$ and $y \in Y$. A connected graph all of whose vertices have degree two is called a *cycle*. A graph that does not contain a cycle as a subgraph is called a *forest* and a connected forest is a *tree*. The vertices of degree

one in a tree are called the *leaves* and the remaining vertices the *internal* vertices. A tree of maximum degree two is called a *path* and the leaves of a path are called *endpoints*. A tree T is called a *caterpillar* if it contains a subgraph $P \subseteq T$ such that P is a path and each vertex in $V(T) \setminus V(P)$ has a neighbor in $V(P)$. The *length* of a path P is $|E(P)|$, i.e. the number of its edges. Given a graph G and two of its vertices $u, v \in V(G)$, the *distance* from u to v in G, denoted by $\text{DIST}_G(u,v)$, is the smallest length of any path $P \subseteq G$ with endpoints u and v.

Matchings. A set of edges $M \subseteq E(G)$ is called a *matching*, if no pair of edges in M shares an endpoint. A matching is called *induced* if $G[V(M)] = (V(M), M)$, i.e. if there are no other edges than the edges in M in the subgraph of G induced by $V(M)$.

Cliques/Independence/Domination/Bipartite Graphs. A graph G is called *complete*, if $E(G) = \binom{V(G)}{2}$. A set of vertices $C \subseteq V(G)$ is called a *clique* if $G[C]$ is a complete graph. A set of vertices $I \subseteq V(G)$ is called an *independent set* if $E(G[I]) = \emptyset$. A set of vertices $D \subseteq V(G)$ is called a *dominating set* if $N_G[D] = V(G)$. A graph G is called *bipartite* if there is a 2-partition (X,Y) of $V(G)$ such that X and Y are independent sets, and we call a bipartite graph G with partition (X,Y) *complete bipartite* if $E(G) = \{xy \mid x \in X, y \in Y\}$. For two disjoint vertex sets $X, Y \subseteq V(G)$, we denote by $G[X,Y]$ the *bipartite subgraph of G induced by* (X,Y), i.e. $G[X,Y] \coloneqq (X \cup Y, E(G) \cap \{xy \mid x \in X, y \in Y\})$.

Multigraphs/Subdivisions. Let G and H be two graphs. A bijection $\varphi \colon V(G) \to V(H)$ is called an *isomorphism from G to H* if for all $u, v \in E(G)$, $uv \in E(G)$ if and only if $\varphi(u)\varphi(v) \in E(H)$. If there is an isomorphism from G to H then say that G and H are *isomorphic*.

A *multigraph* is a pair of a set of vertices $V(G)$ and a *multiset* of size-2 subsets of $V(G)$, called the edges of G and denoted by $E(G)$. For a (multi-) graph H, we let $|H| \coloneqq |V(G)|$ and $||H|| \coloneqq |E(G)|$. Let G be a (multi-) graph. An *edge subdivision* of an edge $e \in E(G)$ with endpoints u and v is the operation of adding to G a new vertex x and replacing the edge e by a path with endpoints u and v, consisting of an edge between u and x and an edge between x and v. We call the vertex x a *subdivision vertex*. A graph G' is called a *subdivision of G* if it can be obtained from G by a series of edge subdivisions.

2.2 Distance-r (σ,ρ)-Domination Problems

Let σ and ρ be finite or co-finite subsets of the natural numbers $\sigma, \rho \subseteq \mathbb{N}$. For a graph G, a vertex set $S \subseteq V(G)$ is a (σ,ρ)-*dominating set*, or simply (σ,ρ) *set* if

- for each vertex $v \in S$ it holds that $|N(v) \cap S| \in \sigma$, and
- for each vertex $v \in V(G) \setminus S$ it holds that $|N(v) \cap S| \in \rho$.

For instance, a $(\{0\}, \mathbb{N})$ set is an independent set as there are no edges between the vertices in the set, and we do not care about adjacencies between S and $V(G) \setminus S$. For another example, a $(\mathbb{N}, \mathbb{N}^+)$-set is a dominating set as we require that for each vertex in $V(G) \setminus S$, it has at least one neighbor in S.

There are three types of (σ,ρ)-domination problems: minimization, maximization, and existence. We denote the problem of finding a minimum (maximum) (σ,ρ) set as the MIN-(σ,ρ) DOMINATION (MAX-(σ,ρ) DOMINATION) problem, or simply MIN-(σ,ρ) (MAX-(σ,ρ)) problem, and we refer to Table 1 for examples of well-studied problems expressible in this framework.

σ	ρ	d	Standard name	W[1]-Hard
$\{0\}$	$\mathbb{N}$	1	Independent set	$\circ_{\max}$
$\mathbb{N}$	$\mathbb{N}^+$	1	Dominating set	$\circ_{\min}$
$\{0\}$	$\mathbb{N}^+$	1	Maximal Independent set	$\circ_{\min}$
$\mathbb{N}^+$	$\mathbb{N}^+$	1	Total Dominating set	$\star_{\min}$
$\{0\}$	$\{0,1\}$	2	Strong Stable set or 2-Packing	
$\{0\}$	$\{1\}$	2	Perfect Code or Efficient Dom. set	
$\{0,1\}$	$\{0,1\}$	2	Total Nearly Perfect set	
$\{0,1\}$	$\{1\}$	2	Weakly Perfect Dominating set	
$\{1\}$	$\{1\}$	2	Total Perfect Dominating set	
$\{1\}$	$\mathbb{N}$	2	Induced Matching	$\star_{\max}$
$\{1\}$	$\mathbb{N}^+$	2	Dominating Induced Matching	$\star_{\max}$, $\star_{\min}$
$\mathbb{N}$	$\{1\}$	2	Perfect Dominating set	
$\mathbb{N}$	$\{d,d+1,...\}$	d	d-Dominating set	$\star_{\min}$
$\{d\}$	$\mathbb{N}$	$d+1$	Induced d-Regular Subgraph	$\star_{\max}$
$\{d,d+1,...\}$	$\mathbb{N}$	d	Subgraph of Min Degree $\geq d$	
$\{0,1,...,d\}$	$\mathbb{N}$	$d+1$	Induced Subg. of Max Degree $\leq d$	$\star_{\max}$

Table 1: Some vertex subset properties expressible as (σ,ρ)-sets, with $\mathbb{N} = \{0,1,...\}$ and $\mathbb{N}^+ = \{1,2,...\}$. Column d shows $d = \max(d(\sigma),d(\rho))$. For each problem, at least one of the minimization, the maximization and the existence problem is NP-complete. For problems marked with $\star_{\max}$ ($\star_{\min}$) in the rightmost column, W[1]-hardness of the maximization (minimization) problem parameterized by mim-width + solution size is shown in the present paper. For problems marked with $\circ_{\max}$ ($\circ_{\min}$) the W[1]-hardness of maximization (minimization) in the same parameterization was shown by Fomin et al. [13].

We naturally generalize (σ,ρ)-domination problems to their distance-r version: For a graph G and a vertex $v \in V(G)$, let $N^r_G(v)$ denote the *ball of radius r around v*, i. e.

$$N^r_G(v) \coloneqq \{w \in V(G) \setminus \{v\} \mid \textsc{dist}_G(v,w) \leq r\}.$$

Let σ and ρ be finite or co-finite subsets of $\mathbb{N}$. Then, a vertex set $S \subseteq V(G)$ is called a *distance-r* (σ,ρ)*-dominating set*, if

- for each vertex $v \in S$ it holds that $|N^r(v) \cap S| \in \sigma$, and
- for each vertex $v \in V(G) \setminus S$ it holds that $|N^r(v) \cap S| \in \rho$.

We associate with these sets minimization, maximization, and existence problems in the natural way.

The *d-value* of a distance-r (σ,ρ) problem is a constant which will ultimately affect the runtime of the algorithm. For a set $\mu \subseteq \mathbb{N}$, the value $d(\mu)$ should be understood as the highest value in $\mathbb{N}$ we need to enumerate in order to describe μ. Hence, if μ is finite, it is simply the maximum value in μ, and if μ is co-finite, it is the maximum natural number *not* in μ (1 is added for technical reasons).

Definition 1 (d-value)**.** Let $d(\mathbb{N}) = 0$. For every non-empty finite or co-finite set $\mu \subseteq \mathbb{N}$, let $d(\mu) = 1 + \min(\max\{x \mid x \in \mu\}, \max\{x \mid x \in \mathbb{N} \setminus \mu\})$.

For a given distance-r (σ,ρ) problem $\Pi_{\sigma,\rho}$, its d-value is defined as $d(\Pi_{\sigma,\rho}) \coloneqq \max\{d(\sigma),d(\rho)\}$, see column d in Table 1.

2.3 Mim-width and Applications

Maximum induced matching width, or *mim-width* for short, was introduced in the Ph. D. thesis of Vatshelle [31], used implicitly by Belmonte and Vatshelle [3], and is a structural graph parameter defined over *decomposition trees* (sometimes called *branch decompositions*), similar to graph parameters such as *rank-width* and *module-width*. Decomposition trees naturally appear in divide and conquer style algorithms where one recursively partitions the pieces of a problem into two parts. When the algorithm is at the point where it combines solutions of its subproblems to form a full solution, the structure of the cuts are (unsurprisingly) important to the runtime; this is especially true for dynamic programming when one needs to store multiple partial solutions at each intermediate node. We will briefly introduce the necessary machinery here, but for a more comprehensive introduction we refer the reader to [31].

A graph of maximum degree at most 3 is called *subcubic*. A *decomposition tree* for a graph G is a pair $(T,\mathscr{L})$ where T is a subcubic tree and $\mathscr{L} : V(G) \to L(T)$ is a bijection between the vertices of G and the leaves of T. Each edge $e \in E(T)$ naturally represents a cut of G, i. e. a 2-partition of $V(G)$: Let T_1^e and T_2^e denote the two connected components of $T-e$. For $i \in [2]$, let A_i^e denote the set of vertices that are mapped to leaves in T_i^e via $\mathscr{L}$. Then, (A_1^e,A_2^e) is a cut of G which in the following we refer to as the *cut associated with* e. One way to measure the complexity of a cut is by considering the maximum size of an induced matching in the bipartite subgraph of G that it induces. For a vertex set $A \subseteq V(G)$, we let $\mathrm{mim}_G(A)$ denote the maximum size of an induced matching in $G[A,V(G)\setminus A]$, the bipartite subgraph of G induced by $(A,V(G)\setminus A)$. Note that mim_G is symmetric, i.e. $\mathrm{mim}_G(A) = \mathrm{mim}_G(V(G)\setminus A)$.

Definition 2 (mim-width)**.** Let G be a graph, and let $(T,\mathscr{L})$ be a decomposition tree for G. The *mim-width* of $(T,\mathscr{L})$ is defined as

$$\mathrm{mimw}_G(T,\mathscr{L}) \coloneqq \max_{e\in E(T)} \mathrm{mim}_G(A_1^e),$$

where for an edge $e \in E(T)$, (A_1^e,A_2^e) denotes the cut associated with e. The *mim-width of the graph* G, denoted $\mathrm{mimw}(G)$, is the minimum value of $\mathrm{mimw}_G(T,\mathscr{L})$ over all possible decomposition trees $(T,\mathscr{L})$. The *linear mim-width of the graph* G is the minimum value of $\mathrm{mimw}_G(T,\mathscr{L})$ over all possible decomposition trees $(T,\mathscr{L})$ where T is a caterpillar.

Note that the definition of a decomposition tree $(T,\mathscr{L})$ allows T to have nodes of degree two. Suppose there is a node $t \in V(T)$ of degree two with incident edges e_1 and e_2. Then, up to renaming the sets, we have that $A_1^{e_1} = A_1^{e_2}$ and $A_2^{e_1} = A_2^{e_2}$, where for $i \in [2]$, $(A_1^{e_i},A_2^{e_i})$ is the cut associated with e_i. Let t_i denote the endpoint of e_i other than t. The observation we just made implies that the decomposition tree $(T',\mathscr{L})$, where T' is obtained from T by removing t and making t_1 and t_2 adjacent, is equivalent to $(T,\mathscr{L})$ both in terms of their mim-width, as well as their structural properties. To that end, for ease of exposition, we will assume in algorithmic applications that a decomposition tree does not have degree two nodes, while when proving bounds on the mim-width of some decomposition tree, we may allow them to have degree two nodes.

Given a decomposition tree, one can subdivide an arbitrary edge and let the newly created vertex be the root of T, in the following denoted by r. For two nodes $t,t' \in V(T)$, we say that t' is a *descendant* of t if t lies on the path from r to t' in T. For $t \in V(T)$, we denote by V_t the set of vertices that are mapped to a leaf that is a descendant of t, i.e. $V_t := \{v \in V(G) \mid \mathscr{L}(v) = t' \text{ where } t' \text{ is a leaf descendant of } t \text{ in } T\}$. We let $\overline{V_t} := V(G) \setminus V_t$ and $G_t := G[V_t]$.

In previous work, Bui-Xuan et al. [7] and Belmonte and Vatshelle [3] showed that all (σ,ρ) problems can be solved in time $n^{\mathscr{O}(w)}$ where w denotes the mim-width of a decomposition tree that is provided as part of the input. As we state the runtime bounds more tightly than what can be read from the statements in the original works, we provide a sketch of the proof of the following result due to [3, 7], mainly focusing on aspects that affect the runtime rather than the correctness of the algorithms.

Proposition 3 ([3, 7])**.** *There is an algorithm that given a graph G and a decomposition tree $(T,\mathscr{L})$ of G with $w := mimw_G(T,\mathscr{L})$ solves each (σ,ρ) problem Π with $d := d(\Pi)$*

(i) in time $\mathscr{O}(n^{4+2d\cdot w})$, if T is a caterpillar, and

(ii) in time $\mathscr{O}(n^{4+3d\cdot w})$, otherwise.

Sketch of the Proof. For a positive integer d, and a set $A \subseteq V(G)$, we call two subsets $X \subseteq A$ and $Y \subseteq A$ *d-neighbor equivalent w.r.t. A*, and we write $X \equiv_A^d Y$, if

$$\forall v \in V(G) \setminus A\colon \ \min(d,|X \cap N(v)|) = \min(d,|Y \cap N(v)|).$$

We denote by $\mathrm{nec}(\equiv_A^d)$ the number of equivalence classes of $\equiv_A^d$, and for $X \subseteq A$, by $\mathrm{rep}_A^d(X)$ an equivalence class representative for X.

The crucial insight that makes the dynamic programming algorithm work within the claimed runtime bound is: when tabulating the partial solutions with respect to a node $t \in V(T)$, it suffices to store one optimum partial solution for each pair $(R_t,R_{\overline{t}})$, where R_t is an equivalence class representative for $\equiv_{V_t}^d$ and $R_{\overline{t}}$ is an equivalence class representative for $\equiv_{\overline{V_t}}^d$. In this way, an upper bound on $\mathrm{nec}(\equiv_A^d)$ gives an upper bound on the number of table entries to be considered.

Now, let ℓ be a leaf of T and $v \in V(G)$ be such that $\mathscr{L}(v) = \ell$. Then, there are two equivalence classes for $\equiv_{\{v\}}^d$, and $d+1$ equivalence classes for $\equiv_{V(G)\setminus\{v\}}^d$, so we only have $\mathscr{O}(d)$ partial solutions to consider. For an internal node $t \in V(T)$ with children a and b, we can compute the necessary partial solutions in the following way. For each triple $(R_a,R_b,R_{\overline{t}})$ of an equivalence class representative R_a of $\equiv_{V_a}^d$, an equivalence class representative R_b of $\equiv_{V_b}^d$, and an equivalence class representative $R_{\overline{t}}$ of $\equiv_{\overline{V_t}}^d$, compute $R_t = \mathrm{rep}_{V_t}^d(R_a \cup R_b)$, $R_{\overline{a}} = \mathrm{rep}_{\overline{V_a}}^d(R_b \cup R_{\overline{t}})$, and $R_{\overline{b}} = \mathrm{rep}_{\overline{V_b}}^d(R_a \cup R_{\overline{t}})$, and update the table entry for $(R_t,R_{\overline{t}})$ according to the partial solution obtained from combining the partial solution stored for $(R_a,R_{\overline{a}})$ with the one stored for $(R_b,R_{\overline{b}})$.

Let $\mathrm{nec}_d(T,\mathscr{L}) = \max_{t\in V(T)} \max\{\mathrm{nec}(\equiv_{V_t}^d), \mathrm{nec}(\equiv_{\overline{V_t}}^d)\}$. We argue that for each internal node $t \in V(T)$, all table entries can be computed in time $\mathscr{O}(\mathrm{nec}_d(T,\mathscr{L})^3 \cdot n^3)$. It can be shown [7, Lemma 1] that for a set $A \subseteq V(G)$, a set of canonical equivalence class representatives of $\equiv_A^d$ can be enumerated in time $\mathscr{O}(\mathrm{nec}(\equiv_A^d) \cdot \log(\mathrm{nec}(\equiv_A^d)) \cdot n^2)$, and given a set $X \subseteq A$, one can find a pointer to $\mathrm{rep}_A^d(X)$ in time $\mathscr{O}(\log(\mathrm{nec}(\equiv_A^d)) \cdot |X| \cdot n)$.

The former computation is invoked once, taking time at most $\mathscr{O}(\mathrm{nec}_d(T,\mathscr{L}) \cdot \log(\mathrm{nec}_d(T,\mathscr{L})) \cdot n^2)$. Next, there are at most $\mathrm{nec}_d(T,\mathscr{L})^3$ triples of equivalence class representatives to consider, and the remaining computations take time at most $\mathscr{O}(n^3)$, applying the latter of the two

aforementioned tasks (finding a pointer to an equivalence class representative), and noting that each representative is of size at most $\mathcal{O}(n)$, and that $\log(\mathrm{nec}_d(T,\mathscr{L})) \leq \mathcal{O}(n)$. Therefore, we compute all table entries for $t \in V(T)$ in time at most $\mathcal{O}(\mathrm{nec}_d(T,\mathscr{L})^3 \cdot n^3)$. As $|V(T)| = \mathcal{O}(n)$, and the computation for a leaf node takes constant time, the whole algorithm takes time at most $\mathcal{O}(\mathrm{nec}_d(T,\mathscr{L})^3 \cdot n^4)$.

In [3, Lemma 2] it is shown that $\mathrm{nec}(\equiv_A^d) \leq n^{d \cdot \mathrm{mim}(A)}$. Hence the dynamic programming algorithms that we sketched above run in time $\mathcal{O}(n^{4+3d \cdot w})$, where $w = \mathrm{mimw}(T,\mathscr{L})$, proving item (ii). For item (i), we observe that for an internal node $t \in V(T)$ with children a and b such that b is a leaf node, the number of triples to consider reduces to $\mathcal{O}(\mathrm{nec}_d(T,\mathscr{L})^2)$, as there are only $\mathcal{O}(1)$ many equivalence class representatives to consider for $\equiv_{V_b}^d$. If T is a caterpillar, then each internal node of T is of that shape, implying that in this case, the algorithms run in time $\mathcal{O}(n^{4+2d \cdot w})$. □

3 Mim-width on Graph Powers

In this section we discuss the structural results of this work, regarding the mim-width of graph powers. These are formally defined as follows.

Definition 4 (The r-th Power of a Graph)**.** Let r be a positive integer and let $G = (V,E)$ be a graph. The r-th power of G, denoted G^r, is the graph obtained from G by adding, for each pair of distinct non-adjacent vertices $u,v \in V(G)$ with $\mathrm{DIST}_G(u,v) \leq r$, the edge uv.

We show in Section 3.1 that taking an (arbitrarily large) power of a graph only increases its mim-width by a factor of at most two. In Section 3.2 we prove results concerning powers of graphs of bounded tree-width and clique-width.

3.1 Arbitrary Graphs

Theorem 5. *Let r be a positive integer and let G be a graph. Then,* $\mathrm{mimw}(G^r) \leq 2 \cdot \mathrm{mimw}(G)$. *Moreover, any decomposition tree of G of mim-width w has mim-width at most $2w$ for G^r.*

Proof. Assume that there is a decomposition tree of mim-width w for the graph G. We show that the same decomposition tree has mim-width at most $2w$ for G^r.

We consider a cut $(A,\overline{A})$ associated with some edge of the decomposition tree. Let M be a maximum induced matching across the cut for G^r. To prove our claim, it suffices to construct an induced matching across the cut M' in G such that $|M'| \geq \frac{|M|}{2}$.

We begin by noticing that for an edge $uv \in M$, the distance between u and v is at most r in G. For each such edge $uv \in M$, we let P_{uv} denote some shortest path between u and v in G (including the endpoints u and v).

Claim 5.1. Let $uv, wx \in M$ be two distinct edges of the matching. Then P_{uv} and P_{wx} are vertex disjoint.

Proof. We may assume that $u,w \in A$ and $v,x \in \overline{A}$. Now assume for the sake of contradiction there exists a vertex $y \in V(P_{uv}) \cap V(P_{wx})$. Because both paths have length at most r, we have that $\mathrm{DIST}_G(u,y) + \mathrm{DIST}_G(y,v) \leq r$, and $\mathrm{DIST}_G(w,y) + \mathrm{DIST}_G(y,x) \leq r$. Adding these together, we get

$$\mathrm{DIST}_G(u,y) + \mathrm{DIST}_G(y,v) + \mathrm{DIST}_G(w,y) + \mathrm{DIST}_G(y,x) \leq 2r.$$

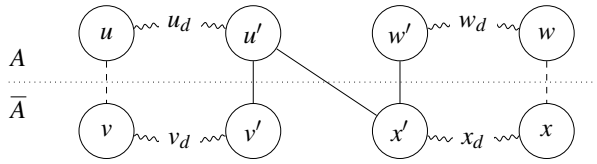


Figure 1: Structure of two paths P_{uv} and P_{wx} when the edge $u'x'$ exists in G. Dashed edges appear in G^r, solid edges appear in G, squiggly lines are (shortest) paths existing in G (possibly of length 0, and possibly crossing back and forth across the cut).

Since uv and wx are both in M, there cannot exist edges ux and wv in G^r. Hence, their distance in G is strictly greater than r, i.e. $\text{DIST}_G(u,y) + \text{DIST}_G(y,x) \geq \text{DIST}_G(u,x) > r$, and $\text{DIST}_G(w,y) + \text{DIST}_G(y,v) > r$. Putting these together, we obtain our contradiction:

$$\text{DIST}_G(u,y) + \text{DIST}_G(y,x) + \text{DIST}_G(w,y) + \text{DIST}_G(y,v) > 2r$$

This concludes the proof of the claim. ⌟

Because for each $uv \in M$, one endpoint of P_{uv} lies in A and the other one lies in $\overline{A}$, there must exist at least one point at which the path crosses from A to $\overline{A}$. For each $uv \in M$ with $u \in A$ and $v \in \overline{A}$, we let $u'v' \in E(P_{uv})$ denote an edge in G such that $u' \in A$ and $v' \in \overline{A}$.

We plan to construct our matching M' by picking a subset of such edges. However, we cannot simply take all of them, since some pairs may be incompatible in the sense that they will not form an induced matching across the cut $(A,\overline{A})$. We examine the structures that arise when two such edges $u'v'$ and $w'x'$ are incompatible, and cannot both be included in the same induced matching across the cut. For easier readability, we let α_d be a shorthand notation for $\text{DIST}_G(\alpha,\alpha')$ for $\alpha \in \{u,v,w,x\}$.

Claim 5.2. Let $uv, wx \in M$ with $\{u,w\} \subseteq A$ and $\{v,x\} \subseteq \overline{A}$ be two distinct edges of M and let $u'v'$ and $w'x'$ be edges on the shortest paths as defined above. If there is an edge $u'x' \in E(G)$, then all of the following hold. See Figure 1.

(a) $u_d + x_d = r$

(b) $u_d + v_d = w_d + x_d = r - 1$

(c) $w_d = u_d - 1$

Proof. (a) Since ux is not an edge in G^r, the distance between u and x must be at least $r+1$ in G, and so $u_d + x_d$ must be at least r. It remains to show that $u_d + x_d \leq r$ for equality to hold. Similarily to the proof of Claim 5.1, we know that P_{uv} and P_{wx} both are of length at most r. We get

$$u_d + v_d + w_d + x_d \leq 2r - 2 \tag{1}$$

The '-2' at the end is because we do not include the length contributed by edges $u'v'$ and $w'x'$ in our sum. Now assume for the sake of contradiction that $u_d + x_d \geq r+1$. Then we get that

$$v_d + w_d \leq 2r - 2 - r - 1 = r - 3$$

Because $\text{DIST}_G(v',w') \leq 3$ (follow the edges $u'v' \to u'x' \to w'x'$), this implies that $\text{DIST}_G(v,w) \leq r$, and the edge vw would hence exist in G^r. This contradicts that uv and wx were both in the same induced matching M.

(b) Assume for the sake of contradiction that $u_d + v_d \leq r-2$. Then, rather than Equation 1, we get the following bound

$$u_d + v_d + w_d + x_d \leq 2r-3$$

By (a) we know that $u_d + x_d = r$, so by a similar argument as above we get that $v_d + w_d \leq r-3$, obtaining a contradiction. An analogous argument holds for $w_d + x_d$.

(c) This follows immediately by substituting (a) into (b). ⌟

We will now construct our induced matching M'. (Recall for the following arguments that $u \in A$ and $v \notin A$.) We construct two candidates for M', and we will pick the biggest one. First, we construct M'_0 by including $u'v'$ for each edge $uv \in M$ where $\text{DIST}_G(u,u')$ is even. Symetrically, M'_1 is constructed by including $u'v'$ if $\text{DIST}_G(u,u')$ is odd. Clearly, at least one of M'_0 and M'_1 contains at least $\frac{|M|}{2}$ edges. It remains to show that M' indeed forms an induced matching across the cut $(A,\overline{A})$ in G.

Consider two distinct edges $u'v'$ and $w'x'$ from M'. By Claim 5.1, the corresponding paths P_{uv} and P_{wx} are vertex disjoint. If there is an edge violating that $u'v'$ and $w'x'$ are both in the same induced matching, it must be either $u'x'$ or $v'w'$. Without loss of generality we may assume it is an edge of the type $u'x'$. By Claim 5.2(c), we then have that the parities of $\text{DIST}_G(u,u')$ and $\text{DIST}_G(w,w')$ are different. However, by the construction of M', this is not possible. This concludes the proof. □

3.2 Graphs of Bounded Tree-Width or Clique-Width

In [31, Section 4.2], it is shown that any graph of clique-width w or tree-width $w-1$ has mim-width at most w. Theorem 5 hence implies that any power of a graph of clique-width w or tree-width $w-1$ has mim-width at most $2w$. In this section we give tighter bounds on the mim-width of powers of graphs of bounded clique-width and powers of graphs of bounded tree-width.

In particular, we show that r-th powers of graphs of tree-width $w-1$, path-width w, or clique-width w all have mim-width at most w. We begin by proving the bound for graphs of bounded tree-width with the following lemma capturing the essential property used in the proof.

Lemma 6. *Let r and w be positive integers and let (A,B,C) be a vertex partition of graph G such that there are no edges between A and C and B has size w. Let $H := G^r$. Then,* $\text{mim}_H(A \cup B) \leq w$.

Proof. Let $B := \{b_1, b_2, \ldots, b_w\}$. It is clear that for $v \in A \cup B$ and $z \in C$, $\text{DIST}_G(v,z) \leq r$ if and only if there exists $i \in \{1,2,\ldots,w\}$ such that $\text{DIST}_G(v,b_i) + \text{DIST}_G(z,b_i) \leq r$.

Suppose for a contradiction that $\text{mim}_H(A \cup B) > w$. Let $\{y_1z_1, y_2z_2, \ldots, y_tz_t\}$ be an induced matching of size $t \geq w+1$ in $H[A \cup B, C]$. There are distinct integers $t_1, t_2 \in \{1,2,\ldots,t\}$ and an integer $j \in \{1,2,\ldots,w\}$ such that

$$\text{DIST}_G(y_{t_1},b_j) + \text{DIST}_G(z_{t_1},b_j) \leq r \text{ and } \text{DIST}_G(y_{t_2},b_j) + \text{DIST}_G(z_{t_2},b_j) \leq r.$$

Then we have either $\text{DIST}_G(y_{t_1},b_j) + \text{DIST}_G(z_{t_2},b_j) \leq r$ or $\text{DIST}_G(y_{t_2},b_j) + \text{DIST}_G(z_{t_1},b_j) \leq r$, which contradicts the assumption that $y_{t_1}z_{t_2}$ and $y_{t_2}z_{t_1}$ are not edges in H. We conclude that $\text{mim}_H(A \cup B) \leq w$. □

Definition 7. A *tree decomposition* of a graph G is a pair $(T, \mathscr{B})$ consisting of a tree T and a family $\mathscr{B} = \{B_t\}_{t \in V(T)}$ of sets $B_t \subseteq V(G)$, called *bags*, satisfying the following three conditions:

(i) $V(G) = \bigcup_{t \in V(T)} B_t$,

(ii) for every edge uv of G, there exists a node t of T such that $u, v \in B_t$, and

(iii) for $t_1, t_2, t_3 \in V(T)$, $B_{t_1} \cap B_{t_3} \subseteq B_{t_2}$ whenever t_2 is on the path from t_1 to t_3 in T.

The *width* of a tree decomposition $(T, \mathscr{B})$ is $\max\{|B_t| - 1 : t \in V(T)\}$. The *tree-width* of G is the minimum width over all tree decompositions of G. A tree decomposition $(T, \mathscr{B} = \{B_t\}_{t \in V(T)})$ is a *nice tree decomposition* with root node $\mathfrak{r} \in V(T)$ if T is a rooted tree with root node $\mathfrak{r}$, and every node t of T is one of the following:

(1) A *leaf node*, i.e. t is a leaf of T and $B_t = \emptyset$.

(2) An *introduce node*, i.e. t has exactly one child t' and $B_t = B_{t'} \cup \{v\}$ for some $v \in V(G) \setminus B_{t'}$.

(3) A *forget node*, i.e. t has exactly one child t' and $B_t = B_{t'} \setminus \{v\}$ for some $v \in B_{t'}$.

(4) A *join node*, i.e. t has exactly two children t_1 and t_2, and $B_t = B_{t_1} = B_{t_2}$.

Theorem 8. *Let r and w be positive integers and G be a graph that admits a nice tree decomposition of width w, all of whose join bags are of size at most w. Then the r-th power of G has mim-width at most w. Furthermore, given such a nice tree decomposition, we can output a decomposition tree of mim-width at most w in polynomial time.*

Proof. Let $H \coloneqq G^r$, and let $(T, \mathscr{B} = \{B_t\}_{t \in V(T)})$ be a nice tree decomposition of G of width w, all of whose join bags have size at most w, with root node $\mathfrak{r}$. We may assume that $B_{\mathfrak{r}} = \emptyset$ and subsequently that $\mathfrak{r}$ is a forget or a join node. (Otherwise, we add a path of forget nodes on top of $\mathfrak{r}$ and make the last node the new root of T.)

We obtain a decomposition tree $(T', \mathscr{L})$ as follows:

- Let T'' be the tree obtained from T by, for each forget node $t \in V(T)$ forgetting a vertex v, attaching a leaf node ℓ_v to t, and assigning $\mathscr{L}(v) \coloneqq \ell_v$.
- We obtain T' from T'' by deleting degree 1 nodes that are not assigned by $\mathscr{L}$.

Since $\mathfrak{r}$ is either a forget node or a join node in $(T, \mathscr{B})$, $\mathfrak{r}$ has not been removed in the second step, so $\mathfrak{r} \in V(T')$, and $\mathfrak{r}$ has degree 2 in T'. Furthermore, since for each vertex $v \in V(G)$, there is a unique forget node forgetting v in $(T, \mathscr{B})$, and all leaves that are not assigned by $\mathscr{L}$ have been removed, the map $\mathscr{L}$ constructed above is a bijection. Thus, $(T', \mathscr{L})$ is a (rooted) decomposition tree with root node $\mathfrak{r}$.

We consider a cut $(V_t, \overline{V_t})$ for some node $t \in V(T')$ in the rooted decomposition tree. If t is a leaf node, then $\mathrm{mim}_H(V_t) \leq 1$. Assume t is an internal node, then t also appears in $(T, \mathscr{B})$. Note that V_t consists precisely of all vertices that have been forgotten below t in $(T, \mathscr{B})$. We argue that we can find a set of at most w vertices $S \subseteq \overline{V_t}$ such that S separates V_t from $\overline{V_t} \setminus S$ which by Lemma 6 will yield the claim.

Let $S \coloneqq N(V_t) \cap \overline{V_t}$ and consider a vertex $x \in S$. By definition, x is a neighbor of some vertex y that has been forgotten below t. By (ii) of the definition of a tree decomposition there is a node, say t', such that $B_{t'}$ contains both x and y. Since y has been forgotten below t, t' is below t.

As $x \in \overline{V_t}$, there is also a node above t, say t'', such that $B_{t''}$ contains x (e.g. the bag below the one that forgets x). Since t lies on the path between t' and t'' in T, we have by property (iii) of the definition of a tree decomposition that $x \in B_t$. We have that $S \subseteq B_t$.

Furthermore, S separates V_t from $\overline{V_t} \setminus S$. If t is a forget node, then by definition $|B_t| \leq w$ and hence $|S| \leq w$. If t is a join node, then by assumption $|B_t| \leq w$ and hence $|S| \leq w$. If t is an introduce node introducing a vertex $u \in V(G)$, then u cannot have any neighbor in V_t, since all vertices in V_t have been forgotten below t. Hence, $S \subseteq B_t \setminus \{u\}$ and we can conclude that $|S| \leq w$. □

It is well-known (see e.g. [21]) that any tree decomposition can be transformed in polynomial time to a nice tree decomposition of the same width. As in a tree decomposition of width $w-1$, all bags (in particular bags at join nodes) have size at most w, the previous theorem implies the following. Note that the bound for pathwidth is slightly tighter than the one for tree-width, as in a path decomposition there are no join nodes.

Corollary 9. *Let r and w be positive integers and let G be a graph of tree-width $w-1$ (path-width w). Then the r-th power of G has mim-width at most w and given a tree decomposition (path decomposition) of G of width $w-1$ (w), one can compute a decomposition tree of mim-width w in polynomial time.*

The following notions are of importance in the field of phylogenetic studies, i.e. the reconstruction of ancestral relations in biology, see e.g. [8]. A graph G is a *leaf power* if there exists a threshold r and a tree T, called a leaf root, whose leaf set is $V(G)$ such that $uv \in E$ if and only if the distance between u and v in T is at most r. Similarly, G is called a *min-leaf power* if $uv \in E$ if and only if the distance between u and v in T is more than r. Thus, G is a leaf power if and only if its complement is a min-leaf power. It is easy to see that trees admit nice tree decompositions all of whose join bags have size 1 and since every leaf power graph is an induced subgraph of a power of some tree, it has mim-width at most 1 by Theorem 8. Together with [31, Lemma 3.7.3], stating that the mim-width of a graph is 1 if and only if the mim-width of its complement is 1, we have the following result.

Corollary 10. *The leaf powers and min-leaf powers have mim-width at most* 1 *and given a leaf root, we can compute in polynomial time a decomposition tree witnessing this.*

Next, we consider powers of graphs of bounded clique-width. A graph is *w-labeled* if there is a labeling function $f : V(G) \to [w]$, and we call $f(v)$ the *label* of v. For a w-labeled graph G, we call the set of all vertices with label i the *label class i* of G. The following can be thought as a generalization of Lemma 6.

Lemma 11. *Let r and w be positive integers and let (A,B) be a vertex partition of graph G such that $G[A]$ is w-labeled and for every pair of vertices x,y in the same label class of $G[A]$, x and y have the same neighborhood in B. Let $H := G^r$. Then,* $\mathrm{mim}_H(A) \leq w$.

Proof. Suppose for contradiction that $\mathrm{mim}_H(A) > w$. Let $\{y_1z_1, y_2z_2, \ldots, y_tz_t\}$ be an induced matching of size at least $w+1$ in $H[A,B]$. For $i \in \{1,2,\ldots,t\}$, there is a path P_i of length at most r from y_i to z_i in G. Let $A^* \subseteq A$ be the set of vertices in A that have a neighbor in B. Let Q_i be the subpath of P_i from the last vertex in A^* to z_i, and q_i be the endpoint of Q_i different from z_i, and let R_i be the subpath of P_i from y_i to q_i. Let a_i be the length of R_i and b_i be the length of Q_i. By construction, $a_i + b_i \leq r$.

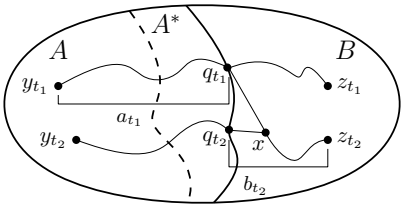


Figure 2: Illustration of the situation in the proof of Lemma 11.

(For an illustration of the following argument, see Figure 2.) Since $t \geq w+1$, there are two integers $t_1, t_2 \in \{1, 2, \ldots, t\}$ such that q_{t_1} and q_{t_2} are contained in the same label class of $G[A]$. Since $a_{t_1} + b_{t_1} \leq r$ and $a_{t_2} + b_{t_2} \leq r$, we either have that $a_{t_1} + b_{t_2} \leq r$ or $a_{t_2} + b_{t_1} \leq r$.

Assume $a_{t_1} + b_{t_2} \leq r$. In this case, we show that the distance from y_{t_1} to z_{t_2} in G is at most r, which contradicts the assumption that $y_{t_1} z_{t_2}$ is not an edge of H. Note that two vertices in a label class of $G[A]$ have the same neighborhood in B. Let x denote the vertex on Q_{t_2} that is adjacent to q_{t_2}. Then, as q_{t_1} and q_{t_2} are in the same label class of $G[A]$, we have that q_{t_1} is also adjacent to x. Therefore, $G[V(R_{t_1}) \cup (V(Q_{t_2}) \setminus \{q_{t_2}\})]$ contains a path of length at most r from y_{t_1} to z_{t_2}. Analogously we can show that if $a_{t_2} + b_{t_1} \leq r$, then $G[V(R_{t_2}) \cup (V(Q_{t_1}) \setminus \{q_{t_1}\})]$ contains a path of length at most r from y_{t_2} to z_{t_1}. But this is a contradiction and we conclude that $\mathrm{mim}_H(A) \leq w$. □

Definition 12. The *clique-width* of a graph G is the minimum number of labels needed to construct G using the following four operations:

(1) Creation of a new vertex v with label i (denoted by $i(v)$).

(2) Disjoint union of two labeled graphs G and H (denoted by $G \oplus H$).

(3) Joining by an edge each vertex with label i to each vertex with label j ($i \neq j$, denoted by $\eta_{i,j}$).

(4) Renaming label i to j (denoted by $\rho_{i \to j}$).

A string of operations given in the previous definition is called a *clique-width w-expression* or shortly a *w-expression* if it uses at most w distinct labels. We can represent this expression as a tree-structure and such trees are known as *syntactic trees* associated with w-expressions. An easy observation is that for a node t in a syntactic tree associated with a w-expression, and the vertex set V_t consisting of vertices introduced in some descendants of t, $G[V_t]$ is a w-labeled graph where two vertices in the same label class has the same neighborhood in $V(G) \setminus V_t$.

Theorem 13. *Let r and w be positive integers and G be a graph of clique-width w. Then the r-th power of G has mim-width at most w. Furthermore, given a clique-width w-expression, we can output a decomposition tree of mim-width at most w in polynomial time.*

Proof. Let H be the r-th power of G and let ϕ be the given clique-width w-expression defining G, and T be the syntactic tree of ϕ with root node $\mathfrak{r}$. We can assume that G contains at least two vertices which implies that T has at least one join node. Let $\mathfrak{r}'$ be the join node in T with minimum $\mathrm{DIST}_T(\mathfrak{r}, \mathfrak{r}')$. Note that if $\mathfrak{r}$ itself is a join node, then $\mathfrak{r}' = \mathfrak{r}$. Note also that for every

vertex v of G, there is a node of T creating v, see the first operation in the definition of clique-width. In the following, we call such a node an *introduce node*. We obtain a decomposition tree $(T', \mathscr{L})$ as follows:

- We obtain T' from T as follows: If $\mathfrak{r} \neq \mathfrak{r}'$, we first remove all vertices in the path from $\mathfrak{r}'$ to $\mathfrak{r}$ in T other than $\mathfrak{r}'$. We fix $\mathfrak{r}'$ to be the root node of T'.
- For each introduce node ℓ_v introducing a vertex v, we assign $\mathscr{L}(v) := \ell_v$.

For the first step, note that if the root $\mathfrak{r}$ of T is not a join node, then it must have degree one in T. This implies that T' is connected, and that each introduce node is a descendant of $\mathfrak{r}'$. Consider a cut $(V_t, \overline{V_t})$ for some $t \in V(T')$, where V_t is the set of vertices that are introduced below t in T, and note that by construction this is also the set of vertices of G that are mapped to leaves in the subtree of T' rooted at t. If t is a leaf node, then $\mathrm{mim}_H(V_t) \leq 1$. Assume t is an internal node. Then t also appears in T.

We observe that $G[V_t]$ is a w-labeled graph such that for any pair of vertices x, y in the same label class, x and y have the same neighborhood in $V(G) \setminus V_t$. So we can apply Lemma 11 to conclude that we have $\mathrm{mim}_H(V_t) \leq w$ which implies that H has mim-width at most w. □

4 Algorithmic Consequences

Using the results from the previous section, we now give algorithmic consequences for distance-r versions of (σ, ρ) problems and more generally of LCVP problems (introduced below). In particular, the results in this section follow from Theorem 5 which states that taking an arbitrary power of a graph never increases its mim-width by more than a factor of two. The second ingredient is the following simple observation about neighborhoods of r-th powers of graphs.

Observation 14. For a positive integer r, a graph G and a vertex $u \in V(G)$, the r-neighborhood of u is equal to the neighborhood of u in G^r, i.e. $N_G^r(u) = N_{G^r}(u)$.

The observation above shows that solving a distance-r (σ, ρ) problem on G is the same as solving the same standard distance-1 variation of the problem on G^r. Hence, we may reduce our problem to the standard version by simply computing the graph power. Combining Theorem 5 with the algorithms provided in Proposition 3, we have the following consequence.

Corollary 15. *There is an algorithm that for all $r \in \mathbb{N}$, given a graph G and a decomposition tree $(T, \mathscr{L})$ of G with $w := mimw_G(T, \mathscr{L})$ solves each distance-r (σ, ρ) problem Π with $d := d(\Pi)$*

(i) in time $\mathscr{O}(n^{4+4d \cdot w})$, if T is a caterpillar, and

(ii) in time $\mathscr{O}(n^{4+6d \cdot w})$, otherwise.

Proof. Let G be the input graph and $(T, \mathscr{L})$ the provided decomposition tree. We apply the following algorithm:

Step 1. Compute the graph G^r.

Step 2. Solve the standard (distance-1) version of the problem on G^r, providing $(T, \mathscr{L})$ as the decomposition tree.

Step 3. Return the answer of the algorithm ran in Step 2 without modification.

Computing G^r in Step 1 takes at most $\mathcal{O}(n^3)$ time using standard methods, Step 3 takes constant time. The worst time complexity is hence found in Step 2. By Theorem 5, the mim-width of $(T,\mathscr{L})$ on G^r is at most twice that of the same decomposition on G. The stated runtime then follows from Proposition 3. The correctness of this procedure follows immediately from Observation 14. □

LCVP Problems. A generalization of (σ,ρ) problems are the *locally checkable vertex partitioning* (LCVP) problems which we now discuss. A *degree constraint matrix* D is a $q \times q$ matrix where each entry is a finite or co-finite subset of $\mathbb{N}$. For a graph G and a partition of its vertices $\mathscr{V} = \{V_1, V_2, \dots V_q\}$ (where some parts of the partition may possibly be empty), we say that it is a D-partition if and only if, for each $i, j \in [q]$ and each vertex $v \in V_i$, it holds that $|N(v) \cap V_j| \in D[i,j]$.

For instance, let D_3 be the 3×3 matrix whose diagonal entries are $\{0\}$ and the non-diagonal ones are $\mathbb{N}$, i. e.

$$D_3 := \begin{bmatrix} \{0\} & \mathbb{N} & \mathbb{N} \\ \mathbb{N} & \{0\} & \mathbb{N} \\ \mathbb{N} & \mathbb{N} & \{0\} \end{bmatrix}.$$

Then, a graph G admits a 3-coloring if and only if it admits a D_3-partition.

Typically, the natural algorithmic questions associated with LCVP properties are existential.[3] Interesting problems which can be phrased in such terms include the H-COVERING and GRAPH H-HOMOMORPHISM problems where H is fixed, as well as q-COLORING, PERFECT MATCHING CUT and more. We refer to [30] for an overview.

We generalize LCVP properties to their distance-r version, by considering the ball of radius r around each vertex rather than just the immediate neighborhood.

Definition 16 (Distance-r neighborhood constraint matrix)**.** A distance-r neighborhood constraint matrix D is a $q \times q$ matrix where each entry is a finite or co-finite subset of $\mathbb{N}$. For a graph G and a partition of its vertices $\mathscr{V} = \{V_1, V_2, \dots V_q\}$ (where some parts of this partition may be empty), we say that it is a D-distance-r-partition if and only if, for each $i, j \in [q]$ and each vertex $v \in V_i$, it holds that $|N^r(v) \cap V_j| \in D[i,j]$.

We say that an algorithmic problem is a *distance-r* LCVP *problem* if the property in question can be described by a distance-r neighborhood constraint matrix. For example, the distance-r version of a problem such as q-COLORING can be interpreted as an assignment of at most q colours to vertices of a graph such that no two vertices are assigned the same colour if they are at distance r or closer.

For a given distance-r LCVP problem Π, its d-value $d(\Pi)$ is the maximum d-value over all the sets in the corresponding neighborhood constraint matrix.

As in the case of (σ,ρ) problems, combining Theorem 5 with Observation 14 and the works [3, 7] we have the following result.

Corollary 17. *There is an algorithm that for all $r \in \mathbb{N}$, given a graph G and a decomposition tree $(T,\mathscr{L})$ of G with $w := mimw_G(T,\mathscr{L})$ solves each distance-r LCVP problem Π with $d := d(\Pi)$*

(i) in time $\mathcal{O}(n^{4+4qd\cdot w})$, if T is a caterpillar, and

[3] Note however that each (σ,ρ) problem can be stated as an LCVP problem via the matrix $D_{(\sigma,\rho)} = \begin{bmatrix} \sigma & \mathbb{N} \\ \rho & \mathbb{N} \end{bmatrix}$, so maximization or minimization of some block of the partition can be natural as well.

(ii) in time $\mathcal{O}(n^{4+6qd\cdot w})$, otherwise.

As we state the runtime bounds in the previous corollary in a different way than in [7] where they have been proved first, we would like to note that this is justified by an argument parallel to the one we provided in the sketch of the proof of Proposition 3 presented in this work.

5 Lower Bounds

We show that several (σ,ρ) problems are W[1]-hard parameterized by linear mim-width plus solution size. Our reductions are based on two recent reductions due to Fomin, Golovach and Raymond [13] who showed that INDEPENDENT SET and DOMINATING SET are W[1]-hard parameterized by linear mim-width plus solution size. In fact they show hardness for the above mentioned problems on *H-graphs* (the parameter being the number of edges in H plus solution size) which we now define formally.

Definition 18 (H-Graph)**.** Let X be a set and $\mathcal{S}$ a family of subsets of X. The *intersection graph* of $\mathcal{S}$ is a graph with vertex set $\mathcal{S}$ such that $S,T \in \mathcal{S}$ are adjacent if and only if $S \cap T \neq \emptyset$. Let H be a (multi-) graph. We say that G is an *H-graph* if there is a subdivision H' of H and a family of subsets $\mathcal{M} := \{M_v\}_{v\in V(G)}$ (called an *H-representation*) of $V(H')$ where $H'[M_v]$ is connected for all $v \in V(G)$, such that G is isomorphic to the intersection graph of $\mathcal{M}$. For a vertex $v \in V(G)$, we call M_v the *model* of v.

We make an immediate observation from the definition of H-graphs that will be useful in later proofs of this section. For a graph H, we can construct a H-representation of itself: We subdivide each edge of H once to obtain H'. Then, we create an H-representation $\mathcal{M}$ of H by adding for each vertex $v \in V(H)$ a model $M_v := N_{H'}[v]$.

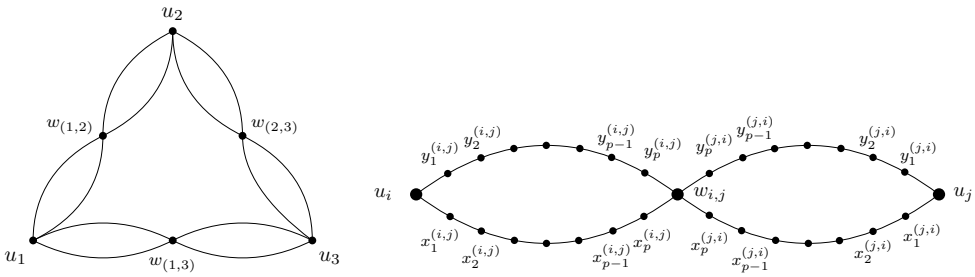
Observation 19. Any graph H is an H-graph.

All of the hardness results presented in this section are obtained via reductions to the respective problems on H-graphs, and the hardness for linear mim-width follows from the following proposition.

Proposition 20 (Theorem 2 in [13])**.** *Let G be an H-graph. Then, G has linear mim-width at most $2\cdot||H||$ and a corresponding decomposition tree can be computed in polynomial time given an H-representation of G.*

5.1 Maximization Problems

The first lower bound concerns several maximization problems that can be expressed in the (σ,ρ) framework. Recall that the (MAXIMUM) INDEPENDENT SET problem can be formulated as MAX-$(\{0\},\mathbb{N})$. The following result states that a large class of (σ,ρ) maximization problems that are related to the INDEPENDENT SET problem according to their (σ,ρ) formulation are W[1]-hard on H-graphs parameterized by $||H||$ plus solution size. These problems include INDUCED MATCHING, DOMINATING INDUCED MATCHING, INDUCED d-REGULAR SUBGRAPH, and INDUCED SUBGRAPH OF MAXIMUM DEGREE d, see Table 1 for the details.



(a) Example of the graph H when $k = 3$.

(b) The subdivisions done for a pair (i, j).

Figure 3: Illustration of the construction of Fomin et al. [13].

Theorem 21. *For any fixed $d \in \mathbb{N}$ and $x \leq d+1$, the following holds. Let $\sigma^* \subseteq \mathbb{N}_{\leq d}$ with $d \in \sigma^*$. Then,* MAX-$(\sigma^*, \mathbb{N}_{\geq x})$ DOMINATION *is* W[1]*-hard on H-graphs parameterized by the number of edges in H plus solution size, and the hardness holds even if an H-representation of the input graph is given.*

Proof. To prove the theorem, we provide a reduction from MULTICOLORED CLIQUE where given a graph G and a partition $V_1, \ldots, V_k$ of $V(G)$, the question is whether G contains a clique of size k using precisely one vertex from each V_i ($i \in [k]$). This problem is known to be W[1]-complete [11, 25].

Let $(G, V_1, \ldots, V_k)$ be an instance of MULTICOLORED CLIQUE. We can assume that $k \geq 3$ and that $|V_i| = p$ for $i \in [k]$. If the second assumption does not hold, let $p := \max_{i \in [k]} |V_i|$ and add $p - |V_i|$ isolated vertices to V_i, for each $i \in [k]$. (Note that adding isolated vertices does not change the answer to the problem.) For $i \in [k]$, we denote by $v_1^i, \ldots, v_p^i$ the vertices of V_i. We first describe the reduction of Fomin et al. [13] and then explain how to modify it to prove the theorem.

The Construction of Fomin, Golovach and Raymond [13]. The graph H is obtained as follows, see Figure 3a.

1. Construct k nodes $u_1, \ldots, u_k$.
2. For every $1 \leq i < j \leq k$, construct a node $w_{i,j}$ and two pairs of parallel edges $u_i w_{i,j}$ and $u_j w_{i,j}$.

We then construct the subdivision H' of H by first subdividing each edge p times. We denote the subdivision nodes for 4 edges of H constructed for each pair $1 \leq i < j \leq k$ in Step 2 by $x_1^{(i,j)}, \ldots, x_p^{(i,j)}$, $y_1^{(i,j)}, \ldots, y_p^{(i,j)}$, $x_1^{(j,i)}, \ldots, x_p^{(j,i)}$, and $y_1^{(j,i)}, \ldots, y_p^{(j,i)}$. This subdivision process is depicted in Figure 3b. To simplify notation, we assume that $u_i = x_0^{(i,j)} = y_0^{(i,j)}$, $u_j = x_0^{(j,i)} = y_0^{(j,i)}$ and $w_{i,j} = x_{p+1}^{(i,j)} = y_{p+1}^{(i,j)} = x_{p+1}^{(j,i)} = y_{p+1}^{(j,i)}$.

We now construct the H-graph G'' by defining its H-representation $\mathscr{M} = \{M_v\}_{v \in V(G'')}$ where each M_v is a connected subset of $V(H')$. (Recall that G denotes the graph of the MULTICOLORED CLIQUE instance.)

1. For each $i \in [k]$ and $s \in [p]$, construct a vertex z_s^i with model

$$M_{z_s^i} := \bigcup_{j \in [k], j \neq i} \left(\left\{ x_0^{(i,j)}, \ldots, x_{s-1}^{(i,j)} \right\} \cup \left\{ y_0^{(i,j)}, \ldots, y_{p-s}^{(i,j)} \right\} \right).$$

2. For each edge $v_s^i v_t^j \in E(G)$ for $s,t \in [p]$ and $1 \leq i < j \leq k$, construct a vertex $r_{s,t}^{(i,j)}$ with:

$$M_{r_{s,t}^{(i,j)}} := \left\{x_s^{(i,j)}, \ldots, x_{p+1}^{(i,j)}\right\} \cup \left\{y_{p-s+1}^{(i,j)}, \ldots, y_{p+1}^{(i,j)}\right\} \cup \left\{x_t^{(j,i)}, \ldots, x_{p+1}^{(j,i)}\right\} \cup \left\{y_{p-t+1}^{(j,i)}, \ldots, y_{p+1}^{(j,i)}\right\}.$$

Throughout the following, for $i \in [k]$ and $1 \leq i < j \leq k$, we use the notation

$$Z(i) := \bigcup\nolimits_{s\in[p]} \left\{z_s^i\right\} \text{ and } R(i,j) := \bigcup_{\substack{s,t\in[p],\\ v_s^i v_t^j \in E(G)}} \left\{r_{s,t}^{(i,j)}\right\},$$

respectively. We now observe the crucial property of G''.

Observation 21.1 (Claim 18 in [13]). For every $1 \leq i < j \leq k$, a vertex $z_h^i \in V(G')$ (a vertex $z_h^j \in V(G')$) is *not* adjacent to a vertex $r_{s,t}^{(i,j)} \in V(G')$ corresponding to the edge $v_s^i v_t^j \in E(G)$ if and only if $h = s$ ($h = t$).

We now describe how to obtain from G'' a graph G' that will be the graph of the instance of MAX-$(\sigma^*, \mathbb{N}_{\geq x})$ DOMINATION, by adding a gadget attached to each set $Z(i)$ and $R(i,j)$ (for all $1 \leq i < j \leq k$).

The New Gadget and the Construction of G'. Let X be a set of vertices of a graph. The gadget $\mathfrak{B}(X)$ is a complete bipartite graph on $2d-1$ vertices and bipartition $(\{\beta_{1,1}, \ldots, \beta_{1,d}\}, \{\beta_{2,1}, \ldots, \beta_{2,d-1}\})$ such that for $h \in [d]$, each vertex $\beta_{1,h}$ is additionally adjacent to each vertex in X.

The graph G' is obtained from G'' by adding the gadgets $\mathfrak{B}(Z(i))$ for all $i \in [k]$ and the gadgets $\mathfrak{B}(R(i,j))$ for all $1 \leq i < j \leq k$. To prove the theorem, we require G' to be a K-graph for some graph K whose number of edges is bounded by a function of k, and possibly d, as d is fixed. We will show that G' is a K-graph for some supergraph K of H that meets this requirement.

Motivated by Observation 19, and the fact that the bipartite graph in each gadget $\mathfrak{B}(\cdot)$ has $\mathcal{O}(d^2)$ edges, we do the following to obtain K from H: For each $i \in [k]$, we add the gadget $\mathfrak{B}(\{u_i\})$, which will be used to encode $\mathfrak{B}(Z(i))$ in G'; furthermore, for each $1 \leq i < j \leq k$, we add the gadget $\mathfrak{B}(\{w_{(i,j)}\})$, which will be used to encode $\mathfrak{B}(R(i,j))$ in G'. For an illustration of K, see Figure 4. We obtain a subdivision K' of K as follows:

(K1) For all edges in $E(K) \cap E(H)$, we do the same subdivisions that were made to obtain H' from H.

(K2) For each gadget $\mathfrak{B}(\cdot)$, we perform the edge subdivisions due to Observation 19 that allow for encoding the bipartite graph in $\mathfrak{B}(\cdot)$ as a $\mathfrak{B}(\cdot)$-graph.

(K3) For each $i \in [k]$, let $\{\beta_{1,1}^i, \ldots, \beta_{1,d}^i, \beta_{2,1}^i, \ldots, \beta_{2,d-1}^i\}$ be the vertices of $\mathfrak{B}(\{u_i\})$. Then, for each $h \in [d]$, we subdivide the edge $u_i \beta_{1,h}^i$, and denote the corresponding subdivision node by $s(i,h)$.

(K4) Similarly, for each $1 \leq i < j \leq k$, let $\{\beta_{1,1}^{(i,j)}, \ldots, \beta_{1,d}^{(i,j)}, \beta_{2,1}^{(i,j)}, \ldots, \beta_{2,d-1}^{(i,j)}\}$ be the vertices of $\mathfrak{B}(\{w_{(i,j)}\})$. Then, for each $h \in [d]$, we subdivide the edge $u_i \beta_{1,h}^{(i,j)}$, and denote the corresponding subdivision node by $s((i,j),h)$.

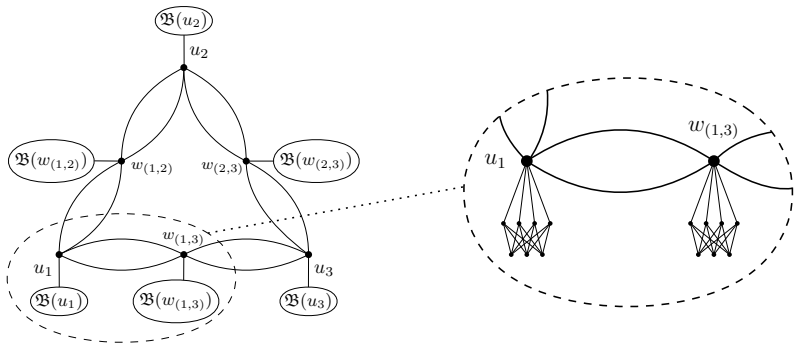


Figure 4: The graph K with respect to which the graph G' constructed in the proof of Theorem 21 is a K-graph. In this example, we have $k = 3$ and $d = 4$.

We now give the K-representation of G', $\mathscr{M}' = \{M'_v\}_{v \in V(G')}$, where each M'_v is a connected subset of $V(K')$; for an illustration of G' see Figure 5.

(R1) For each vertex $v \in V(G') \cap V(G'')$, we let $M'_v := M_v$, where M_v is the model of v defined given in the construction of Fomin et al. which we described above. Note that each such vertex is either some vertex z^i_s or some vertex $r^{(i,j)}_{s,t}$ for appropriate choices for i, j, s, and t.

(R2) For each $i \in [k]$, let $\{\beta^i_{1,1}, \ldots, \beta^i_{1,d}, \beta^i_{2,1}, \ldots, \beta^i_{2,d-1}\}$ be the vertices of $\mathfrak{B}(\{u_i\})$. By the subdivisions we did in Step K2, we obtain a corresponding complete bipartite graph on vertices $\{b^i_{1,1}, \ldots, b^i_{1,d}, b^i_{2,1}, \ldots, b^i_{2,d-1}\}$. Each $b^i_{t,h}$, where $t \in [2]$ and $h \in [d]$ if $t = 1$ and $h \in [d-1]$ if $t = 2$, comes with a model from the subdivision of the complete bipartite graph of $\mathfrak{B}(\{u_i\})$, which we initially use as $M'_{b^i_{t,h}}$.

(R3) We proceed analogously to Step R2 with each $\mathfrak{B}(\{w_{(i,j)}\})$, where $1 \leq i < j \leq k$.

(R4) For each $i \in [k]$, and each $h \in [d]$, we add the node $s(i,h)$ to the models of z^i_s for all $s \in [p]$, and we add $s(i,h)$ to the model (in $\mathscr{M}'$) of $b^i_{1,h}$. (This ensures that each $b^i_{1,h}$ is complete to $Z(i)$.)

(R5) For each $1 \leq i < j \leq k$ and $s,t \in [p]$ such that $v^i_s v^j_t \in E(G)$, and each $h \in [d]$, we add the node $s((i,j),h)$ to the model of $r^{(i,j)}_{s,t}$, and we add $s((i,j),h)$ to the model (in $\mathscr{M}'$) of $b^{(i,j)}_{1,h}$. (This ensures that each $b^{(i,j)}_{1,h}$ is complete to $R(i,j)$.)

We count the size of K. For $|K|$, we observe that $|H| = k + \binom{k}{2}$ and each gadget $\mathfrak{B}(\cdot)$ has $2d-1$ nodes, and we add $k + \binom{k}{2}$ such gadgets. Hence, $|K| = 2d\left(k + \binom{k}{2}\right) = dk(k+1)$. As for $||K||$, we observe that the number of edges in H is $4 \cdot \binom{k}{2}$ and each gadget $\mathfrak{B}(\cdot)$ introduces $d(d-1) + d = d^2$ edges. Hence,

$$||K|| = 4 \cdot \binom{k}{2} + d^2\left(k + \binom{k}{2}\right) = \mathscr{O}(d^2 \cdot k^2). \tag{2}$$

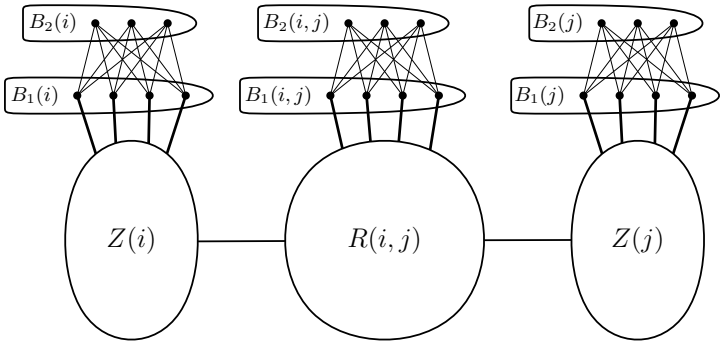


Figure 5: A part of the graph G', where $1 \leq i < j \leq k$ and $d = 4$.

We introduce some more notation. For $1 \leq i < j \leq k$, we let $B_1(i) := \{b^i_{1,1}, \ldots, b^i_{1,d}\}$, $B_2(i) := \{b^i_{2,1}, \ldots, b^i_{2,d-1}\}$, $B_1(i,j) := \{b^{(i,j)}_{1,1}, \ldots, b^{(i,j)}_{1,d}\}$ and $B_2(i,j) := \{b^{(i,j)}_{2,1}, \ldots, b^{(i,j)}_{2,d-1}\}$; furthermore $B(i) := B_1(i) \cup B_2(i)$, $B(i,j) := B_1(i,j) \cup B_2(i,j)$, and $B := \bigcup_{i \in [k]} B(i) \cup \bigcup_{1 \leq i < j \leq k} B(i,j)$. Note that $|B| = (2d-1)(k + \binom{k}{2})$. We furthermore use the notation

$$Z_{+B}(i) := Z(i) \cup B(i) \text{ and } R_{+B}(i,j) := R(i,j) \cup B(i,j).$$

We now turn to the correctness proof of the reduction. We let $k' := 2d \cdot (k + \binom{k}{2})$ and show that G has a multicolored clique if and only if G' has a $(\sigma^*, \mathbb{N}_{\geq x})$ set of size k'. We first prove the forward direction. Note that the following claim yields the forward direction of the correctness proof, since a $(\{d\}, \{d+1, \ldots, d+k\})$ set is a $(\sigma^*, \mathbb{N}_{\geq x})$ set. (Recall that $d \in \rho^*$ and $x \leq d+1$.)

Claim 21.2. If G has a multicolored clique, then G' has a $(\{d\}, \{d+1, \ldots, d+k\})$ set of size $k' = 2d \cdot (k + \binom{k}{2})$ (assuming $k \geq 3$).

Proof. Let $\{v^1_{h_1}, \ldots, v^k_{h_k}\}$ be the vertex set in G that induces the multicolored clique. By Observation 21.1 we can verify that

$$I := \left\{z^1_{h_1}, \ldots, z^k_{h_k}\right\} \cup \left\{r^{(i,j)}_{h_i,h_j} \mid 1 \leq i < j \leq k\right\} \tag{3}$$

is an independent set in G'. We let $S := I \cup B$ and observe that S is a $(\{d\}, \{d+1, \ldots, d+k\})$ set: By construction, there is no edge between any pair of distinct sets of $B(i)$, $B(i')$, $B(i,j)$, $B(i',j')$, for any choice of $1 \leq i < j \leq k$ and $1 \leq i' < j' \leq k$.

Consider any vertex $x \in S$ and suppose that $x \in Z_{+B}(i)$ for some $i \in [k]$. (The case when $x \in R_{+B}(i,j)$ can be argued for analogously.) If $x = z^i_{h_i}$, then x is adjacent to the d vertices $b^i_{1,1}, \ldots, b^i_{1,d}$, if $x = b^i_{1,\ell}$ for some $\ell \in [d]$, then x is adjacent to $z^i_{h_i}$ and the vertices $b^i_{2,1}, \ldots, b^i_{2,d-1}$ and if $x = b^i_{2,\ell'}$ for some $\ell' \in [d-1]$, then it is adjacent to the vertices $b^i_{1,1}, \ldots, b^i_{1,d}$. Hence, in all cases, x has precisely d neighbors in S.

Let $y \in V(G') \setminus S$ and note that $(V(G') \setminus S) \cap B = \emptyset$. If $y \in Z(i)$ for some $i \in [k]$, then $N(y) \cap S \supseteq \{z^i_{h_i}, b^i_{1,1}, \ldots, b^i_{1,d}\}$, so $|N(y) \cap S| \geq d+1$. Since the only additional neighbors of y in S are in the set $R_i := \bigcup_{1 \leq j < i} R(j,i) \cup \bigcup_{i < j \leq k} R(i,j)$ and $R_i \cap S \subseteq I$, we can conclude that $|N(y) \cap (S \setminus B)| \leq k-1$, since I contains precisely one vertex from each set $R(i,j)$. We have argued that $d+1 \leq |N(y) \cap S| \leq d+k$. If $y \in R(i,j)$ for some $1 \leq i < j \leq k$, we can argue as

before that $|N(y) \cap S| \geq d+1$ and since all neighbors of y in $S \setminus B(i,j)$ are contained either in $Z(i)$ or $Z(j)$, we can conclude that $d+1 \leq |N(y) \cap S| \leq d+3 \leq d+k$.

It remains to count the size of S. Clearly, $|I| = k + \binom{k}{2}$ and as observed above, $|B| = (2d-1)(k+\binom{k}{2})$, so

$$|S| = |I| + |B| = k + \binom{k}{2} + (2d-1)\left(k + \binom{k}{2}\right) = 2d\left(k + \binom{k}{2}\right) = k',$$

as claimed. ⌟

We now prove the backward direction of the correctness of the reduction. We begin by making several observations about the structure of $(\sigma^*, \mathbb{N}_{\geq x})$ sets in the graph G'.

Claim 21.3. Let $1 \leq i < j \leq k$.

(i) Any $(\sigma^*, \mathbb{N}_{\geq x})$ set in G' contains at most $d+1$ vertices from each $Z(i) \cup B_1(i)$ or $R(i,j) \cup B_1(i,j)$.

(ii) Any $(\sigma^*, \mathbb{N}_{\geq x})$ set contains at most $2d$ vertices from each $Z_{+B}(i)$ or $R_{+B}(i,j)$.

(iii) If a $(\sigma^*, \mathbb{N}_{\geq x})$ set S contains $2d$ vertices from some $Z_{+B}(i)$ ($R_{+B}(i,j)$), then it contains at least one vertex from $Z(i)$ ($R(i,j)$) and each such vertex in $S \cap Z(i)$ ($S \cap R(i,j)$) has at least d neighbors in $S \cap Z_{+B}(i)$ ($S \cap R_{+B}(i,j)$).

Proof. (i) We prove the claim for a set $Z(i) \cup B_1(i)$. The proof for a set $R(i,j) \cup B_1(i,j)$ works analogously. Suppose for the sake of a contradiction that there is a set $S \subseteq V(G')$ that contains at least $d+2$ vertices from some $Z(i) \cup B_1(i)$. Since $|B_1(i)| = d$, we know that S contains a vertex from $Z(i)$, say x. However, by construction, all vertices in $S \cap (Z(i) \cup B_1(i)) \setminus \{x\}$ are adjacent to x, implying that x has at least $d+1$ neighbors in S, a contradiction with the fact that S is a $(\sigma^*, \mathbb{N}_{\geq x})$ set.

(ii) follows as a direct consequence, since $Z_{+B}(i) \setminus (Z(i) \cup B_1(i)) = B_2(i)$ and $|B_2(i)| = d-1$. Similar for $R_{+B}(i,j)$.

(iii). The claim that S contains at least one vertex from $Z(i)$ is immediate since $|S \cap Z_{+B}(i)| = 2d$ and $|Z_{+B}(i) \setminus Z(i)| = |B(i)| = 2d-1$. Let $x \in S \cap Z(i)$ be such a vertex. Then, the only vertices of $Z_{+B}(i)$ that x is *not* adjacent to are the vertices $B_2(i)$. Since $|B_2(i)| = d-1$, the remaining vertices in $(S \cap Z_{+B}(i)) \setminus (B_2(i) \cup \{x\})$, of which there are at least d as we just argued, are neighbors of x. Similar for $R_{+B}(i,j)$. ⌟

Equipped with the previous claim, we can now finish the correctness proof of the reduction.

Claim 21.4. If G' contains a $(\sigma^*, \mathbb{N}_{\geq x})$ set S of size $k' = 2d(k+\binom{k}{2})$, then G contains a multi-colored clique.

Proof. Let S be a $(\sigma^*, \mathbb{N}_{\geq x})$ set of size k' in G'. By Claim 21.3(ii), we can conclude that S contains precisely $2d$ vertices from each $Z_{+B}(i)$ and each $R_{+B}(i,j)$ (where $1 \leq i < j \leq k$). Consider any pair i, j with $1 \leq i < j \leq k$. By Claim 21.3(iii) we know that there are vertices

$$z^i_{s_i} \in Z(i) \cap S, \;\; z^j_{s_j} \in Z(j) \cap S, \;\; \text{and } r^{(i,j)}_{t_i,t_j} \in R(i,j) \cap S,$$

for some $s_i, s_j, t_i, t_j \in [p]$. Again by Claim 21.3(iii), $z^i_{s_i}$ has d neighbors in $Z_{+B}(i) \cap S$, so if $z^i_{s_i} r^{(i,j)}_{t_i,t_j} \in E(G')$, then $z^i_{s_i}$ has $d+1$ neighbors in S, a contradiction with the fact that S is a

$(\sigma^*, \mathbb{N}_{\geq x})$ set. Hence, $z^i_{s_i} r^{(i,j)}_{t_i,t_j} \notin E(G')$ and $z^j_{s_j} r^{(i,j)}_{t_i,t_j} \notin E(G')$. By Observation 21.1, we then have that $s_i = t_i$ and $s_j = t_j$. We can conclude that $v^i_{s_i} v^j_{s_j} \in E(G)$ and since the argument holds for any pair of indices i, j, G has a multicolored clique. ⌟

We would like to remark that by the proof of the previous claim, we have established that any $(\sigma^*, \mathbb{N}_{\geq x})$ set S in G' of size k' in fact contains all vertices from B and one vertex from each $Z(i)$ and from each $R(i,j)$. Since this is precisely the shape of the set constructed in the forward direction of the correctness proof, this shows that any $(\sigma^*, \mathbb{N}_{\geq x})$ set of size k' in G' is a $(\{d\}, \{d+1, \ldots, d+k\})$ set (assuming $k \geq 3$).

Claims 21.2 and 21.4 establish the correctness of the reduction. We observe that $|V(G')| = \mathcal{O}(|V(G)| + d^2 \cdot k^2)$ and clearly, G' can be constructed from G in time polynomial in $|V(G)|$, d and k as well. Furthermore, by (2), $||K|| = \mathcal{O}(d^2 \cdot k^2)$ which implies that $||K|| = \mathcal{O}(k^2)$ since d is a fixed constant and the theorem follows. □

By Proposition 20, the previous theorem has the following consequence.

Corollary 22. *For any fixed $d \in \mathbb{N}$ and $x \leq d+1$, the following holds. Let $\sigma^* \subseteq \mathbb{N}_{\leq d}$ with $d \in \sigma^*$. Then,* MAX-$(\sigma^*, \mathbb{N}_{\geq x})$ DOMINATION *is* W[1]*-hard parameterized by linear mim-width plus solution size, and the hardness holds even if a corresponding decomposition tree is given.*

5.2 Minimization Problems

In this section we prove W[1]-hardness of minimization versions of several (σ, ρ) problems parameterized by linear mim-width plus solution size. We obtain our results by modifying a reduction from MULTICOLORED INDEPENDENT SET to MINIMUM DOMINATING SET on H-graphs parameterized by solution size plus $||H||$ due to Fomin et al. [13]. In the MULTICOLORED INDEPENDENT SET problem we are given a graph G and a partition $V_1, \ldots, V_k$ of its vertex set $V(G)$ and the question is whether there is an independent set $\{v_1, \ldots, v_k\} \subseteq V(G)$ in G such that for each $i \in [k]$, $v_i \in V_i$. The W[1]-hardness of this problem follows immediately from the W[1]-hardness of the MULTICOLORED CLIQUE problem.

The Reduction of Fomin et al. [13]. Let G be an instance of MULTICOLORED INDEPENDENT SET with partition $V_1, \ldots, V_k$ of $V(G)$. Again we can assume that $k \geq 3$ and that $|V_i| = p$ for all $i \in [k]$. If the latter condition does not hold, let $p := \max_{i \in [k]} |V_i|$ and for each $i \in [k]$, add $p - |V_i|$ vertices to V_i that are adjacent to all vertices in each V_j where $j \neq i$. It is clear that the resulting instance has a multicolored independent set if and only if the original instance does.

The graph G' of the MINIMUM DOMINATING SET instances is obtained as follows. We take the graph G'' as constructed in the proof of Theorem 21, and for each $i \in [k]$, we add a vertex b_i whose model is $\{u_i\}$, i. e. it is adjacent to all vertices in $Z(i)$ and nothing else. We argue that G has a multicolored independent set if and only if G' has a dominating set of size k.

For the forward direction, if G has a multicolored independent set $I := \{v^1_{h_1}, \ldots, v^k_{h_k}\}$, then using Observation 21.1, one can verify that $D := \{z^1_{h_1}, \ldots, z^k_{h_k}\}$ is a dominating set in G': Clearly, for each $i \in [k]$, the vertices in $Z(i) \cup \{b_i\}$ are dominated by $z^i_{h_i} \in D$. Suppose there is a vertex $r^{(i,j)}_{s,t} \in R(i,j)$ that is not dominated by D, then in particular it is neither adjacent to $z^i_{h_i}$ nor to $z^j_{h_j}$. By Observation 21.1, this implies that G contains the edge $v^i_{h_i} v^j_{h_j}$, a contradiction with the fact that I is an independent set.

For the backward direction, suppose that G' has a dominating set D of size k. Due to the vertices b_i (for $i \in [k]$), we can conclude that for all $i \in [k]$, $D \cap (Z(i) \cup \{b_i\}) \neq \emptyset$. If D contains b_i for some $i \in [k]$, then we can replace b_i by any vertex in $Z(i)$ such that the resulting set is still a dominating set of D, so we can assume that $D = \{z^1_{h_1}, \ldots, z^k_{h_k}\}$. We claim that $\{v^1_{h_1}, \ldots, v^k_{h_k}\}$ is an independent set in G. Suppose that for $i, j \in [k]$, there is an edge $v^i_{h_i} v^j_{h_j} \in E(G)$. Observation 21.1 implies that $r^{(i,j)}_{h_i,h_j}$ is neither adjacent to $z^i_{h_i}$ nor to $z^j_{h_j}$, so $r^{(i,j)}_{h_i,h_j}$ is not dominated by D, a contradiction.

Remark 23. We would like to remark that the above reduction is to the MIN-(σ^*, ρ^*) DOMINATION problem, for all $\sigma^* \subseteq \mathbb{N}$ with $0 \in \sigma^*$ and $\rho^* \subseteq \mathbb{N}^+$ with $\{1,2\} \subseteq \rho^*$.

Proposition 24 ([13])**.** *For $\sigma^* \subseteq \mathbb{N}$ with $0 \in \sigma^*$ and $\rho^* \subseteq \mathbb{N}^+$ with $\{1,2\} \subseteq \rho^*$,* MIN-(σ^*, ρ^*) DOMINATION *is* W[1]*-hard on H-graphs parameterized by the number of edges in H plus solution size, and the hardness holds even when an H-representation of the input graph is given.*

Adaption to Total Domination Problems. Recall that the (σ, ρ) formulation for DOMINATING SET is $(\mathbb{N}, \mathbb{N}^+)$. We now explain how to modify the above reduction to obtain hardness for total dominating set problems where each vertex in the solution has to have at least one neighbor in the solution as well. These problems include TOTAL DOMINATING SET and DOMINATING INDUCED MATCHING, which can be formulated as $(\mathbb{N}^+, \mathbb{N}^+)$ and $(\{1\}, \mathbb{N}^+)$, respectively. The minimization problem of either of them is known to be NP-complete.

Theorem 25. *For $\sigma^* \subseteq \mathbb{N}^+$ with $1 \in \sigma^*$ and $\rho^* \subseteq \mathbb{N}^+$ with $\{1,2\} \subseteq \rho^*$,* MIN-(σ^*, ρ^*) DOMINATION *is* W[1]*-hard on H-graphs parameterized by the number of edges in H plus solution size, and the hardness holds even when an H-representation of the input graph is given.*

Proof. We modify the above reduction from MULTICOLORED INDEPENDENT SET as follows. For each $i \in [k]$, we add another vertex c_i to G' which is only adjacent to b_i. We let $B := \bigcup_{i \in [k]} \{b_i\}$ and $C := \bigcup_{i \in [k]} \{c_i\}$. Note that these new vertices can be 'hardcoded' into H with the number of edges in H increasing only by k. To argue the correctness of the reduction, we now show that G has a multicolored independent set if and only if G' has a (σ^*, ρ^*) set of size $k' := 2k$.

For the forward direction, suppose that G has an independent set $\{v^1_{h_1}, \ldots, v^k_{h_k}\}$. Then, $D' := \{z^1_{h_1}, \ldots, z^k_{h_k}\}$ dominates all vertices in $V(G') \setminus C$ by the same argument as above and $D := D' \cup B$ dominates all vertices of G'. Furthermore, each $x \in D$ has precisely one neighbor in D: For each such x, either $x = z^i_{h_i}$ or $x = b_i$ for some $i \in [k]$. In the former case, $N(x) \cap D = \{b_i\}$ and in the latter case, $N(x) \cap D = \{z^i_{h_i}\}$. Now let $y \in V(G') \setminus D$. If $y \in Z(i) \cup \{c_i\}$ for $i \in [k]$, then $\emptyset \neq N(y) \cap D \subseteq \{z^i_{h_i}, b_i\}$. If $y \in R(i,j)$ for some $1 \leq i < j \leq k$, then y is either dominated by one of $z^i_{h_i}$ and $z^j_{h_j}$ or by both and it cannot have any other neighbors in D by construction. Since $1 \in \sigma^*$ and $\{1,2\} \subseteq \rho^*$, D is a (σ^*, ρ^*) set and clearly, $|D| = 2k$.

For the backward direction, suppose that G' has a (σ^*, ρ^*) set D of size $2k$. Let $i \in [k]$. Since $0 \notin \sigma^*$ and $0 \notin \rho^*$, we have that at least one of c_i and b_i is contained in D (either c_i is dominating or it needs to be dominated). Suppose $c_i \in D$. Since each vertex in D has to have at least one neighbor in D and b_i is the only neighbor of c_i, we can conclude that $b_i \in D$. So, in either case, we have that b_i is contained in D and subsequently we have that $B \subseteq D$. Since $0 \notin \sigma^*$, all vertices of B have a neighbor in D. Suppose for some $i \in [k]$ that neighbor is c_i. Then, we can replace c_i with some $z^i_{h_i} \in Z(i)$, without changing the fact that D is a (σ^*, ρ^*) set. We can assume that for each $i \in [k]$, the neighbor of b_i that is contained in D is a vertex $z^i_{h_i} \in Z(i)$.

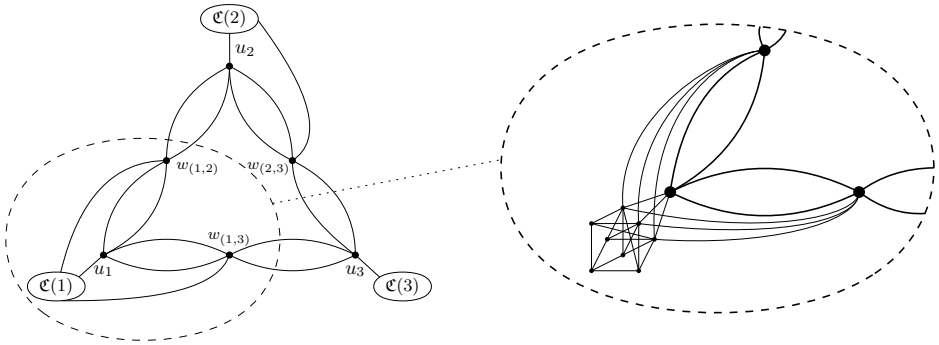


Figure 6: An example graph K w.r.t. which the graph G' constructed in the proof of Theorem 26 is a K-graph. In this example, $k = 3$.

We have that $D = B \cup \{z^1_{h_1}, \ldots, z^k_{h_k}\}$ and since D is a dominating set (in other words, $0 \notin \rho^*$), we can again argue using Observation 21.1 that $\{v^1_{h_1}, \ldots, v^k_{h_k}\}$ is an independent set in G. □

As a somewhat orthogonal result to Theorem 21, we now show hardness of several problems related to the d-DOMINATING SET problem, where each vertex that is not in the solution set has to be dominated by at least some fixed number of d neighbors in the solution.

Adaption to d-Domination Problems. We use a similar gadget as the one constructed in the proof of Theorem 21 to prove hardness of several (σ, ρ) problems where each vertex has to be dominated by at least d vertices. In particular, we prove the following theorem. Note that the analogous statement of the following theorem for $d = 1$ is proved by the reduction explained in the beginning of this section, see Remark 23.

Theorem 26. *For any fixed $d \in \mathbb{N}_{\geq 2}$, the following holds. Let $\sigma^* \subseteq \mathbb{N}$ with $\{0, 1, d-1\} \subseteq \sigma^*$ and $\rho^* \subseteq \mathbb{N}_{\geq d}$ with $\{d, d+1\} \subseteq \rho^*$. Then,* MIN-$(\sigma^*, \rho^*)$ DOMINATION *is* W[1]*-hard on H-graphs parameterized by the number of edges in H plus solution size, and the hardness even holds when an H-representation of the input graph is given.*

Proof. We modify the reduction from MULTICOLORED INDEPENDENT SET to DOMINATING SET on H-graphs due to Fomin et al. [13] that we summarized in the beginning of this section. Let G be a graph with vertex partition $V_1, \ldots, V_k$ and $|V_i| = p$ for all $i \in [k]$ and assume $k \geq 3$. We first describe the gadget we use and then we describe how to construct the graph G' of the MIN-(σ^*, ρ^*) DOMINATION instance.

The Gadget $\mathfrak{C}(i)$. Let $i \in [k]$. The gadget $\mathfrak{C}(i)$ is a complete bipartite graph with bipartition $(C_1(i), C_2(i))$ where $C_1(i) := \{c^i_{1,1}, \ldots, c^i_{1,d}\}$ and $C_2(i) := \{c^i_{2,1}, \ldots, c^i_{2,d}\}$ such that each vertex $c^i_{1,j}$ for $j \in [d-1]$ is additionally adjacent to all vertices in $Z(i)$ as well as to all vertices in $R(i,j)$ for $j > i$. (Note that $c^i_{1,d}$ does not have these additional adjacencies.) Throughout the following, we let $C(i) := C_1(i) \cup C_2(i)$ and $C := \bigcup_{i \in [k]} C(i)$.

The graph G' is now obtained by constructing the graph G'' as in the proof of Theorem 21 and then, for each $i \in [k]$, adding the gadget $\mathfrak{C}(i)$ and adding a 'satellite vertex' s_i, adjacent to all vertices in $Z(i) \cup C_1(i)$. G' is a K-graph for the graph $K \supseteq H$, obtained by 'hardcoding' each $\mathfrak{C}(i)$, for $i \in [k]$, into H. That is, for each $i \in [k]$, we add a complete bipartite graph with bipartition

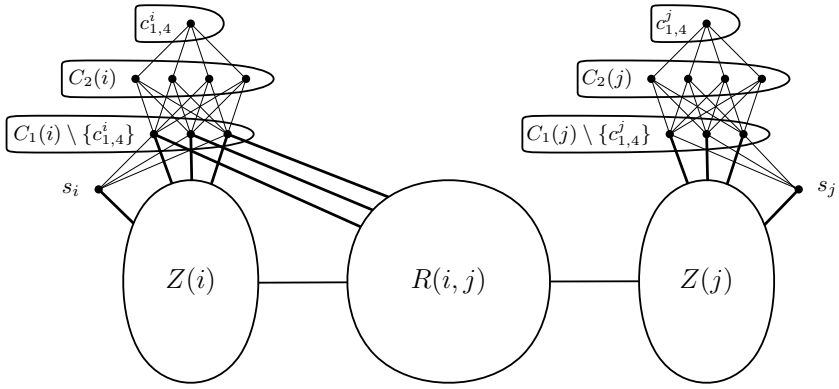


Figure 7: Illustration of a part of G' constructed in the proof of Theorem 26, where $1 \leq i < j \leq k$ and $d = 4$.

$(\{\gamma^i_{1,1}, \ldots, \gamma^i_{1,d}\}, \{\gamma^i_{2,1}, \ldots, \gamma^i_{2,d}\})$, and make all vertices $\gamma^i_{1,h}$, where $h \in [d-1]$, adjacent to u_i as well as to all vertices $w_{(i,j)}$ with $j > i$. For an illustration of K see Figure 6. Note that

$$||K|| = ||H|| + k(d^2+1) + \sum_{i=1}^{k}(k-i)(d-1) = \mathcal{O}(k^2 \cdot d + k \cdot d^2). \tag{4}$$

We illustrate the structure of the graph G' in Figure 7. We now argue that G' is a K-graph. We begin by constructing a subdivision K' of K. First, we do Step K1 that was taken in the proof of Theorem 21 (see page) to construct the subdivision, and then the analogue of Step K2 in the proof of Theorem 21 for the gadgets $\mathfrak{C}(i)$. We continue with the following two steps.

(K3) For each $i \in [k]$, let $\{\gamma^i_{1,1}, \ldots, \gamma^i_{1,d}, \gamma^i_{2,1}, \ldots, \gamma^i_{2,d-1}\}$ be the vertices of the copy of the graph of $\mathfrak{C}(i)$ in K. For each $h \in [d-1]$, we subdivide the edge $u_i\gamma^i_{1,h}$ once, and denote the corresponding subdivision node by $s(i,h)$.

(K4) Furthermore, for each $i \in [k]$, for each $i < j \leq k$, and $h \in [d-1]$ we subdivide the edge $w_{(i,j)}\gamma^i_{1,h}$ once, and denote the corresponding subdivision node by $s((i,j),h)$.

We now sketch how to obtain a K-representation of G', $\mathscr{M}' = \{M'_v\}_{v \in V(G')}$, where each M'_v is a connected subset of $V(K')$; for an illustration of G' see Figure 7. As these steps are very similar to Steps (R1) to (R5) in the proof of Theorem 21 (see page), we focus on pointing out how to adapt them rather than fully restating all of them.

First, for each $i \in [k]$, the model for the vertex s_i consists of the vertex u_i, i.e. we add the model $M'_{s_i} := \{u_i\}$ to $\mathscr{M}'$. Then we do the steps analogous to Steps R1 and R2 taken in the proof of Theorem 21. Following that, we take the steps analogous to R4 and R5, where in Step R4 we consider vertex $c^i_{1,h}$ instead of vertex $b^i_{1,h}$, and in Step R5, we consider vertex $c^i_{1,h}$ instead of vertex $b^{(i,j)}_{1,h}$. Furthermore, in Step R4, we additionally add $s(i,h)$ to the model of s_i. This completes the construction of the K-representation for G'.

Claim 26.1. If G has a multicolored independent set, then G' has a (σ^*, ρ^*) set of size $k' := k \cdot (d+1)$.

Proof. Let $\{v^1_{h_1}, \ldots, v^k_{h_k}\}$ be the independent set in G. By the reduction proving Proposition 24 in the beginning of this section, we have that $D' := \{z^1_{h_1}, \ldots, z^k_{h_k}\}$ is a $(\{0\},\{1,2\})$-set of $G'-C$ of size k (see also Remark 23). Let $C_1 := \bigcup_{i\in[k]} C_1(i)$, $C_2 := C \setminus C_1$ and $D := D' \cup C_1$.

Since each vertex in $V(G') \setminus (D \cup C)$ is adjacent to precisely $d-1$ vertices in C_1 and to either one or two vertices in D' (and $D' \cap C_1 = \emptyset$), we can conclude that each vertex in $V(G') \setminus (D \cup C)$ is adjacent to either d or $d+1$ vertices in D. Since each $C(i)$ induces a $K_{d,d}$, we can conclude that all vertices in C_2 have d neighbors in D as well. Furthermore, $N(s_i) \cap D = (C_1(i) \setminus \{c^i_{1,d}\}) \cup \{z^i_{h_i}\}$, so we have that all vertices in G' that are not contained in D have either d or $d+1$ neighbors in D.

Let $i \in [k]$. Then, $N(z^i_{h_i}) \cap D = \{c^i_{1,1}, \ldots, c^i_{1,d-1}\}$, $N(c^i_{1,d}) \cap D = \emptyset$ and for $\ell \in [d-1]$, $N(c^i_{1,\ell}) \cap D = \{z^i_{h_i}\}$. We can conclude that D is a $(\{0, 1, d-1\}, \{d, d+1\})$-set in G' and clearly, $|D| = k + kd = k'$. ⌟

In what follows, the strategy is to argue that each (σ^*, ρ^*) set of size $k' = k \cdot (d+1)$ contains a set $\{z^1_{h_1}, \ldots, z^k_{h_k}\}$ which will imply that $\{v^1_{h_1}, \ldots, v^k_{h_k}\}$ is an independent set in G. Throughout the following, for $i \in [k]$, we let $Z_+(i) := Z(i) \cup C(i) \cup \{s_i\}$.

Claim 26.2. For all $i \in [k]$, any (σ^*, ρ^*) set D in G' contains at least d vertices from $C(i)$ and at least $d+1$ vertices from $Z_+(i)$.

Proof. We first show that each such D contains at least d vertices from $C(i)$. Suppose not, then $|D \cap C(i)| \leq d-1$ for some $i \in [k]$. If $c^i_{1,d} \notin D$, then $C_2(i) \subseteq D$, otherwise $c^i_{1,d}$ cannot have d or more neighbors in D. But $|C_2(i)| = d$, a contradiction. We can assume that $c^i_{1,d} \in D$. Furthermore, there is at least one vertex $c^i_{2,\ell}$ for $\ell \in [d]$ with $c^i_{2,\ell} \notin D$. To ensure that $c^i_{2,\ell}$ has at least d neighbors in D, we would have to include all remaining vertices from $C_1(i)$ in D, but then $|D \cap C(i)| \geq d$, a contradiction. The second part of the claim now follows since the vertex s_i only has neighbors in $Z_+(i)$ and at most $d-1$ neighbors in $C(i) \cap D$ (namely $C_1(i) \setminus \{c^i_{1,d}\}$): Since D is a (σ^*, ρ^*) set, it either has to contain s_i or at least one additional neighbor of s_i. ⌟

Claim 26.3. For all $i \in [k]$, any (σ^*, ρ^*) set D of size at most $k' = k(d+1)$ contains $C_1(i)$. We furthermore can assume that it additionally contains some $z^i_{h_i} \in Z(i)$, where $h_i \in [p]$.

Proof. By Claim 26.2 we have that D contains $d+1$ vertices from each $Z_+(i')$, $i' \in [k]$, and no other vertices. Consider any vertex $z^i_s \in Z(i)$ (where $s \in [p]$) that is not contained in D. Recall that z^i_s has to have at least d neighbors in D. By Claim 26.2, z^i_s has precisely one neighbor in $(Z(i) \cup \{s_i\}) \cap D$ and since D does not contain any vertex from any $R(j,i)$ $(1 \leq j < i)$ or $R(i,j')$ $(i < j' \leq k)$, the only possible neighbors of z^i_s in D are $(C_1(i) \cup \{s_i\}) \setminus \{c^i_{1,d}\}$. Furthermore, we observe that $c^i_{1,d} \in D$: for if $c^i_{1,d} \notin D$, then $c^i_{1,d}$ has to have either d or $d+1$ neighbors in D. However, D already contains the $d-1$ vertices $C_1(i) \setminus \{c^i_{1,d}\}$ that are not adjacent to $c^i_{1,d}$, and D contains $d+1$ vertices from $Z_+(i)$. So, at most two neighbors of $c^i_{1,d}$ are contained in D. If at most one neighbor of $c^i_{1,d}$ is contained in D, this immediately gives a contradiction with D being a (σ^*, ρ^*) set since $d \geq 2$. However, if $d = 2$, and two neighbors of $c^i_{1,d}$ are contained in D, then each vertex in $Z(i)$ has only $d-1$ neighbors in D, namely the ones in $C_1(i) \setminus \{c^i_{1,d}\}$, again a contradiction with D being a (σ^*, ρ^*) set. We can conclude that $C_1(i) \subseteq D$.

Now suppose that $s_i \in D$. Then, after swapping s_i with any vertex in $Z(i)$, the resulting set remains a (σ^*, ρ^*) set: Clearly, the condition of being a (σ^*, ρ^*) set is not violated by any

vertex in $Z_+(i)$. For $i < j \leq k$, consider any vertex $x \in R(i,j)$. Then, $N(x) \cap D$ contains the $d-1$ vertices $C_1(i) \setminus \{c^i_{1,d}\}$, and at most one more each from $Z(i)$ and $Z(j)$, as D can contain at most one vertex from each $Z(i')$, $i' \in [k]$. Now, if we swapped s_i with some vertex from $Z(i)$, then this means that initially, D contained d neighbors from $N(x)$, namely $C_1(i) \setminus \{c^i_{1,d}\}$ and one vertex from $Z(j)$. Hence, after swapping, D contains $d+1$ vertices and since $d+1 \in \rho^*$, D remained a (σ^*, ρ^*) set. An analogous argument can be given for any $R(j',i)$, where $1 \leq j' < i$. ⌟

We are now ready to conclude the correctness proof of the reduction.

Claim 26.4. If G' has a (σ^*, ρ^*) set of size $k' = k(d+1)$, then G has a multicolored independent set.

Proof. Let D be a (σ^*, ρ^*) set of size k'. By Claim 26.3, we can assume that $D = C_1 \cup \{z^1_{h_1}, \ldots, z^k_{h_k}\}$ for some $h_1, \ldots, h_k \in [p]$. Now, since for each $1 \leq i < j \leq k$, all vertices in $R(i,j)$ have precisely $d-1$ neighbors in C_1, each of them has to have at least one of $z^i_{h_i}$ and $z^j_{h_j}$ as a neighbor. By Observation 21.1, this allows us to conclude that $\{v^1_{h_1}, \ldots, v^k_{h_k}\}$ is an independent set in G. ⌟

Claims 26.1 and 26.4 establish the correctness of the reduction. Clearly, $|V(G')| = \mathcal{O}(|V(G)| + d^2 \cdot k)$ (and G' can be constructed in polynomial time) and by (4), $||K|| = \mathcal{O}(k^2 \cdot d + k \cdot d^2)$. Since d is a fixed constant we have that $||K|| = \mathcal{O}(k^2)$ and the theorem follows. □

Similarly to above, a combination of the previous two theorems with Proposition 20 yields the following hardness results for (σ, ρ) mimization problems on graphs of bounded linear mim-width.

Corollary 27. *Let $\sigma^* \subseteq \mathbb{N}$ and $\rho^* \subseteq \mathbb{N}$. Then,* MIN-$(\sigma^*, \rho^*)$ DOMINATION *is* W[1]*-hard parameterized by linear mim-width plus solution size, if one of the following holds.*

(i) $\sigma^* \subseteq \mathbb{N}^+$ with $1 \in \sigma^*$ and $\rho^* \subseteq \mathbb{N}^+$ with $\{1,2\} \subseteq \rho^*$.

(ii) For some fixed $d \in \mathbb{N}_{\geq 2}$, $\{0, 1, d-1\} \subseteq \sigma^*$ and $\rho^* \subseteq \mathbb{N}_{\geq d}$ with $\{d, d+1\} \subseteq \rho^*$.

Furthermore, the hardness holds even if a corresponding decomposition tree is given.

6 Conclusion

We have introduced the class of distance-r (σ, ρ) and LCVP problems. This generalizes well-known graph distance problems like distance-r domination, distance-r independence, distance-r coloring and perfect r-codes. It also introduces many new distance problems for which the standard distance-1 version naturally captures a well-known graph property.

Using the graph parameter mim-width, we showed that all these problems are solvable in polynomial time for many interesting graph classes. These meta-algorithms will have runtimes which can likely be improved for a particular problem on a particular graph class. For instance, blindly applying our results to solve DISTANCE-r DOMINATING SET on permutation graphs yields an algorithm that runs in time $\mathcal{O}(n^8)$: Permutation graphs have linear mim-width 1 (with a corresponding decomposition tree that can be computed in linear time) [3, Lemmas 2 and 5], so we can apply Corollary 15(i). However, there is an algorithm that solves DISTANCE-r DOMINATING SET on permutation graphs in time $\mathcal{O}(n^2)$ [26]; a much faster runtime.

Recently, Chiarelli et al. [10] gave algorithms for the (TOTAL) k-DOMINATING SET problems on proper interval graphs that run in time $\mathcal{O}(n^{3k})$. The mim-width framework yields an algorithm for these problems that runs in time $\mathcal{O}(n^{2k+4})$ which follows from Proposition 3(i) and the result that interval graphs have linear mim-width 1 [3]. Hence the work [10] improves the generic mim-width based algorithm whenever $k < 4$. We would like to remind the reader however that prior to this work, the result formulated in Proposition 3(i) has not been explicitly stated anywhere.

Regarding lower bounds, we expanded on the previous results by Fomin et al. [13] and showed that many (σ,ρ) problems are W[1]-hard parameterized by mim-width. However, it remains open whether there exists a problem which is NP-hard in general, yet FPT parameterized by mim-width. In particular, several (σ,ρ) problems are not covered by the W[1]-hardness results of Fomin et al. [13] and the ones presented in this paper. Examples include PERFECT CODE and PERFECT DOMINATING SET, see e.g. Table 1. Even so, we conjecture that every NP-hard (distance) (σ,ρ) problem is W[1]-hard parameterized by mim-width.

Somewhat surprisingly, we proved that powers of graphs of bounded tree-width or clique-width have bounded mim-width. Heggernes et al. [14] showed that the clique-width of the k-th power of a path of length $k(k+1)$ is exactly k. This also shows that the expressive power of mim-width is much stronger than clique-width, since all powers of paths have mim-width just 1. As a special case, we show that leaf power graphs have mim-width 1. We believe the notion of mim-width can be of benefit to the study of leaf power graphs. We remark that it is a big open problem whether leaf power graphs can be recognized in polynomial time [6, 8, 22, 24]. Knowing that leaf powers have mim-width 1, we can connect this open problem to the open problem regarding the recognition of graphs of bounded mim-width which has been repeatedly stated (e.g. [18, 31]): A polynomial-time algorithm that recognizes graphs of mim-width 1 could prove itself useful in devising a recognition algorithm for leaf power graphs.

Acknowledgements. We would like to thank the anonymous reviewers whose numerous comments improved the quality of the presentation in this paper.

References

[1] Geir Agnarsson, Peter Damaschke, and Magnús M. Halldórsson. Powers of geometric intersection graphs and dispersion algorithms. *Discrete Appl. Math.*, 132(1-3):3–16, 2003.

[2] Gábor Bacsó, Dániel Marx, and Zsolt Tuza. H-free graphs, independent sets, and subexponential-time algorithms. In *Proc. IPEC '16*, volume 63 of *LIPIcs*, pages 3:1–3:12. Schloss Dagstuhl, 2017.

[3] Rémy Belmonte and Martin Vatshelle. Graph classes with structured neighborhoods and algorithmic applications. *Theor. Comput. Sci.*, 511:54 – 65, 2013.

[4] Norman Biggs. Perfect codes in graphs. *J. Combin. Theory, Ser. B*, 15(3):289 – 296, 1973.

[5] A. Brandstädt, V. Le, and J. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.

[6] Andreas Brandstädt. On leaf powers. Technical report, University of Rostock, 2010.

[7] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theor. Comput. Sci.*, 511:66 – 76, 2013.

[8] Tiziana Calamoneri and Blerina Sinaimeri. Pairwise compatibility graphs: A survey. *SIAM Review*, 58(3):445–460, 2016.

[9] Steven Chaplick, Martin Töpfer, Jan Voborník, and Peter Zeman. On H-topological intersection graphs. In *Proc. WG '17*, volume 10520 of *LNCS*, pages 167–179. Springer, 2017.

[10] Nina Chiarelli, Tatiana Romina Hartinger, Valeria Alejandra Leoni, Maria Inés Lopez Pujato, and Martin Milanič. Improved algorithms for k-domination and total k-domination in proper interval graphs. In *Proc. ISCO '18*, pages 290 – 302, 2018.

[11] Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009.

[12] Carsten Flotow. On powers of m-trapezoid graphs. *Discrete Appl. Math.*, 63(2):187 – 192, 1995.

[13] Fedor V. Fomin, Petr A. Golovach, and Jean-Florent Raymond. On the tractability of optimization problems on H-graphs. In *Proc. ESA '18*, volume 112 of *LIPIcs*, pages 30:1 – 30:14. Schloss Dagstuhl, 2018.

[14] Pinar Heggernes, Daniel Meister, Charis Papadopoulos, and Udi Rotics. Clique-width of path powers. *Discrete Applied Mathematics*, 205:62–72, 2016.

[15] M. A. Henning, Ortrud R. Oellermann, and Henda C. Swart. Bounds on distance domination parameters. *J. Combin. Inform. Syst. Sci.*, 16(1):11–18, 1991.

[16] Lars Jaffke, O-joung Kwon, Torstein J. F. Strømme, and Jan Arne Telle. Generalized distance domination problems and their complexity on graphs of bounded mim-width. In *Proc. IPEC '18*, volume 115 of *LIPIcs*, pages 6:1–6:13. Schloss Dagstuhl, 2018.

[17] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Polynomial-time algorithms for the longest induced path and induced disjoint paths problems on graphs of bounded mim-width. In *Proc. IPEC '17*, volume 89 of *LIPIcs*, pages 21:1–21:13. Schloss Dagstuhl, 2017.

[18] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width I. Induced path problems. *Discrete Applied Mathematics*, 2019. doi:https://doi.org/10.1016/j.dam.2019.06.026.

[19] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Mim-width II. The feedback vertex set problem. *Algorithmica*, 2019. doi:https://doi.org/10.1007/s00453-019-00607-3.

[20] Lars Jaffke, O-joung Kwon, and Jan Arne Telle. A unified polynomial-time algorithm for feedback vertex set on graphs of bounded mim-width. In *Proc. STACS '18*, volume 96 of *LIPIcs*, pages 42:1–42:14. Schloss Dagstuhl, 2018.

[21] Ton Kloks. *Treewidth: Computations and approximations*, volume 842 of *LNCS*. Springer, 1994.

[22] Manuel Lafond. On strongly chordal graphs that are not leaf powers. In *Proc. WG '17*, volume 10520 of *LNCS*, pages 386–398. Springer, 2017.

[23] Stefan Mengel. Lower bounds on the mim-width of some graph classes. *Discrete Applied Mathematics*, 248:28–32, 2018.

[24] Ragnar Nevries and Christian Rosenke. Towards a characterization of leaf powers by clique arrangements. *Graphs and Combinatorics*, 32(5):2053–2077, 2016.

[25] Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. Comput. Syst. Sci.*, 67(4):757–771, 2003.

[26] Akul Rana, Anita Pal, and Madhumangal Pal. An efficient algorithm to solve the distance k-domination problem on permutation graphs. *J. Discrete Math. Sci. Cryptography*, 19(2):241–255, 2016.

[27] Arundhati Raychaudhuri. On powers of strongly chordal and circular arc graphs. *Ars Combin.*, 34:147–160, 1992.

[28] Sigve Hortemo Sæther and Martin Vatshelle. Hardness of computing width parameters based on branch decompositions over the vertex set. *Theor. Comput. Sci.*, 615:120–125, 2016.

[29] Peter J. Slater. R-domination in graphs. *J. ACM*, 23(3):446–450, 1976.

[30] Jan Arne Telle and Andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Discrete Math.*, 10(4):529–550, 1997.

[31] Martin Vatshelle. *New width parameters of graphs*. PhD thesis, University of Bergen, Norway, 2012.

Chapter 9

Subgraph Complementation

Fedor V. Fomin, Petr A. Golovach, Torstein J. F. Strømme and Dimitrios M. Thilikos. In *Algorithmica* (2020).

A preliminary version appeared in SWAT 2018, LIPIcs 101, Pages 21:1–21:13, 2018, and is available online with DOI 10.4230/LIPIcs.SWAT.2018.21.

Subgraph Complementation*

Fedor V. Fomin[1], Petr A. Golovach[1], Torstein J. F. Strømme[†1], and Dimitrios M. Thilikos[2]

[1]University of Bergen, Bergen, Norway
[2]LIRMM, Univ. Montpellier, CNRS, Montpellier, France

Abstract

A *subgraph complement* of the graph G is a graph obtained from G by complementing all the edges in one of its induced subgraphs. We study the following algorithmic question: for a given graph G and graph class $\mathscr{G}$, is there a subgraph complement of G which is in $\mathscr{G}$? We show that this problem can be solved in polynomial time for various choices of the graphs class $\mathscr{G}$, such as bipartite, d-degenerate, or cographs. We complement these results by proving that the problem is NP-complete when $\mathscr{G}$ is the class of regular graphs.

1 Introduction

One of the most important questions in graph theory concerns the efficiency of recognition of a graph class $\mathscr{G}$. For example, how fast we can decide whether a graph is chordal, 2-connected, triangle-free, of bounded treewidth, bipartite, 3-colorable, or excludes some fixed graph as a minor? In particular, the recent developments in parameterized algorithms are partially driven by the problems of recognizing of graph classes which differ only up to a "small disturbance" from graph classes recognizable in polynomial time. The amount of disturbance is quantified in "atomic" operations required for modifying an input graph into the "well-behaving" graph class $\mathscr{G}$. The standard operations could be edge/vertex deletions, additions or edge contractions. Many problems in graph algorithms fall into this graph modification category: is it possible to add at most k edges to make a graph 2-edge connected or to make it chordal? Or is it possible to delete at most k vertices such that the resulting graph has no edges or contains no cycles?

A rich subclass of modification problems concerns edge editing problems. Here the "atomic" operation is the change of adjacency, i. e. for a pair of vertices u, v, we can either add an edge uv or delete the edge uv. For example, the CLUSTER EDITING problem asks to transform an input graph into a cluster graph — that is, a disjoint union of cliques — by flipping at most k adjacency relations.

*The first three authors have been supported by the Research Council of Norway via the projects "CLASSIS" and "MULTIVAL". The fourth author has been supported by project "DEMOGRAPH" (ANR-16-CE40-0028). An extended abstract of this paper was presented in [9].

†Corresponding author.

Besides the basic edge editing, it is natural to consider problems where the set of removed and added edges should satisfy some structural constraints. In particular, such problems were considered for *complementation* problems. Recall that the *complement* of a graph G is a graph H on the same vertices such that two distinct vertices of H are adjacent if and only if they are not adjacent in G. Seidel (see [22, 23, 24]) introduced the operation that is now known as the *Seidel switch*. For a vertex v of a graph G, this operation complements the adjacencies of v, that is, it removes the edges incident to v and makes v adjacent to the non-neighbors of v in G. Seidel switching a set of vertices U entails consecutively switching each vertex in the set. The result is that all adjacencies between U and its complement $V(G)\setminus U$ are flipped. The study of the algorithmic question whether it is possible to obtain a graph from a given graph class by the Seidel switch was initiated by Ehrenfeucht et al. [7]. Further results were established in [14, 15, 16, 18, 19].

Another important operation of this type is the *local complementation*. For a vertex v of a graph G, the *local complementation of G at v* is the graph obtained from G by replacing $G[N(v)]$ by its complement. This operation plays crucial role in the definition of *vertex-minors* [20] and was investigated in this context (see, e.g. [6, 21]). See also [2, 17] for some algorithmic results concerning local complementations.

In this paper we study the *subgraph complement* of a graph, which was introduced by Kamiński, Lozin, and Milanič in [17] in their study of the clique-width of a graph. A *subgraph complement* of a graph G is a graph obtained from G by complementing all the edges of one of its induced subgraphs. More formally, for a graph G and $S\subseteq V(G)$, we define $G\oplus S$ as the graph with the vertex set $V(G)$ whose edge set is defined as follows: a pair of distinct vertices u,v is an edge of $G\oplus S$ if and only if one of the following holds:

- $uv\in E(G)\wedge(u\notin S\vee v\notin S)$, or
- $uv\notin E(G)\wedge u\in S\wedge v\in S$.

Thus when the set S consists only of two vertices $\{u,v\}$, the operation simply changes the adjacency between u and v, and for a larger set S, $G\oplus S$ changes the adjacency relations for all pairs of vertices of S.

We say that a graph H is a subgraph complement of the graph G if H is isomorphic to $G\oplus S$ for some $S\subseteq V(G)$. For a graph class $\mathscr{G}$ and a graph G, we say that there is a *subgraph complement of G to $\mathscr{G}$* if for some $S\subseteq V(G)$, we have $G\oplus S\in\mathscr{G}$. We denote by $\mathscr{G}^{(1)}$ the class of graphs such that its members can be subgraph complemented to $\mathscr{G}$.

Let $\mathscr{G}$ be a graph class. We consider the following generic algorithmic problem.

SUBGRAPH COMPLEMENT TO $\mathscr{G}$ (SC$\mathscr{G}$) **Input:** A simple undirected graph G. **Question:** Is there a subgraph complement of G to $\mathscr{G}$?

In other words, how difficult is it to recognize the class $\mathscr{G}^{(1)}$? In this paper we show that there are many well-known graph classes $\mathscr{G}$ such that $\mathscr{G}^{(1)}$ is recognizable in polynomial time. We show that

- SUBGRAPH COMPLEMENT TO $\mathscr{G}$ is solvable in $\mathcal{O}(f(n)\cdot n^4+n^6)$ time when $\mathscr{G}$ is a triangle-free graph class recognizable in $f(n)$ time. For example, this implies that when $\mathscr{G}$ is the class of bipartite graphs, the class $\mathscr{G}^{(1)}$ is recognizable in polynomial time. This result is found in Section 3.

- SUBGRAPH COMPLEMENT TO $\mathscr{G}$ is solvable in $f(n) \cdot n^{2^{\mathscr{O}(d)}}$ time when $\mathscr{G}$ is a d-degenerate graph class recognizable in $f(n)$ time. Thus when $\mathscr{G}$ is the class of planar graphs, class of cubic graphs, class of graph of bounded treewidth, or class of H-minor free graphs, then the class $\mathscr{G}^{(1)}$ is recognizable in polynomial time. This result is found in Section 4.
- SUBGRAPH COMPLEMENT TO $\mathscr{G}$ is solvable in polynomial time when $\mathscr{G}$ is a class of bounded clique-width expressible in monadic second-order logic (with no edge set quantification). In particular, if $\mathscr{G}$ is the class of P_4-free graphs (cographs), then $\mathscr{G}^{(1)}$ is recognizable in polynomial time. This result is found in Section 6.
- SUBGRAPH COMPLEMENT TO $\mathscr{G}$ is solvable in polynomial time when $\mathscr{G}$ can be described by a 2×2 M-partition matrix. Therefore $\mathscr{G}^{(1)}$ is recognizable in polynomial time when $\mathscr{G}$ is the class of split graphs, as they can be described by such a matrix. This result is found in Section 5.

There are nevertheless cases when the problem is NP-hard. In particular, we prove that this holds when $\mathscr{G}$ is the class of regular graphs. This result is found in Section 7.

2 Preliminaries

We let $\mathbb{N} := \{0,1,2\ldots\}$ be the set of the natural numbers, and we let $\mathbb{N}^+ := \mathbb{N} \setminus \{0\}$ be the set of positive integers. For a set A and a non-negative integer $k \in \mathbb{N}$, we denote the family of all subsets of A of size k as $\binom{A}{k} := \{A' \subseteq A \mid |A'| = k\}$.

Simple Graph. A (simple) graph is a pair $G := (V,E)$, where V is a set of *vertices* and $E \subseteq \binom{V}{2}$ is a set of *edges*. For an edge $\{u,v\} \in E(G)$ we use the shorthand uv (or equivalently, vu). For a given graph G, we refer to its vertex set as $V(G)$, and its edge set as $E(G)$. For a set of edges $F \subseteq E(G)$, we denote by $V(F)$ the set of vertices that are contained in the edges of F, i. e. $V(F) := \bigcup_{uv \in F} \{u,v\}$.

(Induced) Subgraph. For graphs G and H we say that H is a *subgraph* of G, denoted by $H \subseteq G$ if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For a set of vertices $A \subseteq V(G)$, we denote by $G[A]$ the subgraph of G *induced* by A, i.e. $G[A] := (A, E(G) \cap \binom{A}{2})$. We use the notation $G - X := G[V(G) \setminus X]$ and for a set of edges $F \subseteq E(G)$, $G - F := (V(G), E(G) \setminus F)$. For a vertex $v \in V(G)$ (or an edge $e \in E(G)$), we use the shorthand $G - x := G - \{x\}$ (or $G - e := G - \{e\}$, respectively).

Neighborhood. Let G be a graph. For a vertex $v \in V(G)$, we denote by $N_G(v)$ the *open neighborhood* of v, i. e. $N_G(v) := \{w \mid vw \in E(G)\}$, and by $N_G[v]$ the *closed neighborhood* of v, i. e. $N_G[v] := \{v\} \cup N_G(v)$. The *degree* of v is the size of its open neighborhood, i.e. $\deg_G(v) := |N_G(v)|$. For a set of vertices $X \subseteq V(G)$, we let $N_G(X) := \bigcup_{v \in X} N_G(v) \setminus X$ and $N_G[X] := N_G(X) \cup X$. We use the shorthand notations 'N' and 'deg' for 'N_G' and '$\deg_G$', respectively, if G is clear from the context.

Monadic Second Order Logic. $\mathbf{MSO}_1$ is a restricted version of of $\mathbf{MSO}_2$ (Monadic Second Order Logic) without quantifications over edge subsets. More precisely, the syntax of $\mathbf{MSO}_1$ of graphs includes the logical connectives $\vee$, $\wedge$, $\neg$, $\Leftrightarrow$, $\Rightarrow$, variables for vertices and sets of vertices, the quantifiers $\forall$, $\exists$ that can be applied to these variables, and the following binary relations:

1. $u \in U$ where u is a vertex variable and U is a vertex set variable;
2. $\mathbf{adj}(u,v)$, where u and v are vertex variables and the interpretation is that u and v are adjacent;
3. equality of variables representing vertices and sets of vertices.

We refer to [3] for more information on $\mathbf{MSO}_1$ and $\mathbf{MSO}_2$.

3 Subgraph complementation to triangle-free graph classes

A triangle is a complete graph on three vertices. Many graph classes do not allow the triangle as a subgraph, for instance trees, forests, or graphs with large girth. In this section we show that subgraph complementation to triangle-free graphs can be decided in polynomial time.

More precisely, we show that if a graph class $\mathscr{G}$ can be recognized in polynomial time and it is triangle-free, then we can also solve SUBGRAPH COMPLEMENT TO $\mathscr{G}$ in polynomial time. Our algorithm is constructive, and returns a *solution* $S \subseteq V(G)$, that is a set S such that $G \oplus S$ is in $\mathscr{G}$. We say that a solution *hits* an edge uv (or a non-edge $\overline{uv}$), if both u and v are contained in S.

Our algorithm considers each of the following cases.

(*i*) There is a solution S containing two vertices that are non-adjacent in G.

(*ii*) There is a solution S such that it forms a clique of size at least 2 in G.

(*iii*) G is a no-instance.

We start from analyzing the structure of a solution in Case (*i*). We need the following observation.

Observation 1. *Let $\mathscr{G}$ be a class of triangle-free graphs and let G be an instance of* SUBGRAPH COMPLEMENT TO $\mathscr{G}$*, where $S \subseteq V(G)$ is a valid solution. Then*

a) $G[S]$ does not contain an independent set of size 3, and

b) for every triangle $\{u,v,w\} \subseteq V(G)$, at least two vertices are in S.

Because all non-edges between vertices in $G[S]$ become edges in $G \oplus S$ and vice versa, whereas all (non-) edges with an endpoint outside S remain untouched, we see that the observation holds.

Before delving into the analysis of Case (*i*) any further, let us recall that a graph G is a *split graph* if its vertex set can be partitioned into $V(G) = C \cup I$, where C is a clique and I is an independent set. Let us note that the vertex set of a split graph can have several *split partitions*, i.e. partitions into a clique and independent set. However, the number of split partitions of an n-vertex split graphs is at most $n+1$, and they can be enumerated in linear time. This builds on the result of Hammer and Simone [12] (see also [11]), for completeness we provide a proof.

Lemma 2. *Let G be a split graph on n vertices. Then G has at most $n+1$ split partitions. Moreover, they can be enumerated in linear time.*

Proof. Let ℓ denote the cardinality of a maximum clique in G. It is known that every split partition has a clique of size ℓ or $\ell-1$ [12]. We will show that there are at most n partitions where the clique has cardinality ℓ, and that these can be enumerated in linear time — since the split partitions in G and its complement $\overline{G}$ are the same up to swapping the clique and independent set, this is enough to prove the lemma with a bound of $2n$. We will strengthen the bound at the end of the proof.

Let $V_{\geq \ell} \subseteq V(G)$ denote the set of vertices whose degree is at least ℓ. We observe that in every split partition, these vertices need to be in the clique; otherwise there is either an edge between two vertices of the independent set, or it is possible to find a clique of size $\ell+1$.

Similarly, let $V_{\leq \ell-2} \subseteq V(G)$ denote the set of vertices whose degree is at most $\ell-2$. We observe that every vertex of $V_{\leq \ell-2}$ must be in the independent set of a split partition where the clique has size ℓ.

It remains to partition the set of vertices $V_{\ell-1} \subseteq V(G)$ whose degrees are $\ell-1$. We claim that every vertex of $V_{\ell-1}$ must have $V_{\geq \ell}$ contained in its neighborhood, or else there is a unique split partition whose clique of size ℓ is exactly $V_{\geq \ell}$. Assume that there is a vertex $u \in V_{\ell-1}$ whose neighborhood does not include a vertex $v \in V_{\geq \ell}$. Since v is in every clique of size ℓ, u can never be in the clique, implying that all of u's neighbors must be — but then those neighbors have degree at least ℓ, implying that $|V_{\geq \ell}| \geq \ell$.

We proceed with the case when $|V_{\geq \ell}| < \ell$. Consider some subset $U \subseteq V_{\ell-1}$ such that $U \cup V_{\geq \ell}$ is a clique of size ℓ. Then U is a clique, and vertices of U have no neighbors in $V_{\ell-1} \setminus U$. Thus $G[V_{\ell-1}]$ is a collection of cliques, each of size $\ell - |V_{\geq \ell}|$. Each split partition of size ℓ will include exactly one of these cliques in addition to $V_{\geq \ell}$. However, observe that this implies that there is either exactly one clique, or the cliques must have size 1 — otherwise there would be an edge between two vertices in the independent set. This shows that there are at most n distinct split partitions whose clique has size ℓ.

In order to enumerate the partitions in linear time, we first analyze the degree sequence to find ℓ using the result of Hammer and Simone [12] (sorting vertices by degree can be done in $\mathcal{O}(n)$ time using bucket sort). We then find the sets $V_{\geq \ell}$ and $V_{\ell-1}$. If $|V_{\geq \ell}| = \ell$, we have a unique partition; if $|V_{\geq \ell} \cup V_{\ell-1}| = \ell$ we also have a unique partition; otherwise $|V_{\geq \ell}| = \ell - 1$, and $V_{\ell-1}$ is an independent set — the split partitions with a clique of size ℓ have one pick of vertex from $V_{\ell-1}$ each.

It remains to tighten the bound on the number of split partitions to $n+1$. We show that if there is more than one split partition with a clique of size ℓ, then there is a single split partition with a clique of size $\ell-1$. In more detail: if there is more than one split partition with a clique of maximum size, we know from the above paragraph that $V_{\ell-1}$ is an independent set of size at least two, and $V_{\geq \ell}$ is a clique of size $\ell-1$. We claim that every clique of size $\ell-1$ in a split partition contains $V_{\geq \ell}$, thus making the split partition unique. Assume for the sake of contradiction that $u \in V_{\geq \ell}$ is not in the clique. Recall that u is in the neighborhood of every vertex in $V_{\ell-1}$, so a split partition placing u in the independent set in must place $V_{\ell-1}$ in the clique. But $V_{\ell-1}$ is an independent set of size at least two, so this is impossible.

We remark that the bound is tight, and can be obtained by an edgeless graph. □

Returning to the problem of subgraph complement to triangle free graph classes, we are now ready for the analysis of Case (*i*): when there is a solution S containing two vertices that are

non-adjacent in G.

Lemma 3. *Let $\mathscr{G}$ be a class of triangle-free graphs and let G be an instance of* Subgraph Complement to $\mathscr{G}$. *Let $S \subseteq V(G)$ be a valid solution which is not a clique, and let $u, v \in S$ be distinct vertices such that $uv \notin E(G)$. Then*

a) the solution S is a subset of the union of the closed neighborhoods of u and v, that is $S \subseteq N_G[u] \cup N_G[v]$;

b) every common neighbor of u and v must be contained in the solution S, that is $N_G(u) \cap N_G(v) \subseteq S$;

c) the graph $G[N(u) \setminus N(v)]$ is a split graph. Moreover, $(N(u) \setminus N(v)) \cap S$ is a clique and $(N(u) \setminus N(v)) \setminus S$ is an independent set.

Proof. We will prove each point separately, and in order.

a) Assume for the sake of contradiction that the solution S contains a vertex $w \notin N_G[u] \cup N_G[v]$. But then $\{u, v, w\}$ is an independent set in G, which contradicts Observation 1a.

b) Assume for the sake of contradiction that the solution S does not contain a vertex $w \in N_G(u) \cap N_G(v)$. Then the edges uw and vw will both be present in $G \oplus S$, as well as the edge uv. Together, these form a triangle.

c) We first claim that the solution S is a vertex cover for $G[N(u) \setminus N(v)]$. If it was not, then there would exist an edge u_1u_2 of $G[N(u) \setminus N(v)]$ such that both endpoints $u_1, u_2 \notin S$, yet u_1, u_2 would form a triangle with u in $G \oplus S$, which would be a contradiction. Hence $(N(u) \setminus N(v)) \setminus S$ is an independent set. Secondly, we claim that $(N(u) \setminus N(v)) \cap S$ forms a clique. If not, then there would exist $u_1, u_2 \in (N(u) \setminus N(v)) \cap S$ which are nonadjacent. In this case $\{u_1, u_2, v\}$ is an independent set, which contradicts Observation 1a. Taken together, these claims imply the last item of the lemma.

□

We now move on to examine the structure of a solution for Case (*ii*), when there exists a solution which is a clique. Note that we can assume the clique has size at least two, since if $|S| \leq 1$ then subgraph complementation does not alter the graph.

Lemma 4. *Let $\mathscr{G}$ be a class of triangle-free graphs and let G be an instance of* Subgraph Complement to $\mathscr{G}$. *Let $S \subseteq V(G)$ be a solution such that $|S| \geq 2$ and $G[S]$ is a clique. Let $u, v \in S$ be distinct. Then*

a) the solution S is contained in their common neighborhood, that is $S \subseteq N_G[u] \cap N_G[v]$, and

b) the graph $G[N_G[u] \cap N_G[v]]$ is a split graph where $(N_G[u] \cap N_G[v]) \setminus S$ is an independent set.

Proof. We prove each point separately, and in order.

a) Assume for the sake of contradiction that the solution S contains a vertex w which is not in the neighborhood of both u and v. This contradicts that S is a clique.

b) We claim that S is a vertex cover of $G[N_G[u] \cap N_G[v]]$. Because S is also a clique, the statement of the lemma will then follow immediately. Assume for the sake of contradiction that S is not a vertex cover. Then there exist an uncovered edge w_1w_2, where $w_1, w_2 \in N_G[u] \cap N_G[v]$, and also $w_1, w_2 \notin S$. Since $\{u, w_1, w_2\}$ form a triangle, we have by Observation 1b that at least two of these vertices are in S. That is a contradiction, so our claim holds.

□

We now have everything in place to present the algorithm.

Algorithm 5 (SUBGRAPH COMPLEMENT TO $\mathscr{G}$ where $\mathscr{G}$ is triangle-free)**.**
Input: An instance G of SC$\mathscr{G}$ where $\mathscr{G}$ is a triangle-free graph class recognizable in $f(n)$ time for some function f.
Output: A set $S \subseteq V(G)$ such that $G \oplus S$ is in $\mathscr{G}$, or a correct report that no such set exists.

1. For every non-edge $\overline{uv}$ of G:
   (a) If either $G[N(u) \setminus N(v)]$ or $G[N(v) \setminus N(u)]$ is not a split graph, skip this iteration and try the next non-edge.
   (b) Let (I_u, C_u) and (I_v, C_v) denote a split partition of $G[N_G(u) \setminus N_G(v)]$ and $G[N_G(v) \setminus N_G(u)]$ respectively. For each pair of split partitions $(I_u, C_u), (I_v, C_v)$:
      i. Construct solution candidate $S' := \{u, v\} \cup (N_G(u) \cap N_G(v)) \cup C_u \cup C_v$
      ii. If $G \oplus S'$ is a member of $\mathscr{G}$, return S'
2. For each edge $uv \in E(G)$:
   (a) If $G[N_G[u]] \cap N_G[v]]$ is not a split graph, skip this iteration and try the next edge.
   (b) For each possible split partition (I, C) of $G[N_G[u] \cap N_G[v]]$:
      i. Construct solution candidate $S' := \{u, v\} \cup C$
      ii. If $G \oplus S'$ is a member of $\mathscr{G}$, return S'
3. Return 'NONE'

Theorem 6. *Let $\mathscr{G}$ be a class of triangle-free graphs such that deciding whether an n-vertex graph is in $\mathscr{G}$ is solvable in $f(n)$ time for some function f. Then* SUBGRAPH COMPLEMENT TO $\mathscr{G}$ *is solvable in $\mathcal{O}(n^6 + n^4 \cdot f(n))$ time.*

Proof. We will prove that Algorithm 5 is correct, and that its running time is $\mathcal{O}(n^4 \cdot (n^2 + f(n)))$. We begin by proving correctness. We have three cases to consider: (*i*) There exists a solution which hits a non-edge, (*ii*) there is a solution S such that it forms a clique of size at least two in G, and (*iii*) no solution exists.

In the case that there exists a solution S hitting a non-edge uv, we will at some point guess this non-edge in Step 1 of the algorithm. By Lemma 3, we have that both $G[N_G(u) \setminus N_G(v)]$ and $G[N_G(u) \setminus N_G(v)]$ are split graphs, so we do not miss the solution S in Step 1a. Since we try every possible combination of split partitions in Step 1b, we will by Lemma 3 at some point construct S' correctly such that $S' = S$.

In the case that there exist only solutions which hit exactly a clique, we first find some edge $uv \in E(G)$. At some point we guess uv such that both endpoints are in the same solution S. By

Lemma 4b we know that $G[N_G(u) \cap N_G(v)]$ is a split graph, so we will not miss S in Step 2a. Since we try every split partition in Step 2b, we will by Lemma 4 at some point construct S' correctly such that $S' = S$.

Lastly, in the case that there is no solution, we know that there neither exists a solution which hits a non-edge, nor a solution which hits a clique. Since these three cases exhaust the possibilities, we can correctly report that there is no solution when none was found in the previous steps.

For the runtime, recall that determining whether a graph is split can be done in $\mathcal{O}(n)$ time given the degree sequence [12], and Lemma 2 establish that enumerating all split partitions can also be done in linear time.

The sub-procedure of Step 1 is performed at most $\mathcal{O}(n^2)$ times, where Step 1a takes $\mathcal{O}(n)$ time and Step 1b takes at most $\mathcal{O}(n^2 \cdot (n^2 + f(n))$ time. In total, Step 1 will take no longer than $\mathcal{O}(n^4 \cdot (n^2 + f(n)))$ time. The sub-procedure of Step 2 is performed at most $\mathcal{O}(n^2)$ times. Step 2a is done in $\mathcal{O}(n)$ time, and step 2b is done in $\mathcal{O}(n \cdot (n^2 + f(n))$ time; hence the asymptotic runtime of the entire step 2 is $\mathcal{O}(n^3 \cdot (n^2 + f(n)))$. The worst running time among these steps is Step 1, and as such the runtime of Algorithm 5 is $\mathcal{O}(n^4 \cdot (n^2 + f(n)))$. □

4 Complement to d-degenerate graphs

For a constant integer $d > 0$, we say that a graph G is d-degenerate, if every (not necessarily proper) subgraph of G has a vertex of degree at most d. For example, trees are 1-degenerate, while planar graphs are 5-degenerate.

Theorem 7. *Let $\mathcal{G}$ be a class of d-degenerate graphs such that deciding whether an n-vertex graph is in $\mathcal{G}$ is solvable in $f(n)$ time for some function f. Then* SUBGRAPH COMPLEMENT TO $\mathcal{G}$ *is solvable in $f(n) \cdot n^{\mathcal{O}^*(4^d)}$ time.*

Proof. Let G be an n-vertex graph. We are looking for a vertex subset S of G such that $G \oplus S \in \mathcal{G}$.

We start from trying all vertex subsets of G of size at most $2d$ as a candidate for S. Thus, in $\mathcal{O}(n^{2d} \cdot f(n))$ time we either find a solution or conclude that a solution, if it exists, should be of size more than $2d$.

Now we assume that $|S| > 2d$. We try all subsets of $V(G)$ of size $2d+1$. Then if G can be complemented to $\mathcal{G}$, at least one of these sets, say X, is a subset of S. In total, we enumerate $\binom{n}{2d+1}$ sets.

For each guess of X we consider the set Y of all vertices in $V(G) \setminus X$ with at least $d+1$ neighbors in X. The observation here is that most vertices from Y are in S. For every size $d+1$ subset $X' \subseteq X$, there can be at most d vertices of $Y \setminus S$ who have X' contained in their neighborhood — otherwise there remains a $K_{d+1,d+1}$ not killed by S. Because every vertex of Y has at least $d+1$ neighbors in X, it follows that if more than

$$\alpha = \binom{|X|}{d+1} \cdot d = \binom{2d+1}{d+1} \cdot d$$

vertices of Y are not in S, then $G \oplus S$ contains a complete bipartite graph $K_{d+1,d+1}$ as a subgraph, and hence $G \oplus S$ is not d-degenerate. Thus, we make at most $\binom{n}{\alpha}$ guesses on which subset of Y is (not) in S.

Similarly, let the set Z contain all vertices from $V(G) \setminus X$ with at most d neighbors in X — we observe that most vertices of Z must be outside S. Analogous to before, for any particular size $d+1$ subset $X' \subseteq X$, at most d vertices of $Z \cap S$ can avoid X' entirely in their neighborhood. Since every vertex in Z have at least $d+1$ non-neighbors in X, it follows that at most α of vertices from Z could belong to S. We make at most $\binom{n}{\alpha}$ guesses of which vertices of Z are in S. Since $V(G) = X \cup Y \cup Z$, if there is a solution S, it will be found in one from at most

$$\binom{n}{2d+1} \cdot \binom{n}{\alpha}^2 = n^{\mathcal{O}^*(4^d)}$$

guesses. Since for each set S we can check in $f(n)$ time whether $G \oplus S \in \mathscr{G}$, this concludes the proof. □

5 Complement to M-partition

Many graph classes can be defined by whether it is possible to partition the vertices of graphs in the class such that certain constraints at met. For instance, a complete bipartite graph is one which can be partitioned into two vertex sets such that every edge between the two sets is present, and no edge exists within any of the partitions. Other graph classes which can be described in a similar manner are split graphs and k-colorable graphs. Feder et al. [8] formalized such partition properties of graph classes by making use of a symmetric matrix over $\{0,1,\star\}$, called an *M-partition*.

Definition 8 (M-partition)**.** For a symmetric $k \times k$ matrix M over $\{0,1,\star\}$, we say that a graph G belongs to the graph class $\mathscr{G}_M$ if its vertices can be partitioned into k (possibly empty) sets $X_1, X_2, \ldots, X_k$ such that, for every $i \in [k]$:

- if $M[i,i] = 1$, then X_i is a clique, and
- if $M[i,i] = 0$, then X_i is an independent set, and

for every $i, j \in [k]$, $i \neq j$,

- if $M[i,j] = 1$, then every vertex of X_i is adjacent to all vertices of X_j, and
- if $M[i,j] = 0$, then there is no edge between X_i and X_j.

Note that if $M[i,j] = \star$, then there is no restriction on the edges between vertices from X_i and X_j. For example, for matrix

$$M = \begin{pmatrix} 0 & \star \\ \star & 0 \end{pmatrix}$$

the corresponding class of graphs is the class of bipartite graphs, while matrix

$$M = \begin{pmatrix} 0 & \star \\ \star & 1 \end{pmatrix}$$

identifies the class of split graphs.

In this section we prove the following theorem.

Theorem 9. *Let $\mathscr{G} = \mathscr{G}_M$ be a graph class described by an M-partition matrix of size 2×2. Then* SUBGRAPH COMPLEMENT TO $\mathscr{G}$ *is solvable in polynomial time.*

In particular, Theorem 9 yields polynomial-time algorithms for SUBGRAPH COMPLEMENT TO $\mathscr{G}$ when $\mathscr{G}$ is the class of split graphs or (complete) bipartite graphs. The proof of our theorem is based on the following beautiful dichotomy result of Feder et al. [8] on the recognition of classes $\mathscr{G}_M$ described by 4×4 matrices. Note that a symmetric $k \times k$ matrix M is said to contain another symmetric matrix M' if there exists a set of indices $X \subseteq [k]$ such that removing both rows an columns corresponding to X from M yields M'.

Proposition 10 ([8, Corollary 6.3])**.** *Suppose M is a symmetric 4×4 matrix over $\{0, 1, \star\}$. Then the recognition problem for $\mathscr{G}_M$ is*

- *NP-complete when M contains the matrix for 3-coloring or its complement, and no diagonal entry is $\star$.*
- *Polynomial time solvable otherwise.*

Lemma 11. *Let M be a symmetric $k \times k$ matrix giving rise to the graph class $\mathscr{G}_M = \mathscr{G}$. Then there exists a symmetric $2k \times 2k$ matrix M' such that for any input G to* SUBGRAPH COMPLEMENT TO $\mathscr{G}$*, it is a yes-instance if and only if G belongs to $\mathscr{G}_{M'}$. Moreover, M' can be constructed in linear time such that there are equally many 1's as 0's on its diagonal.*

Proof. Given M, we construct a matrix M' in linear time. We let M' be a matrix of dimension $2k \times 2k$, where entry $M'[i, j]$ is defined as $M[\lceil \frac{i}{2} \rceil, \lceil \frac{j}{2} \rceil]$ if at least one of i, j is even, and $\neg M[\frac{i+1}{2}, \frac{j+1}{2}]$ if i, j are both odd. Here, $\neg 1 = 0$, $\neg 0 = 1$, and $\neg \star = \star$. For example, for matrix

$$M = \begin{pmatrix} 0 & \star \\ \star & 1 \end{pmatrix}$$

the above construction results in

$$M' = \begin{pmatrix} 1 & 0 & \star & \star \\ 0 & 0 & \star & \star \\ \star & \star & 0 & 1 \\ \star & \star & 1 & 1 \end{pmatrix}.$$

Observe that for each non-$\star$ diagonal entry in M, two diagonal entries are created in M': one 1 and one 0.

It remains to show that an instance G to SUBGRAPH COMPLEMENT TO $\mathscr{G}$ is a yes-instance if and only if G belongs to $\mathscr{G}_{M'}$. We prove the two directions of the claim separately.

($\Longrightarrow$) Assume there is a subgraph complementation $G \oplus S$ into $\mathscr{G} = \mathscr{G}_M$. Let $X_1, X_2, \ldots, X_k$ be an M-partition of $G \oplus S$. We define partition $X'_1, X'_2, \ldots, X'_{2k}$ of G as follows: for every vertex $v \in X_i$, $1 \leq i \leq k$, we assign v to X'_{2i-1} if $v \in S$ and to X'_{2i} otherwise.

We now show that every edge of G respects the requirements of M'. Let $uv \in E(G)$ be an edge, and let $u \in X_i$ and $v \in X_j$. If at least one vertex from $\{u, v\}$, say v, is not in S, then uv is also an edge in $G \oplus S$, thus $M[i, j] \neq 0$. Since $v \notin S$, it belongs to set $v \in X'_{2j}$. Vertex u is assigned to set X'_ℓ, where ℓ is either $2i$ or $2i - 1$, depending whether u belongs to S or not. But because $2j$ is even irrespectively of ℓ, $M'[\ell, 2j] = M[i, j] \neq 0$.

Now consider the case when both $u,v \in S$. Then the edge does not persist after the subgraph complementation by S, and thus $M[i,j] \neq 1$. We further know that u is assigned to X'_{2i-1} and v to X'_{2j-1}. Both $2i-1$ and $2j-1$ are odd, and by the construction of M', we have that $M'[2i-1,2j-1] \neq 0$, and again the edge uv respects M'. An analogous argument shows that also all non-edges respect M'.

($\Longleftarrow$) Assume that there is a partition $X'_1, X'_2, \ldots, X'_{2k}$ of G according to M'. Let the set S consist of all vertices in odd-indexed parts of the partition. We now show that $G \oplus S$ can be partitioned according to M. We define partition $X_1, X_2, \ldots, X_k$ by assigning each vertex $u \in X'_i$ to $X_{\lceil \frac{i}{2} \rceil}$. It remains to show that $X_1, X_2, \ldots, X_k$ is an M-partition of $G \oplus S$.

Let $u \in X_i$, $v \in X_j$. Suppose first that $uv \in E(G \oplus S)$. If at least one of u,v is not in S, we assume without loss of generality that $v \notin S$. Then $uv \in E(G)$ and $v \in X'_{2j}$. For vertex $u \in X'_\ell$, irrespectively, whether ℓ is $2i$ or $2i-1$, we have that $M'[\ell,2j] = M[i,j] \neq 0$. Otherwise we have $u,v \in S$. Then uv is a non-edge in G, and thus $M'[2i-1,2j-1] \neq 1$. But by the construction of M', we have that $M[i,j] \neq 0$, and there is no violation of M. An analogous argument shows that if u and v are not adjacent in $G \oplus S$, it holds that $M[i,j] \neq 1$. Thus $X_1, X_2, \ldots, X_k$ is an M-partition of $G \oplus S$, which concludes the proof. □

Now we are ready to prove Theorem 9.

Proof of Theorem 9. For a given 2×2 matrix M, we use Lemma 11 to construct a matrix M'. Since M' has equally many 1's as 0's on its diagonal, it can not contain the 3-coloring matrix. Then by Proposition 10, the recognition of whether G admits M'-partition is in P. Thus, SUBGRAPH COMPLEMENT TO $\mathscr{G}$ is solvable in polynomial time. □

6 Subgraph complementation to graph classes of bounded clique-width

We show that SUBGRAPH COMPLEMENT TO $\mathscr{G}$ can be solved in polynomial time when $\mathscr{G}$ has bounded clique-width and can be expressed by an $\mathbf{MSO}_1$ property. We refer to the book [3] for the basic definitions. We will use the following result of Hliněný and Oum [13].

Proposition 12 ([13])**.** *There is an algorithm that for every integer k and graph G in time $O(|V(G)|^3)$ either computes a $(2^{k+1}-1)$ expression for a graph G or correctly concludes that the clique-width of G is more than k.*

Note that the algorithm of Hliněný and Oum only approximates the clique-width but does not provide an algorithm to construct an optimal k-expression tree for a graph G of clique-width at most k. But this approximation is usually sufficient for algorithmic purposes.

Courcelle, Makowsky and Rotics [4] proved that every graph property that can be expressed in $\mathbf{MSO}_1$ can be recognized in linear time for graphs of bounded clique-width when given a k-expression.

Proposition 13 ([4, Theorem 4])**.** *Let $\mathscr{G}$ be a class of graphs with clique-width bounded by constant k such that for each graph $G \in \mathscr{G}$, a corresponding k-expression can be found in $\mathscr{O}(f(|V(G)|,|E(G)|))$ time. Then every $\boldsymbol{MSO}_1$ property ϕ on $\mathscr{G}$ can be recognized in $\mathscr{O}(f(|V(G)|,|E(G)|) \cdot g(\phi,k)))$ time, for some function g.*

The nice property of graphs with bounded clique-width is that their subgraph complementation has also bounded clique-width. In particular, Kamiński, Lozin, and Milanič [17] observed that if G is a graph of clique-width k, then any subgraph complementation of G is of clique-width at most $g(k)$ for some computable function g. For completeness, we here provide a more accurate upper bound.

Lemma 14. *Let G be a graph, $S \subseteq V(G)$. Then* $\textsc{cwd}(G \oplus S) \leq 3\textsc{cwd}(G)$.

Proof. Let $\textsc{cwd}(G) = k$. To show the bound, it is more convenient to use expression trees instead of k-expressions. An *expression tree* of a graph G is a rooted tree T with nodes of four types i, $\dot\cup$, η and ρ such that:

- *Introduce nodes* $i(v)$ are leaves of T. For $i \in [k]$ and a vertex $v \in V(G)$, $i(v)$ is associated to the singleton graph where vertex v is labeled by i.
- *Union node* $\dot\cup$ stands for a disjoint union of k-graphs associated with its children.
- *Relabel node* $\rho_{i\to j}$ has one child and is associated with the k-graph obtained by applying of the relabeling operation to the graph corresponding to its child.
- *Join node* $\eta_{i,j}$ has one child and is associated with the k-graph resulting by applying the join operation on the graph represented by its child: every vertex labeled with i is made adjacent to every vertex labeled by j.
- The graph G is isomorphic to the graph associated with the root of T (with all labels removed).

The *width* of the tree T is the number of different labels appearing in T. If G is of clique-width k, then by parsing the corresponding k-expression, one can construct an expression tree of width k and, vise versa, given an expression tree of width k, it is straightforward to construct a k-expression. Throughout the proof we call the elements of $V(T)$ *nodes* to distinguish them from the vertices of G. Given a node x of an expression tree, T_x denotes the subtree of T rooted in x and the graph G_x represents the k-graph formed by T_x.

An expression tree T is *irredundant* if for any join node $\eta_{i,j}$, the vertices labeled by i and j are not adjacent in the graph associated with its child. It was shown by Courcelle and Olariu [5] that every expression tree T of G can be transformed into an irredundant expression tree T' of the same width in time linear in the size of T.

Let T be an irredundant expression tree of G of width k rooted in r. We construct the expression tree T' for $G' = G \oplus S$ by modifying T.

Recall that the vertices of the graphs G_x for $x \in V(T)$ are labeled $1,\dots,k$. We introduce three groups of distinct labels $\alpha_1,\dots,\alpha_k$, $\beta_1,\dots,\beta_k$ and $\gamma_1,\dots,\gamma_k$. The labels $\alpha_1,\dots,\alpha_k$ and $\beta_1,\dots,\beta_k$ correspond the the labels $1,\dots,k$ for the vertices in S and $V(G)\setminus S$ respectively. The labels $\gamma_1,\dots,\gamma_k$ are auxiliary. Then for every node x of T we construct T'_x using T_x starting the process from the leaves. We denote by G'_x the k-graph corresponding to the root x of T'_x.

For every introduce node $i(v)$, we construct an introduce node $\alpha_i(v)$ if $v \in S$ and an introduce node $\beta_i(v)$ if $v \notin S$. Let x be a non-leaf node of T and assume that we already constructed the modified expression trees of the children of x.

Let x be a union node $\dot\cup$ of T and let y and z be its children.

We construct k relabel nodes ρ_{α_i,γ_i} for $i \in \{1,\ldots,k\}$ that form a path, make one end-node of the path adjacent to y in T'_y and make the other end-node denoted by y' the root of $T'_{y'}$ constructed from T'_y. Notice that in the corresponding graph $G'_{y'}$ all the vertices of S are now labeled by $\gamma_1,\ldots,\gamma_k$ instead of $\alpha_1,\ldots,\alpha_k$.

Next, we construct a union node $\dot\cup$ denoted by $x^{(1)}$ with the children y' and z. This way we construct the disjoint union of $G'_{y'}$ and G'_z.

Notice that vertices which are labeled by the same label in G_y and G_z are not adjacent in G. In other words, we should make the vertices of $V(G_x)\cap S$ and $V(G_y)\cap S$ with the same label adjacent in G'. We achieve it by adding k join nodes η_{α_i,γ_i} for $i \in \{1,\ldots,k\}$, forming a path out of them and making one end-node of the path adjacent to $x^{(1)}$. We declare the other end-node of the path denoted by $x^{(2)}$ the new root.

Observe now that for the set of vertices Y_i of G_y labeled i and the set of vertices Z_j of G_z labeled by j where $i,j \in \{1,\ldots,k\}$ are distinct, it holds that the vertices of Y_i and Z_j are either pairwise adjacent in G or pairwise nonadjacent. More precisely, on this stage of construction we ensure that if the vertices of Y_i are not adjacent to the vertices of Z_j, then the vertices of $Y_i\cap S$ and $Z_j\cap S$ are made adjacent in G'. To do it, for every two distinct $i,j \in \{1,\ldots,k\}$ such that the vertices of Y_i and Z_j are not adjacent in G, construct a new join node η_{γ_i,α_j} and form a path with all these nodes whose one end-node is adjacent to $x^{(2)}$ and the other end-node $x^{(3)}$ is the new root (we assume that $x^{(3)} = x^{(2)}$ if have no new constructed nodes).

Finally, we add k relabel nodes ρ_{γ_i,α_i} for $i \in \{1,\ldots,k\}$ that form a path, make one end-node of the path adjacent to $x^{(3)}$ and make the other end-node denoted by x the root of the obtained T'_x. Clearly, all the vertices of S in G'_x are labeled by $\alpha_1,\ldots,\alpha_k$.

Let x be a relabel node $\rho_{i\to j}$ of T and let y be its child. We construct two relabel nodes $\rho_{\alpha_i\to\alpha_j}$ and $\rho_{\beta_i\to\beta_j}$ denoted by x and x' respectively. We make x' the child of x and we make the root y of T'_y the child of x'.

Now, let x be a join node $\eta_{i\to j}$ of T and let y be its child. Recall that T is irredundant, that is, the vertices labeled by i and j in G_y are not adjacent. Clearly, we should avoid making adjacent the vertices in S in the construction of G'. We do it by constructing three new join nodes $\eta_{\alpha_i\to\beta_j}$, $\eta_{\alpha_j\to\beta_i}$ and $\eta_{\beta_i\to\beta_j}$ denoted by x,x',x'' respectively. We make x' the child of x, x'' the child of x' and the node y of T'_y is made the child of x''.

This completes the description of the construction of T'. Using standard inductive arguments, it is straightforward to verify that G' is isomorphic to the graph associated with the root of T', that is, $\textsc{cwd}(G') \le 3k$. □

Lemma 15. *Let φ be an $\boldsymbol{MSO}_1$ property describing the graph class $\mathscr{G}$. Then there exists an $\boldsymbol{MSO}_1$ property ϕ describing the graph class $\mathscr{G}^{(1)}$ of size $|\phi| \in \mathcal{O}(|\varphi|)$.*

Proof. We will construct ϕ from φ in the following way: We start by prepending $\exists S \subseteq V(G)$. Then for each assessment of the existence of an edge in φ, say $uv \in E(G)$, replace that term with $((u \notin S \vee v \notin S) \wedge uv \in E(G)) \vee (u \in S \wedge v \in S \wedge uv \notin E(G))$. Symmetrically, for each assessment of the non-existence of an edge $uv \notin E(G)$, replace that term with $((u \notin S \vee v \notin S) \wedge uv \notin E(G)) \vee (u \in S \wedge v \in S \wedge uv \in E(G))$.

We observe that if φ is satisfiable for some graph G, then for every $S \subseteq V(G)$, the subgraph complementation $G \oplus S$ will yield a satisfying assignment to ϕ. Conversely, if ϕ is satisfiable for a graph G, then there exist some S such that φ is satisfied for $G \oplus S$. For the size, we note that each existence check for edges blows up by a constant factor. □

We are ready to prove the main result of this section.

Theorem 16. *Let $\mathscr{G}$ be a graph class expressible in **MSO**$_1$ which has clique-width bounded by constant k. Then* SUBGRAPH COMPLEMENT TO $\mathscr{G}$ *is solvable in polynomial time.*

Proof. Let φ be the $\mathbf{MSO}_1$ formula which describes $\mathscr{G}$, and let G be an n-vertex input graph. We apply Proposition 12 for G and in $O(n^3)$ time either obtain a $k' \leq 2^{3k+1}-1$ -expression for G or conclude that the clique-width of G is more than $3k$. In the latter case, by Lemma 14, G cannot be subgraph complemented to $\mathscr{G}$.

We then obtain an $\mathbf{MSO}_1$ formula ϕ from Lemma 15, and apply Proposition 13, which works in $\mathscr{O}(n^3 \cdot g(\phi,k'))$ time for some function g. As ϕ and k' depend only on constants φ and k, the runtime of the algorithm is $\mathscr{O}(n^3)$. □

We remark that if clique-width expression is provided along with the input graphs, and $\mathscr{G}$ can be expressed in $\mathbf{MSO}_1$, then there is a linear time algorithm for SUBGRAPH COMPLEMENT TO $\mathscr{G}$. This follows directly from Lemma 15 and Proposition 13.

Theorem 16 implies that for every class of graphs $\mathscr{G}$ of bounded clique-width characterized by a finite set of finite forbidden induced subgraphs, e. g. P_4-free graphs (also known as cographs) or classes of graphs discussed in [1], the SUBGRAPH COMPLEMENT TO $\mathscr{G}$ problem is solvable in polynomial time. However, Theorem 16 does not imply that SUBGRAPH COMPLEMENT TO $\mathscr{G}$ is solvable in polynomial time for $\mathscr{G}$ being of the class of graphs having clique-width at most k. This is because such a class $\mathscr{G}$ cannot be described by $\mathbf{MSO}_1$. Interestingly, for the related class $\mathscr{G}$ of graphs of bounded *rank-width* (see [5] for the definition) at most k, the result of Oum and Courcelle [6] combined with Theorem 16 implies that SUBGRAPH COMPLEMENT TO $\mathscr{G}$ is solvable in polynomial time.

7 Hardness of subgraph complementation to regular graphs

Let us remind that a graph G is regular if all its vertices have the same degree. We consider SUBGRAPH COMPLEMENT TO $\mathscr{G}$ where $\mathscr{G}$ is the class of regular graphs, which we call SUBGRAPH COMPLEMENT TO REGULAR GRAPH (SCR). In this section, we show that this problem is NP-complete by a reduction from CLIQUE IN REGULAR GRAPH.

CLIQUE IN REGULAR GRAPH (KR) **Input:** An undirected simple graph G which is regular, a positive integer k. **Question:** Does G contain a clique on k vertices?

We will need the following well-known proposition.

Proposition 17 ([10])**.** CLIQUE IN REGULAR GRAPH *is NP-complete.*

We remark that the above proposition follows because the INDEPENDENT SET problem is NP-complete for cubic graphs [10]; finding an independent set is equivalent to finding a clique in the complement graph, and the complement of a regular graph is also regular.

We are now ready to prove that CLIQUE IN REGULAR GRAPH is many-one reducible to SUBGRAPH COMPLEMENT TO REGULAR GRAPH. We begin by defining a gadget which we will use in the reduction.

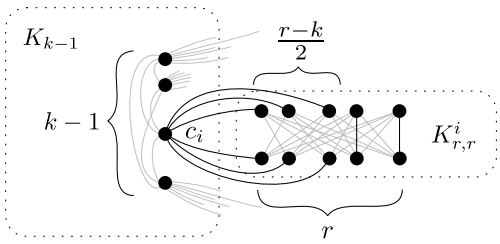


Figure 1: The gadget graph $H_{k,r}$ is built of k parts, namely a clique K_{k-1}, and $k-1$ complete bipartite graphs $K^1_{r,r},\ldots,K^{k-1}_{r,r}$ with some rewiring.

Definition 18 (Gadget $H_{k,r}$)**.** For integers $k \geq 2$ and $r \geq k$ such that $r-k$ is even, we build the gadget graph $H_{k,r}$ as follows. Initially, we let $H_{k,r}$ consist of one clique on $k-1$ vertices, as well as $k-1$ distinct copies of $K_{r,r}$. These are all the vertices of the gadget, which is a total of $(k-1)+2r\cdot(k-1)$ vertices. We denote the vertices of the clique $c_1,c_2,\ldots,c_{k-1}$, and we let the complete bipartite graphs be denoted by $K^1_{r,r},K^2_{r,r},\ldots,K^{k-1}_{r,r}$. For a bipartite graph $K^i_{r,r}$, let the vertices of the two parts be denoted by $a^i_1,a^i_2,\ldots,a^i_r$ and $b^i_1,b^i_2,\ldots,b^i_r$ respectively.

If $r=k$, then the construction is already complete. Otherwise, we will now do some rewiring of the edges to complete the construction of $H_{k,r}$. Recall that $r-k$ is even. For each vertex c_i of the clique, add one edge from c_i to each of $a^i_1,a^i_2,\ldots,a^i_{\frac{r-k}{2}}$. Similarly, add an edge from c_i to each of $b^i_1,b^i_2,\ldots,b^i_{\frac{r-k}{2}}$. Now remove the edges $a^i_1b^i_1,a^i_2b^i_2,\ldots,a^i_{\frac{r-k}{2}}b^i_{\frac{r-k}{2}}$. Once this is done for every $i\in[k-1]$, the construction is complete. See Figure 1.

We observe the following property of vertices a^i_j, b^i_j, and c_i of $H_{k,r}$.

Observation 19. *For every $i\in[k-1]$ and $j\in[r]$, it holds that the degrees of a^i_j and b^i_j in $H_{k,r}$ are both exactly r, whereas the degree of c_i is $r-2$.*

We now present our reduction in the form of an algorithm.

Algorithm 20 (Reduction KR to SCR)**.**
Input: An instance (G,k) of KR. Let r denote the regularity of G (the degree of its vertices).
Output: An instance G' of SCR such that it is a yes-instance if and only if (G,k) is a yes-instance of KR.

1. If $k<7$ or $k>r$, solve the instance of KR by brute force. If it is a yes-instance, return a trivial yes-instance to SCR, if it is a no-instance, return a trivial no-instance to SCR.

2. If $r-k$ is odd, modify G by taking two copies of G which are joined by a perfect matching between corresponding vertices. Then r increases by one, whereas k remains the same. Going forth, we assume G (and r) have already undergone this transformation if required.

3. Construct the graph G' by taking the disjoint union of G and the gadget $H_{k,r}$. Return G'.

Let $n=|V(G)|$. We observe that the number of vertices in the returned instance is at most $2n+(k-1)+2r\cdot(k-1)$, which is $\mathcal{O}(n^2)$. The running time of the algorithm is $\mathcal{O}(n^7)$ and thus is polynomial.

Correctness of the reduction follows from the following two lemmata.

Lemma 21. *Let (G,k) be the input of Algorithm 20, and let G' be the returned result. If (G,k) is a yes-instance to* CLIQUE IN REGULAR GRAPH*, then G' is a yes-instance of* SUBGRAPH COMPLEMENT TO REGULAR GRAPH.

Proof. Let $C \subseteq V(G)$ be a clique of size k in G. If the clique is found in step 1, then G' is a trivial yes-instance, so the claim holds. Thus, we can assume that the graph G' was constructed in step 3. If G was altered in step 2, we let C be the clique in one of the two copies that was created. Let $S \subseteq V(G')$ consist of the vertices of C as well as the vertices of the clique K_{k-1} of the gadget $H_{k,r}$. We claim that S is a valid solution to G'.

We show that $G' \oplus S$ is r-regular. Any vertex not in S will have the same number of neighbors as it had in G'. Since the only vertices that weren't originally of degree r were those in the gadget clique K_{k-1}, all vertices outside S also have degree r in $G' \oplus S$. What remains is to examine the degrees of vertices of C and of K_{k-1}.

Let c_i be a vertex of K_{k-1} in G'. Then c_i lost its $k-2$ neighbors from K_{k-1}, gained k neighbors from C, and kept $r-k$ neighbors in $K^i_{r,r}$. We see that its new neighborhood has size $k+r-k=r$.

Let $u \in C$ be a vertex of the clique from G. Then u lost $k-1$ neighbors from C, gained $k-1$ neighbors from K_{k-1}, and kept $r-(k-1)$ neighbors from $G-C$. In total, u will have $r-(k-1)+(k-1)=r$ neighbors in $G' \oplus S$. Since every vertex of $G' \oplus S$ has degree r, it is r-regular, and thus G' is a yes-instance. □

Lemma 22. *Let (G,k) be the input of Algorithm 20, and let G' be the returned result. If G' is a yes-instance to* SUBGRAPH COMPLEMENT TO REGULAR GRAPH*, then (G,k) is a yes-instance of* CLIQUE IN REGULAR GRAPH.

Proof. Let $S \subseteq V(G')$ be a solution witnessing that G' is a yes-instance. If G' was the trivial yes-instance returned in step 1 of Algorithm 20, the statement trivially holds. Going forward we may thus assume G' was returned in step 3, and that $k \geq 7$.

Our first claim is that $G' \oplus S$ is regular with regularity r. Assume for the sake of contradiction it is regular with regularity $r' \neq r$. Note that if r' is any other value than $r-2$, then every vertex of G' must be in S, as degrees of vertices outside S do not change under subgraph complementation; but then $G' \oplus S$ is simply the complement of G', and it is not regular. So r' must be $r-2$. Then every vertex except those in the gadget clique K_{k-1} must be in S. Consider a vertex in one of the complete bipartite graphs from the gadget — it will have at least n new incident edges (from the graph G). Since $n > r$, the vertex will not have $r-2$ neighbors in $G' \oplus S$, a contradiction.

We have established that $G' \oplus S$ is r-regular. It must thus be the case that every vertex of the gadget clique K_{k-1} is in S, since these vertices do not have degree r in G'.

Our second claim is that $|S| = 2k-1$, and moreover, that no neighbor of K_{k-1} is from S in G'. To show this, we let $p = |S \setminus K_{k-1}|$, and proceed to show that $p = k$. Towards this end, consider a vertex $c_i \in K_{k-1}$. This vertex has some number of neighbors in $S \setminus K_{k-1}$, denoted $x_i = |N_{G'}(c_i) \cap (S \setminus K_{k-1})|$. We know that c_i has r neighbors in $G' \oplus S$. Let us count them: some neighbors are preserved by the subgraph complementation, namely $r-k-x_i$ of its neighbors, found in $K^i_{r,r}$. Some neighbors are gained, namely $p-x_i$ of the vertices in S. Thus, we have that $r = r-k-x_i+p-x_i$. The r's cancel, and we get $x_i = \frac{p-k}{2}$. This is true for every $i \in [k-1]$, so we simply denote the number by $x = x_i$, and get $p = k+2x$.

Towards the claim, it remains to show that $x = 0$. Because the neighborhoods of distinct c_i and c_j are disjoint outside K_{k-1} in G', we get that $p \geq (k-1) \cdot x$. We substitute p, and get

$$k + 2x \geq (k-1) \cdot x$$

$$k \geq (k-3) \cdot x$$
$$\frac{k}{k-3} \geq x$$

Recalling that $k \geq 7$, we have that x is either 1 or 0. Assume for the sake of contradiction that $x = 1$. Then without loss of generality, each c_i has some neighbor a^i_j which is in S. Since a^i_j had degree r in G', it must hold that a^i_j has equally many neighbors as non-neighbors in S. At most one of a^i_j's neighbors in S is from $K^i_{r,r}$ in G', this means that at least $\frac{|S|-3}{2}$ vertices of $K^i_{r,r}$ are in S. Because $k \geq 7$ and the $K^i_{r,r}$'s are completely disjoint in G' for different values of $i \in [k-1]$, we get that

$$|S| \geq \frac{|S|-3}{2} \cdot (k-1) \geq \frac{|S|-3}{2} \cdot 6$$
$$|S| \geq 3 \cdot |S| - 9$$
$$9 \geq 2 \cdot |S|$$

Seeing that $|S| \geq k-1 \geq 6$, this is a contradiction. Thus, x must be 0, so $p = k+2x = k$ and the second claim holds.

We now show that $S \setminus K_{k-1}$ is a clique in G'. Assume for the sake of contradiction it is not, and let $u, v \in S \setminus K_{k-1}$ be vertices such that $uv \notin E(G')$. Consider the vertex u. By the second claim above we know that u does not have a neighbor from K_{k-1} in G'. It will thus gain at least k edges going to $K_{k-1} \cup \{v\}$, and lose at most $k-2$ edges going to $S \setminus (K_{k-1} \cup \{u,v\})$. Because u was of degree r in G' yet gained more edges than it lost by the subgraph complementation, its degree is strictly greater than r in $G' \oplus S$. This is a contradiction, hence $S \setminus K_{k-1}$ is a clique in G'.

Because $k \geq 3$, the clique $S \setminus K_{k-1}$ can not be contained in the gadget $H_{k,r}$ nor span across both copies of G created in step 2 of the reduction (if that step was applied). It must therefore be contained in the original G. Thus, G has a clique of size k, and (G,k) is a yes-instance of CLIQUE IN REGULAR GRAPH. □

Theorem 23. SUBGRAPH COMPLEMENT TO REGULAR GRAPH *is NP-complete.*

Proof. Lemmata 21 and 22 together with Proposition 17 conclude the proof of NP-hardness. Membership in NP is trivial, so NP-completeness holds. □

We remark that if the regularity r is a constant, the problem becomes polynomial time solvable by Theorem 7.

8 Conclusion and open problems

In this paper we initiated the study of SUBGRAPH COMPLEMENT TO $\mathscr{G}$. Many interesting questions remain open. In particular, what is the complexity of the problem when $\mathscr{G}$ is

- the class of chordal graphs,
- the class of interval graphs,
- the class of graph excluding a path P_5 as an induced subgraph,
- the class of graphs with min degree $\geq r$ for some constant r.

Another natural question is to consider a class $\mathscr{C}$, such that $\mathscr{C}^{(1)}$ can be recognized in polynomial time. Let $\overline{\mathscr{C}}$ denote the class of complement graphs to $\mathscr{C}$. Is it then also possible to recognize $\overline{\mathscr{C}}^{(1)}$ in polynomial time?

More broadly, it is also interesting to see what happens as we allow more than one subgraph complementation; how quickly can we recognize the class $\mathscr{G}^{(k)}$ for some class $\mathscr{G}$? It will also be interesting to investigate what happens if we combine subgraph complementation with other graph modifications, such as the Seidel switch.

Acknowledgment We thank Saket Saurabh for helpful discussions, and also a great thanks to the anonymous reviewers who provided valuable feedback.

References

[1] Alexandre Blanché, Konrad K. Dabrowski, Matthew Johnson, Vadim V. Lozin, Daniël Paulusma, and Viktor Zamaraev. Clique-width for graph classes closed under complementation. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, volume 83 of *LIPIcs*, pages 73:1–73:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[2] André Bouchet. Recognizing locally equivalent graphs. *Discrete Mathematics*, 114(1-3):75–86, 1993.

[3] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

[4] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

[5] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[6] Bruno Courcelle and Sang-il Oum. Vertex-minors, monadic second-order logic, and a conjecture by Seese. *J. Comb. Theory, Ser. B*, 97(1):91–126, 2007.

[7] Andrzej Ehrenfeucht, Jurriaan Hage, Tero Harju, and Grzegorz Rozenberg. Complexity issues in switching of graphs. In *Theory and Application of Graph Transformations, 6th International Workshop, TAGT'98, Paderborn, Germany, November 16-20, 1998, Selected Papers*, volume 1764 of *Lecture Notes in Computer Science*, pages 59–70. Springer, 2000.

[8] Tomás Feder, Pavol Hell, Sulamita Klein, and Rajeev Motwani. List partitions. *SIAM J. Discrete Math.*, 16(3):449–478, 2003.

[9] Fedor V. Fomin, Petr A. Golovach, Torstein J. F. Strømme, and Dimitrios M. Thilikos. Partial Complementation of Graphs. In David Eppstein, editor, *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, volume 101 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[10] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[11] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004.

[12] Peter L Hammer and Bruno Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981.

[13] Petr Hlinený and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008.

[14] Vít Jelínek, Eva Jelínková, and Jan Kratochvíl. On the hardness of switching to a small number of edges. In *Computing and Combinatorics - 22nd International Conference, COCOON 2016, Ho Chi Minh City, Vietnam, August 2-4, 2016, Proceedings*, volume 9797 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2016.

[15] Eva Jelínková and Jan Kratochvíl. On switching to H-free graphs. *Journal of Graph Theory*, 75(4):387–405, 2014.

[16] Eva Jelínková, Ondrej Suchý, Petr Hlinený, and Jan Kratochvíl. Parameterized problems related to Seidel's switching. *Discrete Mathematics & Theoretical Computer Science*, 13(2):19–44, 2011.

[17] Marcin Kaminski, Vadim V. Lozin, and Martin Milanic. Recent developments on graphs of bounded clique-width. *Discrete Applied Mathematics*, 157(12):2747–2761, 2009.

[18] Jan Kratochvíl. Complexity of hypergraph coloring and Seidel's switching. In *Graph-Theoretic Concepts in Computer Science, 29th International Workshop, WG 2003, Elspeet, The Netherlands, June 19-21, 2003, Revised Papers*, volume 2880 of *Lecture Notes in Computer Science*, pages 297–308. Springer, 2003.

[19] Jan Kratochvíl, Jaroslav Nešetřil, and Ondřej Zýka. On the computational complexity of Seidel's switching. In *Fourth Czechoslovakian Symposium on Combinatorics, Graphs and Complexity (Prachatice, 1990)*, volume 51 of *Ann. Discrete Math.*, pages 161–166. North-Holland, Amsterdam, 1992.

[20] Sang-il Oum. Rank-width and vertex-minors. *J. Comb. Theory, Ser. B*, 95(1):79–100, 2005.

[21] Sang-il Oum. Rank-width: Algorithmic and structural results. *Discrete Applied Mathematics*, 231:15–24, 2017.

[22] J. J. Seidel. Graphs and two-graphs. pages 125–143. Congressus Numerantium, No. X, 1974.

[23] J. J. Seidel. A survey of two-graphs. pages 481–511. Atti dei Convegni Lincei, No. 17, 1976.

[24] J. J. Seidel and D. E. Taylor. Two-graphs, a second survey. In *Algebraic methods in graph theory, Vol. I, II (Szeged, 1978)*, volume 25 of *Colloq. Math. Soc. János Bolyai*, pages 689–711. North-Holland, Amsterdam-New York, 1981.

Chapter 10

Time-inconsistent planning: simple motivation is hard to find

Fedor V. Fomin and Torstein J. F. Strømme. An extended abstract of this work was presented at the *34th AAAI conference*.

Time-inconsistent Planning: Simple Motivation Is Hard to Find

Fedor V. Fomin,[1] Torstein J. F. Strømme[1]

[1]University of Bergen, Norway

Abstract

People sometimes act differently when making decisions affecting the present moment versus decisions affecting the future only. This is referred to as time-inconsistent behavior, and can be modeled as agents exhibiting *present bias*. A resulting phenomenon is abandonment, which is when an agent initially pursues a task, but ultimately gives up before reaping the rewards.

With the introduction of the graph-theoretic *time-inconsistent planning model* due to Kleinberg and Oren, it has been possible to investigate the computational complexity of how a task designer best can support a present-biased agent in completing the task. In this paper, we study the complexity of finding a *choice reduction* for the agent; that is, how to remove edges and vertices from the task graph such that a present-biased agent will remain motivated to reach his target even for a limited reward. While this problem is NP-complete in general, this is not necessarily true for instances which occur in practice, or for solutions which are of interest to task designers. For instance, a task designer may desire to find the best task graph which is not too complicated.

We therefore investigate the problem of finding *simple* motivating subgraphs. These are structures where the agent will modify his plan at most k times along the way. We quantify this simplicity in the time-inconsistency model as a structural parameter: The number of branching vertices (vertices with out-degree at least 2) in a minimal motivating subgraph.

Our results are as follows: We give a linear algorithm for finding an optimal motivating path, i. e. when $k = 0$. On the negative side, we show that finding a simple motivating subgraph is NP-complete even if we allow only a single branching vertex — revealing that simple motivating subgraphs are indeed hard to find. However, we give a pseudo-polynomial algorithm for the case when k is fixed and edge weights are rationals, which might be a reasonable assumption in practice.

1 Introduction

Time-inconsistent behavior is a theme attracting great attention in behavioral economics and psychology. The field investigates questions such as why people let their bills go to debt collection, or buy gym memberships without actually using them. More generally, inconsistent behavior over time occurs when an agent makes a multi-phase plan, but does not follow through

on his initial intentions despite circumstances remaining essentially unchanged. Resulting phenomenons include procrastination and abandonment.

A common explanation for time-inconsistent behavior is the notion of *present bias*, which states that agents give undue salience to events that are close in time and/or space. This idea was described mathematically already in 1930's when (Samuelson 1937) introduced the discounted-utility model, which has since been refined in different versions (Laibson 1994).

George Akerlof describes in his lecture (Akerlof 1991) an even simpler mathematical model; here, the agent simply has a *salience factor* causing immediate events to be emphasized more than future ones. He goes on to show how even a very small salience factor in combination with many repeated decisions can lead to arbitrary large extra costs for the agent. This salience factor also has support from psychology, where (McClure et al. 2004) showed by using brain imaging that separate neural systems are in play when humans value immediate and delayed rewards.[1]

In 2014, (Kleinberg and Oren 2018) introduced a graph-theoretic model which elegantly captures the salience factor and scenarios of Akerlof. In this framework, where the agent is maneuvering through a weighted directed acyclic graph, it is possible to model many interesting situations. We will provide an example here.

1.1 Example

The student Bob is planning his studies. He considers taking a week-long course whose passing grade is a reward he quantifies to be worth $r = 59$. And indeed, Bob discovers that he can actually complete the course incurring costs and effort he quantifies to be only 46 — if he works evenly throughout the week (the upper path in Figure 1). Bob will reevaluate the cost every day, and as long as he perceives the cost to be at most equal to the reward, he will follow the path he finds to have the lowest cost.

The first day of studies incurs a cost of 6 for Bob due to some mandatory tasks he needs to do that day. But because Bob has a salience factor $b = 3$, he actually perceives the cost of that day's work to be 18, and of the course as a whole to be 58 ($18 + 10 + 10 + 10 + 10$). The reward

[1] We remark that quasi-hyperbolic discounting (discussed in (Laibson 1994; McClure et al. 2004)) can be seen as a generalization of both the discounted-continuity model (Samuelson 1937) and the salience factor (Akerlof 1991). There has been some empirical support for this model; however there are also many known psychological phenomena about time-inconsistent behavior it does not capture (Frederick, Loewenstein, and O'Donoghue 2002).

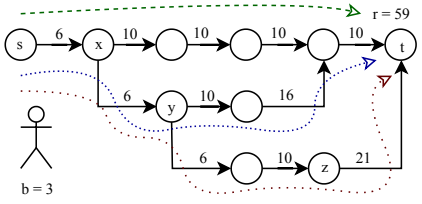


Figure 1: Acyclic digraph illustrating the ways in which Bob can distribute his efforts in order to complete the course. The upper path (green dashed line) is Bob's initial plan requiring the least total effort. The middle path (blue, narrowly dotted line) is the plan which appears better when at vertex x, and lower path (red, widely dotted line) is the plan he ultimately changes to at vertex y.

is even greater, though, so Bob persists to the next day.

When the second day of studies is about to start, Bob quasi-subconsciously makes the incorrect judgment that reducing his studies slightly now is the better strategy. He then changes his plan to the middle path in Figure 1. In terms of our model, the agent Bob standing at vertex x reevaluates the original plan (the upper path) to now cost $3 \cdot 10 + 10 + 10 + 10 = 60$, whereas the middle path is evaluated to only cost $3 \cdot 6 + 10 + 16 + 10 = 54$. He therefore chooses to go on with the plan that postpones some work to later, incurring a small extra cost to be paid at that time.

On the third day Bob finds himself at vertex y, and is yet again faced with a choice. The salience factor, as before, cause him to do less work in the present moment at the expense of more work in the future. He thus changes his plan to the lower path of Figure 1. However, it turns out that the choice was fatal — on the last day of the course (at vertex z), Bob is facing what he perceives to be a mountain of work so tall that it feels unjustified to complete the course; he evaluates the cost to be $3 \cdot 21 = 63$, strictly larger than the reward. He gives up and drops the course.

Because Bob abandons the task in our example above, we say that the graph in Figure 1 is not *motivating*. A natural question is to ask what we can do in order to make it so.

An easy solution for making a model motivating is to simply increase the reward. By simulating the process, it is also straightforward to calculate the minimum required reward to obtain this. However, it might be costly if we are the ones responsible for purchasing the reward, or even impossible if the reward is not for us to decide. A more appealing strategy might therefore be to allow the agent to only move around in a subgraph of the whole graph; for instance, if the lower path did not exist in our example above, then the graph would actually be motivating for Bob.[2] Finding such a subgraph is a form of *choice reduction*, and can be obtained by introducing a set of rules the agent must follow; for instance deadlines.

The aim of the current paper is not, however, to delve into the details of any particular scenario, but rather to investigate the formal underlying graph-theoretic framework. In the same spirit, (Kleinberg and Oren 2018) showed that the structure of a minimal motivating subgraph is actually quite restricted, and ask whether there is an efficient algorithm finding such subgraphs. Unfortunately, (Tang et al. 2017) and (Albers and Kraft 2019) independently proved that this problem is NP-complete in the general case. However, this does not exclude the existence of polynomial time algorithms for more restricted classes of graphs, or algorithms where the exponential blow-up occurs in parameters which in practical instances are small. This is what we investigate in the current paper; specifically, we look at restricting the number of *branching vertices* (vertices with out-degree at least 2) in a minimal motivating subgraph. This parameter can also be understood as the number of times a present-biased agent changes his plan.

Before we present our results, let us introduce the model more formally.

1.2 Formal model

We here present the model due to (Kleinberg and Oren 2018). Formally, an instance of the *time-inconsistent planning model* is a 6-tuple $M = (G, w, s, t, r, b)$ where:

[2]Removing edges and/or vertices that destroys the lower path is also the *only* option for how to make the example graph motivating; the upper path must be kept in its entirety, otherwise the agent will not be motivated to move from s to x; the middle path must also be kept in its entirety, otherwise the agent will give up when at x.

- $G = (V(G), E(G))$ is an acyclic digraph called a *task graph*. $V(G)$ is a set of elements called *vertices*, and $E(G) \subseteq V(G) \times V(G)$ is a set of directed *edges*. The graph is *acyclic*, which means that there exists an ordering of the vertices called a *topological order* such that, for each edge, its first endpoint comes strictly before its second endpoint in the ordering. Informally speaking, vertices represent states of intermediate progress, whereas edges represent possible actions that transitions an agent between states.
- $w : E(G) \to \mathbb{R}_{\geq 0}$ is a function assigning non-negative weight to each edge. Informally speaking, this is the cost incurred by performing a certain action.
- $s \in V(G)$ is the start vertex.
- $t \in V(G)$ is the target vertex.
- $r \in \mathbb{R}_{\geq 0}$ is the reward.
- $b \in \mathbb{R}_{\geq 1}$ is the agent's salience factor.[3]

An agent with salience factor b is initially at vertex s and can move in the graph along edges in their designated direction. The agent's task is to reach the target t, at which point the agent can pick up a reward worth r. We can usually assume that there is at least one path from s to each vertex, and at least one path from each vertex to t, as otherwise these vertices are of no interest to the agent.

When standing at a vertex u, the agent evaluates (with a present bias) all possible paths from u to t. In particular, a u-t path $P \subseteq G$ with edges $e_1, e_2, \ldots, e_p$ is evaluated by the agent standing at u to cost $\zeta_M(P) = b \cdot w(e_1) + \sum_{i=2}^{p} w(e_i)$. We refer to this as the *perceived* cost of the path. For a vertex u, its perceived cost to the target is the minimum perceived cost of any path to t, $\zeta_M(u) = \min\{\zeta_M(P) \mid P \text{ is a } u\text{-}t \text{ path}\}$. If the perceived cost from vertex u to the target is strictly larger than the reward, $\zeta_M(u) > r$, then an agent standing there *abandons* the task. Otherwise, he will (non-deterministically) pick one of the paths which minimize perceived cost of reaching t, and traverse its first edge. This repeats until the agent either reach t or abandons the task.

If every possible route chosen by the agent will lead him to t, then we say that the model instance is *motivating*. If the model instance is clear from the context, we take that the *graph* is motivating to mean the same thing, and we may drop the subscript $_M$ in the notation.

Definition 1 (Motivating subgraph)**.** *If G' is a subgraph of G belonging to a time-inconsistent planning model $M = (G, w, s, t, r, b)$, then we call G' a* motivating subgraph *if G' contains s and t and $M' = (G', w|_{E(G')}, s, t, r, b)$ is motivating.*

In the current paper, we investigate the problem of finding a simple motivating subgraph. In order to quantify what we mean by *simple*, we first provide the definition of a *branching vertex*:

Definition 2 (Branching vertex)**.** *The* out-degree *of a vertex u in a digraph G is the number of edges in G that have u as its first endpoint. We say that u is a* branching vertex*, if its out-degree is at least two.*

[3] While we in the current paper use b for the salience factor as introduced in (Kleinberg and Oren 2018), the literature about time-inconsistent planning commonly use the term $\beta = b^{-1}$ instead, which (while slightly more convoluted to work with for our purposes) seamlessly integrates with the quasi-hyperbolic discounting model. An artifact of our current definition is that the reward is not scaled by b when the agent is one leg away; however, both algorithms and hardness proofs can be adapted to account for this technical difference.

A_φ	For a set A and a constraint $\varphi : A \to \{\mathtt{T},\mathtt{F}\}$, the set A *constrained to* φ denotes the elements of A that satisfy φ, i. e. $\{a \in A \mid \varphi(a) = \mathtt{T}\}$. For example, $\mathbb{Z}_{\geq 0}$ indicates the set of all non-negative integers.
$[x]$	For $x \in \mathbb{Z}_{\geq 1}$, $[x]$ is the set $\{1,2,\ldots,x\}$.
$\subseteq$	For sets, $\subseteq$ is the standard subset notation. For graphs G and H, H is a *subgraph* of G, denoted $H \subseteq G$, if $V(H) \subseteq V(G)$, and $E(H) \subseteq E(G)$.
$f\|_A$	For a function $f : B \to C$ and a set $A \subseteq B$, the function f *restricted to* A is a function $f\|_A : A \to C$ such that for every $a \in A$, $f\|_A(a) = f(a)$.
$\mathrm{C}_G(u,v)$	For a graph G and vertices $u,v \in V(G)$, the (true) *cost* from u to v is the minimum sum of weights for a path from u to v in G.
$G[A]$	For a directed graph G and vertex set $A \subseteq V(G)$, the *induced subgraph* $G[A]$ is the graph where $V(G[A]) = A$ and $E(G[A]) = E(G) \cap A \times A$.

Table 1: Summary of notation.

In its most general form, we will investigate the following problem:

SIMPLE MOTIVATING SUBGRAPH
Input: A time-inconsistent planning model $M = (G, w\, s, t, r, b)$, and a non-negative integer $k \in \mathbb{Z}_{\geq 0}$.
Question: Does there exist a motivating subgraph $G' \subseteq G$ with at most k branching vertices?

1.3 Previous work

Time-inconsistent behavior is a field with a long history in behavioral economics, see (Akerlof 1991; Frederick, Loewenstein, and O'Donoghue 2002; O'Donoghue and Rabin 1999). The model used in this paper was introduced by (Kleinberg and Oren 2018), and they also give structural results concerning how much extra cost the salience factor can incur for an agent, and how many values the salience factor can take which will lead the agent to follow distinct paths. They raise the issue of motivating subgraphs, and give a characterization of the minimal among them which we will see later.

(Tang et al. 2017) refine the structural results concerning extra costs caused by present bias. Furthermore, they show that finding motivating subgraphs is NP-complete in the general case by a reduction from 3-SAT. They also investigate a few variations of the problem where intermediate rewards can be placed on vertices.

(Albers and Kraft 2019) independently show that finding motivating subgraphs is NP-complete, by a reduction from the ℓ-LINKAGE problem in acyclic digraphs. Furthermore, they show that the approximation version of the problem (finding the smallest r such that a motivating subgraph exists) cannot be approximated in polynomial time to a ratio of $\sqrt{n}/3$ unless P = NP; but a $1+\sqrt{n}$ -approximation algorithm exists. They also explore another variation of the problem with intermediate rewards.

There have been work on variations on the model and problem where the designer is free to raise edge costs (Albers and Kraft 2017a), where the agents are more sophisticated (Kleinberg, Oren, and Raghavan 2016), exhibit multiple biases simultaneously (Kleinberg, Oren, and Raghavan 2017), or where the salience factor varies (Albers and Kraft 2017b; Gravin et al. 2016).

1.4 Our contribution

We prove two main results about the complexity of SIMPLE MOTIVATING SUBGRAPH. First, we show that the problem is solvable in linear time when $k = 0$, and is NP-complete otherwise. Secondly, we show that when the costs are bounded by some polynomial in the size of the task graph, then it is solvable in polynomial time for every fixed k. More precisely, we show the following.

Theorem 3. SIMPLE MOTIVATING SUBGRAPH *is solvable in polynomial time for $k = 0$, and is NP-complete for any $k \geq 1$.*

The reduction we use to prove NP-completeness of the problem in Theorem 3 is from SUBSET SUM, which is weakly NP-complete. Indeed, our hardness reduction strongly exploits constructions with exponentially large (or small) edge weights; the parameter W in our reduction — the sum of all edge weights when scaled to integer values — is exponential in the number of vertices in the graph. A natural question is hence whether SIMPLE MOTIVATING SUBGRAPH can be solved by a pseudo-polynomial algorithm, i. e. an algorithm which runs in polynomial time when W is bounded by some polynomial of the size of the graph. Unfortunately, this is highly unlikely. A closer look at the NP-hardness proof of (Tang et al. 2017) for finding a motivating subgraph, reveals that instances created in the reduction from 3-SAT have weights that depend only on the constant b.

On the other hand, in the reduction of (Tang et al. 2017) the number of branchings in their potential solutions grows linearly with the number of clauses in the 3-SAT instance. This leaves a possibility that when W is bounded by a polynomial of the size of the input graph G and k is a constant, then SIMPLE MOTIVATING SUBGRAPH is solvable in polynomial time. Theorem 4 confirms that this is exactly the case. (In this theorem we assume that all edge weights are integers — but since scaling the reward and all edge weights by a common constant is trivially allowed, it also works for rationals.)

Theorem 4. SIMPLE MOTIVATING SUBGRAPH *is solvable in time $(|V(G)| \cdot W)^{\mathcal{O}(k)}$ whenever all edge weights are integers and W is the sum of all weights.*

Theorem 4 naturally leads to another question, whether SIMPLE MOTIVATING SUBGRAPH is solvable in time $(|V(G)| \cdot W)^{\mathcal{O}(1)} \cdot f(k)$ for some function f of k only. Or in other words, whether the problem is fixed-parameter tractable parameterized by k when W is encoded unary? We observe that the hardness proof of (Albers and Kraft 2019), reducing from the ℓ-LINKAGE problem in acyclic digraphs, combined with hardness result for the ℓ-LINKAGE problem by (Slivkins 2010), implies the following theorem.

Theorem 5. *Unless* FPT = W[1]*, there is no algorithm solving* SIMPLE MOTIVATING SUBGRAPH *in time $(|V(G)| \cdot W)^{\mathcal{O}(1)} \cdot f(k)$ for any function f of k only.*

2 A dichotomy (proof of Theorem 3)

In this section we prove Theorem 3. Recall that the theorem states that SIMPLE MOTIVATING SUBGRAPH is solvable in polynomial time for $k = 0$, and is NP-complete for any $k \geq 1$. We split the proof into two subsections. The first subsection contains a polynomial time algorithm solving SIMPLE MOTIVATING SUBGRAPH for $k = 0$ (Lemma 7). The second subsection proves that the problem is NP-complete for every $k \geq 1$ (Lemma 8).

2.1 A linear-time algorithm for motivating paths

Any connected graph without branching vertices is a path. Thus, we refer to the variant of SIMPLE MOTIVATING SUBGRAPH with $k = 0$ as to the MOTIVATING PATH problem. This algorithm is essentially a variation on the classical linear time algorithm computing a shortest path in an acyclic digraph.

We give an algorithm which solves the MOTIVATING PATH problem in $\mathcal{O}(|V(G)| + |E(G)|)$ time. In fact, our algorithm will solve the problem of finding the minimum cost such path, if one exists.

Algorithm 6: MOTIVATING PATH

Input: An instance of MOTIVATING PATH
Output: The cost of a cheapest motivating s-t path witnessing a yes-instance; or ∞ if no motivating path exists.

For each vertex $u \in V(G)$, let $d_u \leftarrow \infty$
For the target t, let $d_t \leftarrow 0$
for *each vertex* $u \in V(G)$ *in rev. topological order* **do**
  for *each out-neighbor* $v \in N(u)$ **do**
    if $b \cdot w(uv) + d_v \leq r$ **and** $w(uv) + d_v < d_u$ **then**
      $d_u \leftarrow w(uv) + d_v$
    end
  end
end
return d_s

Lemma 7. *The* MOTIVATING PATH *problem can be solved in linear time.*

Proof. We prove that Algorithm 6 is correct. We assume that every vertex has a path to t in G (otherwise we can simply remove it), hence t will come last in the topological order. For every $u \in V(G)$, we claim that d_u holds the minimum cost of a motivating path from u to t. We observe that our base, $u = t$, is correct since $d_t = 0$.

Consider some vertex u. Because the vertices are visited in reverse topological order, all out-neighbors of u are already processed, and hold by the induction hypothesis the correct value. An agent standing at vertex u is motivated to move to the next vertex v in a path P if $b \cdot w(uv) + \mathrm{C}_P(v,t) \leq r$. Hence, if the condition holds, prepending a motivating path from v to t with u will also yield a motivating path. By choosing the minimum total cost among all feasible candidates for the next step, the final value is in accordance with our claim.

Finally, we observe that the runtime is correct. Assuming an adjacency list representation of the graph, a topological sort can be done in linear time. The algorithm then process each vertex once and touches each edge once. □

We remark that the graph produced by Algorithm 6 is a $(1 + \sqrt{n})$ -approximation to the general motivating subgraph problem. This follows because the approximation algorithm of (Albers and Kraft 2019) with the stated approximation ratio always produce a path; Algorithm 6 will on the other hand find the optimal path, and is hence at least as good.

2.2 Hardness of allowing branching vertices

Deciding whether there exists a *path* which will motivate a biased agent to reach the target turns out to be easy, but we already know by the results of (Tang et al. 2017) and (Albers and Kraft 2019) that finding a motivating *subgraph* in general is NP-hard. Both aforementioned results give hardness reductions where the feasible solutions to the reduced instance have a "complicated" structure in the sense that they contain many branching vertices. A natural question is whether it could be easier to find motivating subgraphs whose structure is simpler, as in the case of the path.

A first question might be whether SIMPLE MOTIVATING SUBGRAPH is *fixed parameter tractable* (FPT) parameterized by k, the number of branching vertices. Unfortunately, this is already ruled out by the reduction of Albers and Kraft, since they reduce from the W[1]-hard ℓ-LINKAGE problem for acyclic digraphs — the number of branchings in their feasible solutions is linear in ℓ.

As the ℓ-LINKAGE problem can be solved in time $n^{f(\ell)}$ for acyclic digraphs (Bang-Jensen and Gutin 2008), their reduction does not rule out an XP-algorithm. In this section we show that SIMPLE MOTIVATING SUBGRAPH is actually NP-hard even for $k = 1$, and is thus an even harder problem.

We show this by a reduction from the NP-hard SUBSET SUM problem. The reduction is provided in the form of Algorithm 9, whereby someone who wants to solve SUBSET SUM can give their problem instance as input, and receive a SIMPLE MOTIVATING SUBGRAPH-instance with $k = 1$ as output. This can then be piped into an (imaginary) algorithm which solves SIMPLE MOTIVATING SUBGRAPH efficiently for $k = 1$. If said imaginary algorithm runs in polynomial time, this implies P=NP.

Lemma 8. *The* SIMPLE MOTIVATING SUBGRAPH *problem is NP-complete for every* $k \geq 1$.

Proof. Deciding whether a given graph is motivating can be done in polynomial time by simply checking whether it is possible for the agent to reach a vertex where the perceived cost is greater than the reward. This implies the membership of SIMPLE MOTIVATING SUBGRAPH in NP.

To prove NP-completeness, we reduce from the classical NP-complete problem SUBSET SUM (Karp 1972).

SUBSET SUM
Input: A set of integers $X = \{x_1, x_2, \ldots, x_n\} \subseteq \mathbb{Z}_{\geq 0}$ and a target $W \in \mathbb{Z}_{\geq 0}$.
Question: Does there exists a subset $X' \subseteq X$ such that its elements sums to W?

The reduction is described in the form of Algorithm 9 and an example is given in Figure 2. Soundness of the reduction is proved in the following claim.

Claim 10. *Algorithm 9 is safe. Given as input an instance I of* SUBSET SUM *and a salience factor $b \in \mathbb{R}_{>0}$, the output instance I' of* SIMPLE MOTIVATING SUBGRAPH *is a yes-instance if and only if I is a yes-instance.*

Proof of claim. Before we begin the proof, we will adapt the following notation: For a vertex u and subgraph $H \subseteq G$ containing at least one path from u to t, we let $\zeta_H(u)$ denote the perceived cost from vertex u to t in H. In other words, $\zeta_H(u) = \min\{\zeta(P) \mid P \subseteq H \text{ is a } u\text{-}t \text{ path}\}$.

For the forward direction of the proof, assume that I is a yes-instance and let $X' \subseteq X$ be a witness to this. Let $G' \subseteq G$ be the graph where for each $i \in [n]$ the vertex c_i^* and incident edges

Algorithm 9: Reduction from SUBSET SUM to SIMPLE MOTIVATING SUBGRAPH ($k = 1$)

Input: An instance $I = (X = \{x_1, x_2, \ldots, x_n\}, W)$ of SUBSET SUM; and any salience factor $b \in \mathbb{R}_{>1}$.

Output: An instance $I' = (G, w, s, t, b, r, k)$ of SIMPLE MOTIVATING SUBGRAPH with $k = 1$ and salience factor b.

$V(G) \leftarrow \{s, a_0, a_1, a_2, a_3, t\} \cup \{c_1, c_1^*, c_2, c_2^*, \ldots, c_n, c_n^*\} \cup \{c_{n+1}, c_{n+2}\}$

$E(G) \leftarrow \{sa_0, a_0a_1, a_1a_2, a_2a_3, a_3t\} \cup \{c_ic_i^*, c_ic_{i+1}, c_i^*c_{i+1} \mid i \in [n]\} \cup \{a_0c_1, c_{n+1}c_{n+2}, c_{n+2}t\}$

$$w \leftarrow \left\{ \begin{array}{rcl} a_3t & \mapsto & \frac{1}{b} \\ a_2a_3 & \mapsto & \frac{1-w(a_3t)}{b} \\ a_1a_2 & \mapsto & \frac{1-w(a_2a_3)-w(a_3t)}{b} \\ a_0a_1 & \mapsto & \frac{1-w(a_1a_2)-w(a_2a_3)-w(a_3t)}{b} \\ sa_0 & \mapsto & \frac{1-(\sum_{i=0}^{2} w(a_ia_{i+1}))-w(a_3t)}{b} + \frac{\varepsilon}{b} \\ c_{n+2}t & \mapsto & w(a_3t) + \varepsilon \\ c_{n+1}c_{n+2} & \mapsto & w(a_2a_3) - 2\varepsilon - \frac{2\varepsilon}{b-1} \\ c_ic_{i+1} & \mapsto & \frac{x_i \cdot w(a_1a_2)}{W} \quad \text{for } i \in [n] \\ a_0c_1 & \mapsto & w(a_0a_1) + \frac{2\varepsilon}{b-1} \\ & \mapsto & 0 \quad \text{otherwise} \end{array} \right\}$$

return $(G, w, s, t, b, r = 1, k = 1)$

are removed if $x_i \in X'$, and the edge c_ic_{i+1} is removed if $x_i \notin X'$. We make a series of step-wise observations which shows that G' is motivating:

1. G' contains a single vertex with out-degree at least 2, namely a_0. There are two possible paths from s to t:

   - the a-path $P_a = [s, a_0, a_1, a_2, a_3, t]$, and
   - the c-path $P_c = [s, a_0, c_1, (c_1^*), c_2, (c_2^*), \ldots, c_n, (c_n^*), c_{n+1}, c_{n+2}, t]$ (where for each $i \in [n]$, P_c only include vertex c_i^* when $x_i \notin X'$).

2. In the path P_a, the agent will by construction perceive the cost of moving towards t to be 1, regardless of which vertex he resides on. The only exception to this is when the agent is at s; then the perceived cost of following the path P_a is $1 + \varepsilon$. In other words, $\zeta(P_a) = 1 + \varepsilon$ and for each of $i \in \{0, 1, 2, 3\}$, $\zeta_{P_a}(a_i) = 1$.

3. The cost from c_1 to c_{n+1} in G' is $\sum_{x_i \in X'} \frac{x_i \cdot w(a_1a_2)}{W}$, which simplifies to exactly $w(a_1a_2)$. This holds because X' sums to W by the initial assumption. In other words, $\mathrm{C}_{P_c}(c_1, c_{n+1}) = w(a_1a_2)$.

4. The cost from a_0 to t is ε shorter in the c-path compared to the a-path, $\mathrm{C}_{P_c}(a_0, t) = \mathrm{C}_{P_a}(a_0, t) - \varepsilon$. This follows from (3) and how the remaining weights in the c-path are defined.

5. The perceived cost from s to t in P_c is 1, $\zeta(P_c) = 1$ (follows by combining (2) and (4), and observing that the two paths share their first leg). The agent is hence motivated to move from s to a_0 with a plan of following P_c towards the target.

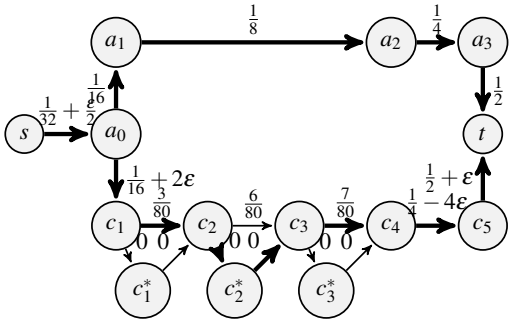


Figure 2: The graph G constructed by Algorithm 9 on input $X = \{3,6,7\}, W = 10, b = 2$. The solution to SUBSET SUM is $X' = \{3,7\}$ and the corresponding motivating subgraph G' is formed by thick arcs. In graph G the agent is tempted to pursue the lower path $P_c = [s, a_0, c_1, c_1^*, c_2, c_2^*, c_3, c_3^*, c_4, c_5, t]$ but will lose motivation at node c_5. In graph G', at node s, the agent's plan will be to follow the lower path P_c' but at node a_0 the agent will switch to the upper path P_a. At every step on this path the agent is motivated to reach t.

6. The perceived cost from a_0 to t in P_c is ε more than the perceived cost from a_0 to t in P_a. This follows from the following chain of substitutions:

$$\begin{aligned}
\zeta_{P_c}(a_0) &= b \cdot w(a_0c_1) + \mathrm{C}_{P_c}(c_1, t) \\
&= b \cdot w(a_0a_1) + b \cdot \frac{2\varepsilon}{b-1} + \mathrm{C}_{P_c}(c_1, c_{n+1}) + w(c_{n+1}c_{n+2}) + w(c_{n+2}t) \\
&= b \cdot w(a_0a_1) + b \cdot \frac{2\varepsilon}{b-1} + w(a_1a_2) + w(a_2a_3) - 2\varepsilon - \frac{2\varepsilon}{b-1} + w(a_3t) + \varepsilon \\
&= b \cdot w(a_0a_1) + w(a_1a_2) + w(a_2a_3) + w(a_3t) + (b-1) \cdot \frac{2\varepsilon}{b-1} - \varepsilon \\
&= b \cdot w(a_0a_1) + w(a_1a_2) + w(a_2a_3) + w(a_3t) + \varepsilon \\
&= b \cdot w(a_0a_1) + \mathrm{C}_{P_a}(a_1, t) + \varepsilon \\
&= \zeta_{P_a}(a_0) + \varepsilon
\end{aligned}$$

It follows from (6) that an agent standing at a_0 is *not* willing to walk along P_c, but change his plan to walking along P_a instead. The agent will stay motivated on the a-path, and will reach the reward (2). Hence, the graph G' is motivating.

For the reverse direction of the proof, assume there is a motivating subgraph $G' \subseteq G$ that motivates the agent to reach t. We notice that G' must contain the a-path, since any agent moving out of the a-path will by the construction of G' need to walk past c_{n+2} — however, an agent standing at this vertex will by the construction always give up. We also notice that P_a itself is not motivating, hence G' must contain at least one other path from s to t. Let P_c denote the *cheapest* s-t path that is different from P_a.

We begin by observing that P_c must start at s, include a_0 and c_1,, a path from c_i to c_{i+1} for every $i \in [n]$, as well as the vertices c_{n+1}, c_{n+2}, and t.

There are some cost requirements that P_c needs to fulfill in order for G' to be motivating. In order to motivate an agent at s to move to a_0, the cost from a_0 to t in P_c can be at most $\mathrm{C}_{P_a}(a_0, t) - \varepsilon$. This implies that $\mathrm{C}_{P_c}(c_1, c_{n+1}) \leq w(a_1a_2)$.

However, P_c can not have so low cost that it tempts the agent to move off the a-path. In particular, an agent standing at a_0 must perceive the a-path to be strictly cheaper than walking along P_c. We obtain the following inequality:

$$\begin{aligned}
\zeta_{P_a}(a_0) &< \zeta_{P_c}(a_0) \\
b \cdot w(a_0a_1) + \mathrm{C}_{P_a}(a_1,t) &< b \cdot w(a_0c_1) + \mathrm{C}_{P_c}(c_1,t) \\
\mathrm{C}_{P_a}(a_1,t) &< b \cdot \frac{2\varepsilon}{b-1} + \mathrm{C}_{P_c}(c_1,t) \\
w(a_1a_2) + w(a_2a_3) + w(a_3t) &< b \cdot \frac{2\varepsilon}{b-1} + \mathrm{C}_{P_c}(c_1,c_{n+1}) + w(a_2a_3) - 2\varepsilon - \frac{2\varepsilon}{b-1} + w(a_3t) + \varepsilon \\
w(a_1a_2) &< (b-1) \cdot \frac{2\varepsilon}{b-1} + \mathrm{C}_{P_c}(c_1,c_{n+1}) - \varepsilon \\
w(a_1a_2) - \varepsilon &< \mathrm{C}_{P_c}(c_1,c_{n+1})
\end{aligned}$$

We now construct a solution $X' \subseteq X$ to SUBSET SUM by including x_i in X' if P_c use the edge c_ic_{i+1}. Notice that the sum of edge weights for these edges will make up the cost $\mathrm{C}_{P_c}(c_1,c_{n+1})$. We can lift the bounds for that cost to the sum of elements in X':

$$\begin{aligned}
w(a_1a_2) - \varepsilon < \mathrm{C}_{P_c}(c_1,c_{n+1}) &\leq w(a_1a_2) \\
w(a_1a_2) - \varepsilon < \sum_{x_i \in X'} \frac{x_i \cdot w(a_1a_2)}{W} &\leq w(a_1a_2) \\
W - \varepsilon \cdot \frac{W}{w(a_1a_2)} < \sum_{x_i \in P_c} x_i &\leq W
\end{aligned}$$

By choosing ε strictly smaller than $\frac{w(a_1a_2)}{W}$, we guarantee that the set X' has value exactly W (note: this bound on ε is a function of b and W). This concludes the proof of the soundness of the reduction. ◇

By the claim, SIMPLE MOTIVATING SUBGRAPH is NP-complete for $k = 1$ and hence is also for every $k \geq 1$. □

3 A pseudo-polynomial algorithm (proof of Theorem 4)

The reduction in the previous section shows that restricting the number of branchings in the motivating structures we look for does not make the task of finding them significantly easier. However, we notice that the weights used in the graph constructed in the reduction can be exponentially small, and depend on W as well as on b. On the other hand, in the hardness proof for MOTIVATING SUBGRAPH by (Tang et al. 2017), the instances created in the reduction from 3-SAT have weights that depend only on the constant b; but the number of branchings in their potential solutions grows linearly with the number of clauses in the 3-SAT instance.

Hence, MOTIVATING SUBGRAPH is hard even if the number of branchings in the solution we are looking for is bounded, and it is also hard if the input instance only use integer weights bounded by a constant. But what if we impose both restrictions simultaneously? In this section

we prove that if the input instance has bounded integer weights, then we can quickly determine whether it contains a motivating subgraph with few branchings.

We will make a use of an auxiliary problem which is a variation of the exact ℓ-LINKAGE problem in acyclic digraphs, but which also impose restrictions on the links — requiring them to be motivating for biased agents. We define the problem:

EXACT MOTIVATING k-LINKAGE IN DAG (EMKL)
Input: An acyclic digraph G; edge weights $w : E(G) \to \mathbb{Z}_{\geq 0}$ s. t. $\sum_{e \in E(G)} w(e) = W$; sources $s_1, s_2, \ldots, s_k \in V(G)$; sinks $t_1, t_2, \ldots, t_k \in V(G)$; target link weights $\ell_1, \ell_2, \ldots, \ell_k \in Z_{\geq 0}$; salience factors $b_1, b_2, \ldots, b_k \in \mathbb{R}_{\geq 1}$; and rewards $r_1, r_2, \ldots, r_k \in \mathbb{R}_{\geq 0}$.
Question: Does there exist (internally) vertex disjoint paths $P_1, P_2, \ldots, P_k$ such that for each $i \in [k]$, P_i starts in s_i, ends in t_i, has weight ℓ_i, and is such that an agent with salience factor b_i will be motivated by a reward r_i to move from s_i to t_i?

We solve the EXACT MOTIVATING k-LINKAGE IN DAG problem using dynamic programming, inspired by the solution of (Fortune, Hopcroft, and Wyllie 1980) for the ℓ-LINKAGE problem in acyclic digraphs (see also (Bang-Jensen and Gutin 2008)).

Lemma 11. EXACT MOTIVATING k-LINKAGE IN DAG *can be solved in time* $\mathcal{O}(kn^{k+1}W^k)$.

Proof. We will in the upcoming proof assume all sources and sinks are distinct. If they are not, simply make multiple copies by splitting each such vertex, such that there is one for each occurrence as a source or sink.

We solve the problem by dynamic programming using a Boolean table dp of size $\mathcal{O}(n^k W^k)$. The table is indexed by vertices $u_i \in V(G)$ and weights $d_i \in [W]$ for $i \in [k]$. We define a cell:

$dp[u_1, u_2, \ldots, u_k, d_1, d_2, \ldots, d_k] :=$ `TRUE` if there exists vertex disjoint paths $P_1, P_2, \ldots, P_k$ such that for each $i \in [k]$ the following holds:
- P_i starts in u_i and ends in t_i, and
- P_i has weight d_i, and
- an agent with salience factor b_i is motivated by a reward r_i to move from u_i to t_i in P_i.

`FALSE` otherwise.

Observe that the final answer to the EMKL instance will by this definition be found in $dp[s_1, s_2, \ldots, s_k, \ell_1, \ell_2, \ldots, \ell_k]$. We proceed to establish the base case of our recurrence, when the vertices align perfectly with the sinks. To comply with the definition, the entry is `TRUE` if the required distances are 0, and `FALSE` otherwise.

$$dp[u_1, u_2, \ldots, u_k, d_1, d_2, \ldots, d_k] \leftarrow \begin{cases} \texttt{TRUE} & \text{if for every } i \in [k], u_i = t_i \text{ and } d_i = 0. \\ \texttt{FALSE} & \text{if for every } i \in [k], u_i = t_i \text{ and } d_i \neq 0. \end{cases}$$

We move on to describe the recurrence. The idea is to move one step forward, trying every neighbor of each vertex in the current state, to see whether it is possible to find a slightly shorter partial solution we can extend.

$dp[u_1, u_2, \ldots, u_k, d_1, d_2, \ldots, d_k] \leftarrow$ `TRUE` if there exists $i \in [k]$ and $v \in N(u_i)$ such that:

- for all $j \in [k] \setminus \{i\}$, there is no path from u_j to u_i in G, and
- for all $j \in [k]$, $v \neq u_j$
- $(b_i - 1) \cdot w(u_i v) + d_i \leq r$, and
- $dp[u_1 \ldots, u_{i-1}, v, u_{i+1}, \ldots, u_k, d_1, \ldots, d_{i-1}, d_i - w(u_i v), d_{i+1}, \ldots, d_k] =$ `TRUE`.

`FALSE` otherwise.

Calculating the recurrence can be done in time $\mathcal{O}(kn)$ if efficient data structures are used for storing the sets of reachable vertices for each vertex. Before we prove the correctness of the recurrence, note that we can safely assume that no source contains any in-edges, and no sink contains any out-edges; otherwise we do some simple prepossessing to ensure this holds. We can further assume all sinks come at the end of a topological sort of the graph.

We begin by proving the forward direction. Assume there exist vertex disjoint paths $P_1, P_2, \ldots, P_k$ which satisfy the required conditions. For a path P_i and a positive integer $j_i \leq |V(P)|$, we let $P_i^{j_i}$ denote the tail of P_i containing the j_i last vertices. Further, let $s_i^{j_i}$ be the first vertex of $P_i^{j_i}$, and let $d_i^{j_i}$ be the weight of $P_i^{j_i}$. We claim that for every combination of j_i for distinct $i \in [k]$, it holds that $dp[s_1^{j_1}, s_2^{j_2}, \ldots, s_k^{j_k}, d_1^{j_1}, d_2^{j_2}, \ldots, d_k^{j_k}] =$ `TRUE`.

We prove the claim by induction on the sum of the lengths of the tails, $J = \sum_{i \in [k]} j_i$. The base case occurs at $J = k$, when every tail has length 1. The induction hypothesis when proving the claim for larger J will be that the claim holds for $J - 1$, and from there we work our way up to the case when $J = \sum_{i \in [k]} |V(P_i)|$, at which point the forward direction of the correctness proof is complete.

In the base case $J = k$, we observe that $j_i = 1$ for every $i \in [k]$, since the domains for the j_i's are subsets of strictly positive integers, and their sum is k. Hence, $s_i^{j_i} = t_i$ and $d_i^{j_i} = 0$. By the base case of the recurrence, it then holds that $dp[s_1^{j_1}, s_2^{j_2}, \ldots, s_k^{j_k}, d_1^{j_1}, d_2^{j_2}, \ldots, d_k^{j_k}] =$ `TRUE`.

For the inductive step, consider some combination of j_i for all distinct $i \in [k]$ whose sum is J. Among the first vertices of the corresponding tails, pick the one who comes first according to some topological order, say $s_i^{j_i}$. Recall that we can assume all sinks are last in the topological order, hence $s_i^{j_i}$ is not a sink and $P_i^{j_i}$ contains at least two vertices. Let v be the second vertex of $P_i^{j_i}$. We observe that all the bullet points for the recurrence to give `TRUE` is satisfied when we try i and v: Since $s_i^{j_i}$ is the earliest in the topological order, no other start vertices has a path to it. By virtue of $P_1, P_2, \ldots, P_k$ being a valid solution, we know v is not in another path, and we know that the agent with salience factor b_i is willing to move along the edge $s_i^{j_i} v$ when the distance from v to t_i is $d_i^{j_i} - w(s_i^{j_i} v)$. And by the induction hypothesis, we know that $dp[s_1^{j_1}, \ldots, s_{i-1}^{j_{i-1}}, v, s_{i+1}^{j_{i-1}}, \ldots, d_{i-1}^{j_{i-1}}, d_i^{j_i} - w(s_i^{j_i} v), d_{i+1}^{j_{i+1}}, \ldots, d_k^{j_k}] =$ `TRUE`. This concludes the proof for the forward direction.

For the backward direction, assume $dp[s_1, s_2, \ldots, s_k, \ell_1, \ell_2, \ldots, \ell_k] =$ `TRUE`. We retrieve a solution as follows. Initially, let $P_i = s_i$ for each $i \in [k]$. Next, we iteratively append vertices to the paths in the following manner: Starting with $u_j \leftarrow s_j$ and $d_j \leftarrow \ell_j$ for $j \in [k]$, let i and v be a choice in the recurrence that satisfied the bullet point requirements for $dp[u_1, u_2, \ldots, u_k, d_1, d_2, \ldots, d_k]$.

Append v to P_i, and let $d_i \leftarrow d_i - w(u_i v)$ and $u_i \leftarrow v$ in the next iteration. Continue until we are left with only sinks.

We show that the paths created are disjoint. Assume for the sake of contradiction that two paths P_i and P_j are not disjoint, and let v be the topologically first vertex they share. Let v'_i and v'_j be the predecessors of v in respectively P_i and P_j. Without loss of generality, assume v was added to P_i before it was added to P_j. When v was added to j, it was done so when $u_i = v'_i$. But at the time, the vertex u_j must have had a path to v as well (through v'_j), hence the first condition of the recurrence is not satisfied. This is a contradiction.

We observe that the distance of P_i is ℓ_i, and that by the requirements in the recurrence an agent will always be motivated to move along the path. Thus, the constructed paths do indeed form a solution. This concludes the proof. □

Returning to our problem SIMPLE MOTIVATING SUBGRAPH with integer weights, we make use of the following observations that enables us to give an algorithm.

Proposition 12 (Kleinberg and Oren 2018, Theorem 5.1)**.** *If G' is a minimal motivating subgraph, then it contains a unique s-t path $P \subseteq G'$ that the agent will follow. Moreover, every node of G' has at most one outgoing edge that does not lie on P.*

Note that a consequence of Proposition 12 is that all branching vertices of a minimal motivating subgraph are on the path P.

Definition 13 (Merging vertex)**.** *The* in-degree *of a vertex v in a directed graph G is the number of edges in G that have v as its second endpoint. We say that v is a* merging vertex*, if its in-degree is at least two.*

Observation 14. *Consider an instance of the time-inconsistent planning model $M = (G, w, s, t, r, b)$, and let $G' \subseteq G$ be a minimal motivating subgraph with k branchings. Then there are at most k merging vertices in G'.*

We now give the algorithm for SIMPLE MOTIVATING SUBGRAPH with integer weights, where we use the algorithm for EXACT MOTIVATING k-LINKAGE IN DAG above as a subroutine. In short, the approach is to first guess which $\mathcal{O}(k)$ vertices that are "interesting" in the solution, i. e. have either in-degree or out-degree (or both) larger than 1 in G', and then try every possible way of connecting these points together using disjoint paths. Finally, we guess how heavy each such path should be, and apply the algorithm for EXACT MOTIVATING k-LINKAGE IN DAG. For details, see Algorithm 15.

Algorithm 15 (SIMPLE MOTIVATING SUBGRAPH with integer weights)**.** *Input:* An instance (G, w, s, t, b, r, k) of SIMPLE MOTIVATING SUBGRAPH, whose edge weights are integer that sum to W. *Output:* `TRUE` if there exists a motivating subgraph $G' \subseteq G$ with at most k branchings, `FALSE` otherwise.

We assume that $k \geq 1$, as otherwise we apply Algorithm 6. We also assume that there exists no motivating subgraph with strictly less than k branching vertices; if we are unsure, we first run this same algorithm with parameter $k-1$ (this will cause an extra factor k in the runtime).

We begin the algorithm by guessing the "interesting" vertices (in addition to s and t) of the subgraph we are looking for, G'. We guess:

- A set B of k distinct branching vertices; we let $B = \{u_1, u_2, \ldots, u_k\} \subseteq V(G)$ such that each selected vertex has out-degree at least two in G. Vertices are named according to their unique topological order — if there is no unique such order, skip this iteration (this is safe, since if B does not have a unique topological order, then there can not exist a path that visit all of B).

- A set $B^\star$ of k distinct "next-step" vertices; these are the immediate next vertices our agent will go to when standing at a branching vertex in G'. We let $B^\star = \{u_1^\star, u_2^\star, \ldots, u_k^\star\} \subseteq V(G)$ such that for each $i \in [k]$, $u_i^\star$ is in the out-neighborhood of u_i. Furthermore, for each $i \in [k-1]$, $u_i^\star$ must have a path to u_{i+1} (unless $u_i^\star = u_{i+1}$).

- A set $B^\diamond$ of k (not necessarily distinct) "shortcut" vertices; these are vertices the agent will *not* choose to go to from a branching vertex in G'. We let $B^\diamond = \{u_1^\diamond, u_2^\diamond, \ldots, u_k^\diamond\} \subseteq V(G)$ be such that for each $i \in [k]$, $u_i^\diamond$ is in the out-neighborhood of u_i, yet is different from $u_i^\star$.

- A set $B^\succ$ of k (not necessarily distinct) merge vertices; these are the vertices of G' with in-degree at least two. We let $B^\succ = \{u_1^\succ, u_2^\succ, \ldots, u_k^\succ\} \subseteq V(G)$.

While it is at this point mostly clear how the agent's path P will move through our set of vertices (it will follow the unique topological order of $B \cup B^\star$) we still need to guess how the shortcuts behave, and in particular how the merge vertices interact.

Towards this purpose, we create the (unweighted) *reachability graph* H, which illustrates every possible way of interconnecting the interesting vertices of G' we just guessed. Let $V(H) = B \cup B^\star \cup B^\diamond \cup B^\succ \cup \{s,t\}$, and let there be an edge if one vertex is reachable from another in G without going through vertices of H — in other words, let $E(H) = \{uv \in V(H) \times V(H) \mid$ there is a path from u to v in $G[(V(G) \setminus V(H)) \cup \{u,v\}]\}$.

We are now ready to guess exactly how the interesting points are interconnected in G'. We will do this by guessing H', a graph which can be obtained from G' (if it exists) by repeatedly smoothing non-interesting vertices. *Smoothing* a vertex is possible when it has both in-degree and out-degree exactly 1, and entails replacing the vertex by an edge whose weigh is the sum of the two previous edge weights. In particular, we guess:

- A subgraph $H' \subseteq H$ that obeys the following constraints:
  - $V(H') = V(H)$.
  - For each $i \in [k]$, the out-neighbors of u_i are exactly $u_i^\star$ and $u_i^\diamond$.
  - For each non-branching vertex $u \in V(H') \setminus (B \cup \{t\})$, it has exactly one out-neighbor.
  - There exists an s-t path P' that contains the edge $u_i u_i^\star$ for every $i \in [k]$.

- A weight function $w' : E(H') \to \mathbb{Z}_{\geq 0}$ that obeys the following constraints:
  - The sum of edges is bounded by W, i. e. $\sum_{e \in E(H')} w'(e) \leq W$.
  - The weight function w' is consistent with w: for each $e \in E(H') \cap E(G)$, $w'(e) = w(e)$.
  - For each branching vertex u_i, an agent standing there must evaluate the path along P' to be strictly better than moving to $u_i^\diamond$. More formally, for each $i \in [k]$, it holds that $b \cdot w'(u_i u_i^\star) + \textsc{Dist}_{H'}(u_i^\star, t) < b \cdot w'(u_i u_i^\diamond) + \textsc{Dist}_{H'}(u_i^\diamond, t)$.

We now create an instance I of EXACT MOTIVATING k-LINKAGE IN DAG:

- Let G be the graph.
- Let w be the weight function.
- Let the source terminals be the startpoints of edges $E(H')$.
- Let the sink terminals be the endpoints of edges $E(H')$.
- Let the target distances be the weight of edges in $E(H')$ according to w'.
- Salience factor: If the corresponding edge $uv \in E(H')$ is also in P', then let the salience factor be b. Otherwise, let it be 1.
- Reward: If the corresponding edge $uv \in E(H')$ is also in P', let the reward be r minus the shortest distance from v to t in H'. Otherwise, let the reward equal to the target distance.

If I is a yes-instance of EXACT MOTIVATING k-LINKAGE IN DAG, we return `TRUE`. If all created I-instances across all guesses are no-instances, we return `FALSE`.

Proof of Theorem 4. Recall that the theorem states that SIMPLE MOTIVATING SUBGRAPH can be solved in time $(|V(G)| \cdot W)^{\mathcal{O}(k)}$. We first prove that Algorithm 15 is correct, and return to runtime at the end of the proof.

Observe that the pre-processing steps do not change the answer. For the forward direction, assume $G' \subseteq G$ is a minimal solution witnessing that the input is a yes-instance with the minimum possible number of branching vertices, and let $P \subseteq G'$ be the path taken by the agent in G'. Let k be the number of branchings in G' — we assume $k \geq 1$, as otherwise Algorithm 6 would suffice. Let $B = \{u \in V(G') \mid |N_{G'}(u)| \geq 2\}$. Since vertices of B all have out-degree at least 2 in G, this set B will be guessed by Algorithm 15.

Similarly, let $B^\star$ be the set of vertices which are immediate successors to a vertex of B in P, let $B^\diamond$ be the neighbors of vertices of B that are not an immediate successor of that same vertex in P, and let $B^\succ$ be the set of vertices of in-degree at least two in G'. Due to Observation 14 it holds that $|B^\succ| \leq k$, so we see that Algorithm 15 will at some point guess all these sets correctly.

Imagine that we create the reachability graph H' by repeatedly smoothing all vertices of G' that have both in-degree and out-degree exactly 1, unless the vertex is in one of the sets $B^\star, B^\diamond$. In the smoothing process of a vertex u, we give the resulting edge weight equal to the sum of edge weights previously incident to u. By the minimality of G', the graph that remains is on the vertex set $B \cup B^\star \cup B^\diamond \cup M \cup \{s,t\}$. The number of edges in H' is exactly $V(H') + k - 1$, since every vertex has one out-edge, except the k branching vertices, which have two each, and the sink, which has none. Hence, H' and its weight function will be guessed by Algorithm 15.

Conducting the same smoothing process on P will yield the path P' found by the algorithm as well. By virtue of G' being a valid solution with P being the path taken by the agent, the iteration where all of the above is guessed correctly will not be skipped due to some branch tempting the agent to walk off the path. We observe that the resulting instance of EXACT MOTIVATING k-LINKAGE IN DAG is a yes-instance — the smoothed segments of G' to obtain H' are internally vertex disjoint and have length equal to the corresponding edge in H'. For the segments with motivation requirements, that is, edges in P' corresponding to path segments of P, we know that they satisfy the motivation requirement because they do so in G'.

For the backward direction, assume the algorithm returned TRUE, and let B, $B^\star$, $B^\diamond$, $B^\succ$, H', and w' be the guesses that led to this conclusion, let P' be the found path in H', and let I be the created yes-instance of EXACT MOTIVATING k-LINKAGE IN DAG. We build a solution G' by gluing together the vertices of H with the paths witnessing that I is a yes-instance, and let P denote the expansion of P' in G'. The agent will never be tempted to walk away from P, since the next step of P always appears like the best option at all branching points in every guess of w'. It remains to show that the agent is indeed motivated to move at all, for every edge of P.

Every edge $uv \in E(P)$ is part of some segment found in the solution to I. Let s' and t' be the source and sink in I where uv was part of the solution. Since uv is in P, the edge $s't'$ was in P', and as such the corresponding salience factor b', was set to b in I. Since I is a yes-instance, the agent is indeed motivated to move along every segment between s' and t', including across uv.

Finally, we argue for the runtime. The number of ways to guess B, $B^\star$, $B^\diamond$, and $B^\succ$, is $\mathcal{O}(n^{4k})$. Notice that the number of edges in H' is exactly $|V(H')| + k' - 1 \leq 5k+1$. However, the out-edges of B and t are not up for guessing, and the other vertices each have exactly one out-edge each in H'. The number of ways to guess H' is thus $\mathcal{O}((4k)^{3k})$. For the weights, $2k$ of them are already fixed because they exist in $E(G)$, so it remains to guess the weights for at most $3k+1$ of them. The number of ways to guess w' is bounded by $\mathcal{O}(W^{3k+1})$. Checking validity of weight functions takes $\mathcal{O}(k)$ and constructing I takes $\mathcal{O}(n+m)$, but these will be dominated by the $\mathcal{O}(kn^{5k+2}W^{5k+1})$ time spent to solve the EXACT MOTIVATING k-LINKAGE IN DAG instance. Including the extra factor k for incrementally trying larger values of k, gives a total runtime of $\mathcal{O}(kn^{4k}(4k)^{3k}W^{3k+1}kn^{5k+2}W^{5k+1})$, which simplifies to $(nW)^{\mathcal{O}(k)}$. □

4 Proof of Theorem 5

In this section we make the observation that SIMPLE MOTIVATING SUBGRAPH can not be solved in time $(|V(G)| \cdot W)^{\mathcal{O}(1)} \cdot f(k)$ for any function f of k only, unless FPT=W[1].

It follows from Slivkins Slivkins 2010 that the ℓ-LINKAGE problem is W[1]-hard parameterized by ℓ. In Albers and Kraft 2019, Albers and Kraft give a reduction from the ℓ-LINKAGE problem to MOTIVATING SUBGRAPH where all feasible minimal motivating subgraphs have exactly ℓ branching vertices. Moreover, the edge weights in their reduction — when scaled to integers — are bounded by a polynomial function of ℓ. It thus follows that a run-time of $(|V(G)| \cdot W)^{\mathcal{O}(1)} \cdot f(k)$ for SIMPLE MOTIVATING SUBGRAPH would place the ℓ-LINKAGE problem in FPT.

5 Conclusion

We have shown that the SIMPLE MOTIVATING SUBGRAPH problem is polynomial-time solvable when $k = 0$, and NP-complete otherwise (Theorem 3). However, when edge weights are (scaled to) integers and their sum is bounded by W, we gave a pseudo-polynomial algorithm which solves the problem in time $(|V(G)| \cdot W)^{\mathcal{O}(k)}$ (Theorem 4). Finally, we observed that an algorithm with run-time $(|V(G)| \cdot W)^{\mathcal{O}(1)} \cdot f(k)$ where f is a function of k only is not possible unless FPT=W[1] (Theorem 5).

We end the paper with an open question and a reflection for further research. First, a question; in Theorem 5, a more careful analysis reveals that the statement holds even when $W \in \mathcal{O}(k^2)$. But is it also the case when W is a constant?

Finally, a reflection. In Theorem 4 we give a pseudo-polynomial algorithm for SIMPLE MOTIVATING SUBGRAPH when the edge weights are scaled to integer values and the number of branchings is constant. While we can reasonably assume that edge weights are represented as fractions or integers after storing them in a computer, this suggests that the values might have been quantized or approximated in some way. However, this rounding could potentially alter the solution space. In particular when the cost of two paths are perceived to be almost equal at the branching point — then rounding values even a tiny bit can have big consequences. To overcome this, one might change the model such that the perceived difference of costs must exceed an epsilon for the agent's choice to be unequivocal.

6 Acknowledgments

This work has been supported by the Research Council of Norway via the project "MULTIVAL". We also thank the AAAI reviewers for valuable suggestions.

References

Akerlof, George A. 1991. "Procrastination and obedience". *The American Economic Review* 81 (2): 1–19.

Albers, Susanne, and Dennis Kraft. 2019. "Motivating time-inconsistent agents: A computational approach". *Theory of Computing Systems* 63 (3): 466–487.

— . 2017a. "On the Value of Penalties in Time-Inconsistent Planning". In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, ed. by Ioannis Chatzigiannakis et al., 80:10:1–10:12. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN: 978-3-95977-041-5. doi:10.4230/LIPIcs.ICALP.2017.10. http://drops.dagstuhl.de/opus/volltexte/2017/7387.

— . 2017b. "The price of uncertainty in present-biased planning". In *International Conference on Web and Internet Economics*, 325–339. Springer.

Bang-Jensen, J., and G.Z. Gutin. 2008. *Digraphs: Theory, Algorithms and Applications*. Springer Monographs in Mathematics. Springer London. ISBN: 9781848009981. https://books.google.no/books?id=5GdXCWhE4-MC.

Fortune, Steven, John Hopcroft, and James Wyllie. 1980. "The directed subgraph homeomorphism problem". *Theoretical Computer Science* 10 (2): 111 –121. ISSN: 0304-3975. doi:https://doi.org/10.1016/0304-3975(80)90009-2. http://www.sciencedirect.com/science/article/pii/0304397580900092.

Frederick, Shane, George Loewenstein, and Ted O'Donoghue. 2002. "Time Discounting and Time Preference: A Critical Review". *Journal of Economic Literature* 40 (2): 351–401. ISSN: 00220515. http://www.jstor.org/stable/2698382.

Gravin, Nick, et al. 2016. "Procrastination with Variable Present Bias". In *Proceedings of the 2016 ACM Conference on Economics and Computation*, 361–361. ACM.

Karp, Richard M. 1972. “Reducibility among combinatorial problems”. In *Complexity of Computer Computations*, 85–103. New York: Plenum Press.

Kleinberg, Jon, and Sigal Oren. 2018. “Time-inconsistent Planning: A Computational Problem in Behavioral Economics”. *Communications of the ACM* (New York, NY, USA) 61, no. 3 (): 99–107. ISSN: 0001-0782. doi:10.1145/3176189. http://doi.acm.org/10.1145/3176189.

Kleinberg, Jon, Sigal Oren, and Manish Raghavan. 2016. “Planning problems for sophisticated agents with present bias”. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, 343–360. ACM.

— . 2017. “Planning with multiple biases”. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, 567–584. ACM.

Laibson, David I. 1994. “Hyperbolic Discounting and Consumption”. PhD thesis, Massachusetts Institute of Technology, Department of Economics. http://hdl.handle.net/1721.1/11966.

McClure, Samuel M., et al. 2004. “Separate Neural Systems Value Immediate and Delayed Monetary Rewards”. *Science* 306 (5695): 503–507. ISSN: 0036-8075. doi:10.1126/science.1100907. eprint: https://science.sciencemag.org/content/306/5695/503.full.pdf. https://science.sciencemag.org/content/306/5695/503.

O’Donoghue, Ted, and Matthew Rabin. 1999. “Doing it now or later”. *American Economic Review* 89 (1): 103–124.

Samuelson, Paul A. 1937. “A Note on Measurement of Utility”. *The Review of Economic Studies* 4, no. 2 (): 155–161. ISSN: 0034-6527. doi:10.2307/2967612. eprint: http://oup.prod.sis.lan/restud/article-pdf/4/2/155/4345109/4-2-155.pdf. https://doi.org/10.2307/2967612.

Slivkins, Aleksandrs. 2010. “Parameterized Tractability of Edge-Disjoint Paths on Directed Acyclic Graphs”. *SIAM Journal on Discrete Mathematics* 24 (1): 146–157. doi:10.1137/070697781. https://doi.org/10.1137/070697781.

Tang, Pingzhong, et al. 2017. “Computational issues in time-inconsistent planning”. In *Proceedings of the 31st t AAAI Conference on Artificial Intelligence (AAAI)*.

Graphic design: Communication Division, UiB / Print: Skipnes Kommunikasjon AS

uib.no

ISBN: 9788230844021 (print)
9788230851104 (PDF)