Order-Related Problems Parameterized by Width

Emmanuel Arrighi

Thesis for the degree of Philosophiae Doctor (PhD)
University of Bergen, Norway
2022

UNIVERSITY OF BERGEN

Order-Related Problems Parameterized by Width

Emmanuel Arrighi

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

Date of defense: 25.05.2022

Year: 2022

Title: Order-Related Problems Parameterized by Width

Name: Emmanuel Arrighi

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.

During the work on this thesis, starting from April 1'st 2019, the author was funded by the Research Council of Norway (RCN) project "Parameterized Complexity for Practical Computing", Grant number 274526.

The author is deeply grateful to the RCN for this support.

During 2020 and 2021 the author and his supervisor Mateus de Oliveira Oliveira from the University of Bergen were part of a collaboration with Petra Wolf and Henning Fernau from Universität Trier, Germany. This collaboration was supported by IS-DAAD, with the project number 309319 on the Norwegian side, and on the German side by Deutscher Akademischer Austauschdienst (DAAD), Programm des Projektbezogenen Personenaustauschs (PPP), with the project number DAAD PPP 57525246.

All collaborators of the project are deeply grateful to the DAAD for this support.

The experiments in the paper "Three Is Enough for Steiner Tree" [Arrighi and de Oliveira Oliveira, 2021] received the support from the Sigma2 Network (Project no. NN9535K).

Acknowledgements

I had the luxury of having three supervisors Mike Fellows, Helwig Hauser and Mateus de Oliveira Oliveira. I am happy to say that I have learned from each of them and I am deeply grateful to each of them.

To Mike Fellows: First, you offered me this PhD position. I am very thankful for that. This represents the start of my journey in academia and I am really pleased that it started with you. During my PhD, you gave me a lot of freedom. I did not really follow the original plan but I am happy with the end result. I really appreciated the support and kind words for each of my publications. Sadly, a pandemic prevented you from being in Bergen during parts of my PhD. I would also like to thank Frances Rosamond for her unfailing energy and cheerfulness. You are a source of inspiration, especially for your projects about teaching and outreach.

To Helwig Hauser: As illustrated by the title of my thesis, my main focus was not in the field of visualization and we did not end up working on a project together. Nevertheless, you introduced me to the field of visualization, and I have learned quite a bit about it thanks to you. I am sure that what I learned will be help me in the future. I also want to thank you for the nice discussions we had. And I really enjoyed playing table tennis with you.

To Mateus de Oliveira Oliveira: You joined the party slightly after the others but you ended up as my main supervisor. I am very grateful for your day to day supervision and your close follow-up of my work. You put me at work, and that was not an easy task. I learned a lot from you. Last but not least, you invited me to the collaboration between Bergen and Trier with Petra Wolf and Henning Fernau and I am deeply thankful for this. This has been and, I hope, will continue to be an awesome and fruitful collaboration.

This collaboration was possible thanks to the support of the German Academic Exchange Service, and I am thankful for that. I would also like to thank the Research Council of Norway which funded my PhD position as part of the project “Parameterized Complexity for Practical Computing”. I am grateful to the Sigma2 Network for giving me access to their computational resources.

I would like to thank Petr Golovach for being the leader of the evaluation committee of my thesis and Ana Shirley Ferreira da Silva and Vinícius Fernandes dos Santos for agreeing to review my thesis.

The university administrative staff deserve a shout-out. You have always been available, kind and willing to help. You made my stay in Bergen really smooth. I want to give a special thank you to Pinar Heggernes and Linda Vagtskjold for being really accessible and making this department of informatics really welcoming.

All my thanks to the algorithm group of Bergen for having me. The home office times were made all the more painful because of how nice the group was. Thankfully, we managed to restart the weekly meetings with the Friday seminars. The list of people I want to thank is long, but I will name a few anyway. Thank you Torstein J.F. Strømme and Emmanuel Sam, I am really happy to have shared the best office of the group with you. A special "merci" for the French team William Lochet, Christophe Crespelle and Benjamin Bergougnoux. Thank you Jan Arne for your help and advice. Thank you Noeska Smit for being my mentor and for all the advice you gave me. Additional thanks to Benjamin Chetioui, Erlend Vagset, Jonathan Cubides and Yngve Kristiansen.

The collaboration with Henning Fernau and Petra Wolf constituted the biggest part of my PhD. I am extremely grateful to both of you. Henning Fernau, it is always a delight to work with you. I also want to thank you a lot for all the help and guidance you have given me. Petra Wolf, you brought me so much. I cannot thank you enough. You were sort of my big sister in academia, and I am blessed that you are so much more now. You also brought three students of yours in the project, namely Kevin Goergen, Sven Fiergolla and Patrik Neises. It is a real pleasure to work with them, thanks to the group! Through this project, I was able to meet other collaborators, I want to thank Markus Holzer and Ismaël Jecker for the nice time working together.

This next part is for my family, and therefore in French. J'ai la chance d'avoir une famille merveilleuse et c'est important pendant une thèse. Même si je n'ai pas toujours été très bavard, particulièrement sur le déroulement de ma thèse, ça a toujours été un plaisir, un grand réconfort et une source de motivation de rentrer vous voir. Maman, c'est toujours avec joie que je rentre à la maison (et promis je rangerai ma chambre). J'apprécie ta curiosité et tes efforts pour comprendre mon travail. C'est toujours avec beaucoup de plaisir que je réponds à tes questions et que je t'explique ce que je fais. Caroline et Claire, je devrais vous appeler plus souvent car à chaque fois c'est un plaisir, vous êtes les meilleures grandes sœurs possible. Vous m'avez offert une filleule, Gabrielle, ainsi qu'un neveu, Marin, tous les deux merveilleux. J'espère pouvoir les voir plus souvent.

Olivier, on a des discussions plus technique mais c'est toujours un plaisir. J'espère qu'on pourra naviguer ensemble plus souvent. Merci à tous!

I am fortunate enough to have an amazing group of friends, the dream team from ENS Cachan. I have awesome memories, thank you all for the hours of discussion, games and everything. I would especially like to name some of them. To keep it short, I will only mention one trait for each of you, but you are way more than that. Mathieu Huot, discussing with you is always a blast and you always give good advice. William Babonnaud, seeing you is the promise of a great meal! Thomas Dupriez, my daily play partner, after a day of work, it is great to join you on some game. Guillaume Hocquet, my best roommate ever. Antonin Penon, the best physicist, having you around always brings a good mood. Clément Beauseigneur, the traitor in the industry, but we still like you.

In Bergen, I want to thank BSI Seiling for the great time I had sailing. I want to thank Basile de Fleurian, Laura Gaiger, Natacha Fabregas, Tobias Zolles, Tristan Petit, Zoé Koenig and Irina Kraft for the great time sailing with you! Furthermore, my special thanks go to Basile for all our winter sailing.

Petra, you are my optimal solution. I don't have the words to express how happy I am to be with you. I am not exaggerating when I say that this thesis would not have existed without you. You have given me so much. Simply thank you.

Abstract

In the main body of this thesis, we study two different order theoretic problems. The first problem, called Completion of an Ordering, asks to extend a given finite partial order to a complete linear order while respecting some weight constraints. The second problem is an order reconfiguration problem under width constraints.

While the Completion of an Ordering problem is NP-complete, we show that it lies in FPT when parameterized by the interval width of ρ. This ordering problem can be used to model several ordering problems stemming from diverse application areas, such as graph drawing, computational social choice, and computer memory management. Each application yields a special partial order ρ. We also relate the interval width of ρ to parameterizations for these problems that have been studied earlier in the context of these applications, sometimes improving on parameterized algorithms that have been developed for these parameterizations before. This approach also gives some practical sub-exponential time algorithms for ordering problems.

In our second main result, we combine our parameterized approach with the paradigm of *solution diversity.* The idea of solution diversity is that instead of aiming at the development of algorithms that output a *single* optimal solution, the goal is to investigate algorithms that output a small set of *sufficiently good* solutions that are *sufficiently diverse* from one another. In this way, the user has the opportunity to choose the solution that is most appropriate to the context at hand. It also displays the richness of the solution space. There, we show that the considered diversity version of the Completion of an Ordering problem is fixed-parameter tractable with respect to natural paramaters that capture the notion of *diversity* and the notion of *sufficiently good* solutions. We apply this algorithm in the study of the Kemeny Rank Aggregation class of problems, a well-studied class of problems lying in the intersection of order theory and social choice theory.

Up to this point, we have been looking at problems where the goal is to find an optimal solution or a diverse set of good solutions. In the last part, we shift our focus from finding solutions to studying the solution space of a problem. There we consider the following order reconfiguration problem: Given a graph G together with linear orders τ and τ' of the vertices of G, can one transform τ into τ' by a sequence of swaps of adjacent elements in such a way that at each time step the resulting linear order has cutwidth (pathwidth) at most w? We show that this problem always has an affirmative answer when the input linear orders τ and τ' have cutwidth (pathwidth) at most $w/2$. Using this result, we establish a connection between two apparently unrelated problems: the reachability problem for two-letter string rewriting systems and the graph isomorphism problem for graphs of bounded cutwidth. This opens an avenue for the study of the famous graph isomorphism problem using techniques from term rewriting theory.

In addition to the main part of this work, we present results on two unrelated problems, namely on the Steiner Tree problem and on the Intersection Non-emptiness problem from automata theory.

List of publications

This section presents a list of papers to which the author of this thesis contributed. The list is separated in two parts: the main line of research on which most of this thesis is based on, and two additional papers in distinct topics. While the author of this thesis published his research papers under the name Emmanuel Arrighi, the full name of the author is Emmanuel Jean-Paul Pierre Arrighi.

Parameterized Complexity of Ordering Related Problems and Diversity of Solutions

1. Emmanuel Arrighi, Henning Fernau, Mateus de Oliveira Oliveira, Petra Wolf. Width Notions for Ordering-Related Problems. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14–18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)* 182 (2020) pp. 9:1 — 9:18.
   DOI: 10.4230/LIPIcs.FSTTCS.2020.9.

2. Emmanuel Arrighi, Henning Fernau, Mateus de Oliveira Oliveira, Petra Wolf. Order Reconfiguration Under Width Constraints. In *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23–27, 2021, Tallinn, Estonia* 202 (2021) pp. 8:1 — 8:15.
   DOI: 10.4230/LIPIcs.MFCS.2021.8.

3. Emmanuel Arrighi, Henning Fernau, Daniel Lokshtanov, Mateus de Oliveira Oliveira, Petra Wolf. Diversity in Kemeny Rank Aggregation: A Parameterized Approach. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19–27 August 2021* (2021) Main Track pp. 10 — 16.
   DOI: 10.24963/ijcai.2021/2.

Other works

4. Emmanuel Arrighi, Mateus de Oliveira Oliveira. Three Is Enough for Steiner Trees. In *19th International Symposium on Experimental Algorithms, SEA 2021, June 7–9, 2021, Nice, France* 190 (2021) pp. 5:1 — 5:15.
   DOI: 10.4230/LIPIcs.SEA.2021.5.

5. Emmanuel Arrighi, Henning Fernau, Stefan Hoffmann, Markus Holzer, Ismaël Jecker, Mateus de Oliveira Oliveira, Petra Wolf. On the Complexity of Intersection Non-emptiness for Star-Free Language Classes. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15–17, 2021, Virtual Conference* 213 (2021) pp. 34:1 — 34:15.
   DOI: 10.4230/LIPIcs.FSTTCS.2021.34.

Contents

Part I

Basic Concepts

Chapter 1

Introduction

Many computational problems can be phrased as the task of arranging a collection of combinatorial objects into a minimum-cost linear order that satisfies certain constraints. Examples of natural problems that fall in this category are One-Sided Crossing Minimization (OSCM), a prominent problem in the field of graph drawing and VLSI design [Battista et al., 1999, Healy and Nikolov, 2013, Mutzel, 2009, Sechen, 2012, Stallmann et al., 2001], Grouping by Swapping (GbS), a problem with applications in computer memory management [Downey and Fellows, 2013, Garey and Johnson, 1979, Wong and Reingold, 1991], and Kemeny Rank Aggregation (KRA), a prominent problem in the field of computational social choice [Dwork et al., 2001, Kemeny, 1959]. Some graph properties can be defined by linear orders of the vertices such as the perfect elimination linear order for chordal graphs [Fulkerson and Gross, 1965], cutwidth [Thilikos et al., 2005] and the vertex separation number [Ellis et al., 1994]. In this introduction, we will use several notions that will be formally defined in later chapters of this part.

Given a partial order, a fundamental question is to complete it into a linear order. A direct application of this problem is the scheduling of tasks. Given some tasks together with dependencies between these, to realise those tasks, one needs to first find an order of the tasks that satisfy the dependency conditions. Without constraints, the problem of finding a linear extension is solvable in linear time [Cormen et al., 2001, Section 22.4]. However, asking for a linear order that minimizes a cost function makes the problem NP-complete [Fernau, 2005, Sec. 6.4]. The latter problem is called Completion of an Ordering (CO). In this work, we first give a unified view on OSCM, GbS and OSCM through the lens of the Completion of an Ordering problem under the framework of parameterized complexity and the framework of diversity of solutions. Then, we look at the reconfiguration of linear orders of vertices of a graph under width constraints.

A central notion in the framework of parameterized complexity is the fixed-parameterized tractability. An algorithm for a given computational problem is said to be fixed-parameter tractable with respect to parameters $k_1, \dots, k_r$ if it runs in time $f(k_1, \dots, k_r) \cdot n^{\mathcal{O}(1)}$, where n is the size of the input and f is a function that depends only on the parameters. The intuition is that if the range of the parameters is small on instances of practical relevance, then even if the function f grows relatively fast, the algorithm can be considered to be fast enough *for practical purposes.*

Linear extension as a general framework

A natural parameter that arises when studying problems such as OSCM, GbS and KRA from the perspective of parameterized complexity theory is the cost k of a solution. In particular, the best algorithm for OSCM, parameterized by the cost of a solution k, is the algorithm due to Kobayashi and Tamaki [Kobayashi and Tamaki, 2015] which runs in time[1] $\mathcal{O}^\star(2^{\sqrt{2k}})$ and the best single-exponential algorithm for KRA runs in time $\mathcal{O}^\star(1.403^k)$ [Simjour, 2009], while sub-exponential algorithms of type $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ have been proposed in [Karpinski and Schudy, 2010], with some unclear constant hidden in the $\mathcal{O}$-notation of the exponent. Not surprisingly, they have been devised under distinct paradigms and with substantially distinct sets of techniques.

In Chapter 6, significantly extending the ideas started out in [Fernau, 2005, Fernau et al., 2014], we leverage the Completion of an Ordering problem (CO) to provide a unified framework for the study of several cost-parameterized ordering problems. In this problem, we are given a partial order ρ on a set V, and a function $\mathfrak{c} : V \times V \to \mathbb{N}$ assigning costs to incomparable pairs, and the goal is to compute a minimum-cost linear extension of ρ. Interestingly, a natural *structural* parameter that arises in this context is the pathwidth of the cocomparability graph of the input partial order ρ. This graph has V as its vertex-set and there is an undirected edge between vertices u and v if and only if u and v are not related in the partial order. The main result of this chapter states that CO, parameterized by the interval width w of the input partial order, can be solved in time $\mathcal{O}^\star(2^w)$. Additionally, our algorithm is optimal under the Exponential Time Hypothesis (ETH for short). Using our result on CO, together with reductions from OSCM, GbS and KRA to PCO, the natural restriction of CO to positive costs, we obtain algorithms for these three problems (parameterized by width, or by the standard parameter, or by other problem-specific structural parameters) whose running times often match or improve on the best algorithms for the three problems.

When reducing OSCM or GbS to PCO, the partial order one obtains is an interval order,

[1]Recall that the $\mathcal{O}^\star$-notation suppresses polynomial factors.

meaning that the cocomparability graph of this order is an interval graph. Interval orders play an important role in partial order theory due to the fact that their interval width can be computed in linear time. Additionally, they find applications in many contexts of practical relevance such as scheduling, online and packing algorithms, see [Trotter, 1997]. Inspired by this, we define the Positive Completion of an Interval Ordering (PCIO) problem, a version of PCO where the input partial order is required to be an interval order. In this restricted version, our main algorithm for CO parameterized by interval width can be converted into a sub-exponential $\mathcal{O}^\star(2^{\sqrt{2k}})$-time FPT algorithm for PCIO, parameterized by cost k.

Building on recent advances in the theory of C_k-free graphs [Chudnovsky et al., 2020] we establish an upper bound for the pathwidth of a cocomparability graph in terms of the number of edges of the graph. As a by-product of this result, we obtain the first algorithm running in time $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ (Theorem 41) for the positive completion of an ordering problem (PCO). Previous to our work, the best algorithm for this problem parameterized by cost had asymptotic time complexity of $\mathcal{O}^\star(k^{\mathcal{O}(\sqrt{k})}) = \mathcal{O}^\star(2^{\sqrt{k}\log k})$. Therefore, we remove the log-factor in the exponent. This is optimal under ETH.

Our width-based approach also allows us to improve on a parameterized algorithm for KRA based on the parameter *maximum range* (of a candidate) as introduced and discussed in [Betzler et al., 2009]. Further, it can be used to show that GbS is also fixed parameter tractable when parameterized by a parameter called scope coincidence degree, a natural parameter in the context of strings. This gives the first algorithmic use of this structural string parameter.

Our approach for CO is built on dynamic programming on a path decomposition of the cocomparability graph of the partial order. Notice that this path decomposition structure has been recently exploited for counting the number of linear extensions by Eiben *et al.* [Eiben et al., 2019]. Here, we use this approach to find the cheapest linear extension.

Diversity is the key

Traditionally, in optimization theory, when given an instance of a computational problem, one is interested in computing *some* optimal solution for the instance in question. For certain problems of practical relevance, this framework may not be satisfactory because it precludes the user from the possibility of choosing among optimal solutions in case more than one exists, or even from choosing a slightly less optimal solution that may be a better fit for the intended application, due to subjective factors.

A recent up-coming trend of research in artificial intelligence, called *diversity of solutions* [Petit and Trapp, 2019, Baste et al., 2020, Ingmar et al., 2020, Baste et al., 2019, Fomin et al., 2020], has focused on the development of notions of optimality that may be more appropriate in settings where subjectivity is essential. The idea is that instead of aiming at the development of algorithms that output a *single* optimal solution, the goal is to investigate algorithms that output a small set of *sufficiently good* solutions that are *sufficiently diverse* from one another. In this way, the user has the opportunity to choose the solution that is most appropriate to the context at hand. The intuition is that the criteria employed by the user to decide what an appropriate solution is may be subjective, and therefore, impractical or even impossible to be formalized at the level of the problem specification. Examples of such criteria are aesthetic, economic, political, environmental criteria. Another motivation comes from the problem of finding several good committees such that each committee member is not overloaded with these commitments, as described in [Bredereck et al., 2020]; again, some diversity could be helpful.

The framework of diversity of solutions, under distinct notions of diversity, has found applications in several subfields of artificial intelligence, such as information search and retrieval [Gollapudi and Sharma, 2009, Abbassi et al., 2013], mixed integer programming [Glover et al., 2000, Danna and Woodruff, 2009, Petit and Trapp, 2015], binary integer linear programming [Greistorfer et al., 2008, Trapp and Konrad, 2015], constraint programming [Hebrard et al., 2005, Hebrard et al., 2007], SAT solving [Nadel, 2011], recommender systems [Adomavicius and Kwon, 2014], routing problems [Schittekat and Sörensen, 2009], answer set programming [Eiter et al., 2013], decision support systems [Løkketangen and Woodruff, 2005, Hadžić et al., 2009], genetic algorithms [Gabor et al., 2018, Wineberg and Oppacher, 2003], planning [Baste et al., 2019], and in many other fields. Recently, a general framework for addressing diversity of solutions from the perspective of parameterized complexity theory was developed [Baste et al., 2020]. This framework allows one to convert dynamic programming algorithms for finding an optimal solution for instances of a given problem into dynamic programming algorithms for finding a small set of sufficiently diverse solutions.

Notice that there is also the related area of enumerating all optimal solutions, or at least encoding them all; this is known as knowledge compilation in artificial intelligence, see, e.g., [Darwiche and Marquis, 2002, Fargier and Marquis, 2014, Marquis, 2011]. These types of questions have also been considered from a more combinatorial viewpoint; confer [Fomin et al., 2014, Golovach et al., 2018, Kanté et al., 2014, Kanté and Nourine, 2016]. But from a practical perspective, it is not really desirable to confront a user with an exponential number of different solutions, but she wants to know what the real alternatives are.

Two measures of diversity of a set S of solutions have been particularly explored in the literature. The first one is the sum of distances between pairs of solutions in S. The second one is the minimum distance s between any two solutions in S. This last notion has been also known in the literature as *scatteredness* [Galle, 1989]. Both notions have been used in the context of vertex- and edge-problems on graphs using the Hamming distance of solutions as the distance measure [Baste et al., 2020, Gabor et al., 2018, Wineberg and Oppacher, 2003, Fomin et al., 2020]. It is worth noting that when used alone, the diversity measure defined as the sum of Hamming distances has some weaknesses. For instance, if we take a pair $\{A, B\}$ of solutions of diversity d, then the list $A_1, A_2, \ldots, A_r, B_1, B_2, \ldots, B_r$, where each A_i is a copy of A, and each B_i is a copy of B, has high diversity ($d' > r^2 \cdot d$), while this list clearly opposes the intuitive notion of a diverse set of solutions. This weakness can be significantly mitigated by considering diversity in conjunction with scatteredness. For instance, by setting $s \geq 1$, we already guarantee that all elements in a list of solutions will be distinct from each other.

One source of difficulty when trying to develop efficient algorithms for diverse variants of computational problems is the fact that these problems may be computationally hard. In particular, many interesting computational problems that are suitable for being studied from the perspective of diversity of solutions are already NP-hard in the usual variant in which one asks for a single solution. Additionally, it may be the case that even problems that are polynomial-time solvable in the single-solution version become NP-hard when considering diverse sets of solutions. One way to circumvent this difficulty is to combine the framework of diversity of solutions with the framework of parameterized complexity theory [Downey and Fellows, 1999].

In Chapter 7, we investigate the impact of the notions of diversity of solutions and of fixed parameter tractability theory in the context of social choice theory. In particular, we focus on the framework of preference list aggregation introduced by Kemeny in the late fifties [Kemeny, 1959]. Intuitively, preference lists are a formalism used in social choice theory to capture information about choice in applications involving the selection of candidates, products, etc., by a group of voters. The task is then to find a ranking of the candidates that maximizes the overall satisfaction among the voters. This problem is commonly referred to in modern terminology as the *Kemeny rank aggregation* (KRA) problem. In its most general setting, the ranking cast by each voter is a partial order on the set of candidates. The distance measure we use to define our diverse version for KRA is the Kendall-Tau distance which is widely used in the context of preference aggregation.[2] Its popularity is underlined by articles describing these issues for the

[2]To cite from the Stanford Encyclopedia of Philosophy [List, 2013]: *At the heart of social choice theory is the analysis of preference aggregation, understood as the aggregation of several individuals' preference rankings of two or more social alternatives into a single, collective preference ranking (or*

interested public audience; see [Farkas and Timotity, 2019].

The result of this chapter is a multiparametric algorithm for Diverse KRA over partially ordered votes that runs in time $f(w, r, \delta, s) \cdot d \cdot n \cdot \log(n^2 \cdot m)$ where n is the number of candidates, m is the number of votes, r, δ, s and d are the parameters discussed above, and w is the *unanimity width* of the votes. That is to say, the pathwidth of the cocomparability graph of the unanimity order of the input votes (Corollary 84). Intuitively, this width measure is a quantification of the amount of disagreement between the votes. Note that pathwidth and treewidth coincide for the class of cocomparability graphs [Habib and Möhring, 1994].

Second, we note that the notion of Kendall-Tau distance between partial orders (formally defined in Section 5.3), which is used to define our notion of diversity, can be applied equally well in the more general context of the Completion of an Ordering problem (CO), a problem of fundamental importance in order theory that unifies several problems of relevance for artificial intelligence, such as KRA, One-Sided Crossing Minimization (an important sub-routine used in the search for good hierarchical representations of graphs), and Grouping by Swapping (a relevant problem in the field of memory management) [Wong and Reingold, 1991]. For a matter of generality, we first develop a $f(w, r, \delta, s) \cdot d \cdot n \cdot \log(n^2 \cdot m)$ time algorithm for Diverse CO (Theorem 83) and then obtain our main result for Diverse KRA as a corollary. In the more general context of CO, the parameter w is the width of the cocomparability graph of the partial order given at the input.

It is worth noting that in the context of KRA over totally ordered votes, the existence of diverse sets of high-quality solutions implies that any optimal solution disagrees significantly with some of the voters. More precisely, let k_{opt} be the cost of an optimal solution and suppose that there are two solutions with cost $k_{\text{opt}} + \delta_1$ and $k_{opt} + \delta_2$, respectively. It is possible to show that $\max\{k_{opt} + \delta_1, k_{opt} + \delta_2\}$ is at least half the number of votes. Therefore, if two solutions have small solution imperfection, then k_{opt} is large. In the other direction, if there is a strong consensus among the voters (k_{opt} is small) then δ_1 or δ_2 must be large. Intuitively, in the context of aggregation over totally-ordered votes, the more disagreement there is between an optimal ranking and the ranking provided by the voters, the more one can benefit from the framework of solution diversity. In the context of aggregation over partially ordered votes, such a correlation between solution imperfection and optimality does not necessarily hold even for constant unanimity width. For instance, consider a set of partially ordered votes where the unanimity order is a bucket order with buckets of size 2 (i.e., unanimity width equal to 1). Then depending on the instance, we can have diverse sets of solutions with solution imperfection 0 and optimal

choice) over these alternatives.

cost 0. Therefore, in the context of partially ordered votes, the notion of diversity makes sense even in the case where voters have small disagreement between each other.

How to reorder your library without having a mess in between

While in the last setting, we were interested in finding a set of solution where the solutions are sufficiently far apart from each other, we are now shifting our focus to relationships among solutions of a problem instance. Those relationships are the topic of the field of reconfiguration [Ito et al., 2011, Nishimura, 2018, Wrochna, 2018]. Here, by reconfiguration of one solution into another, we mean a sequence of steps where each step transforms a feasible solution into another. Three fundamental questions in this context are: (1) Is it the case that any two solutions can be reconfigured into each other? (2) Can any two solutions be reconfigured into each other in a polynomial number of steps? (3) Given two feasible solutions X and Y, can one find in polynomial time a reconfiguration sequence from X to Y?

In Chapter 8, we study the reconfiguration problem in the context of linear arrangements of the vertices of a given graph G. The space of feasible solutions is the set of all linear orders of cutwidth (pathwidth) at most w for some given $w \in \mathbb{N}$. We say that a linear order τ can be reconfigured into a linear order τ' in width w if there is a sequence $\tau_1, \ldots, \tau_m$ of linear orders of width at most w such that $\tau_1 = \tau$, $\tau_m = \tau'$ and for each $i \in \{2, \ldots, m\}$, τ_i is obtained from τ_{i-1} by swapping two adjacent vertices. The main result of the chapter (Theorem 88) states that if τ and τ' are linear orders of cutwidth at most w, then τ can be reconfigured into τ' in width at most $2w$. Additionally, reconfiguration in width at most $2w$ can be done using at most $\mathcal{O}(n^2)$ swaps. Finally, we show that a reconfiguration sequence can be found in polynomial time.

Our results on reconfiguration of linear arrangements can be used to establish an interesting connection between two apparently unrelated computational problems: reachability for two-letter string rewriting and graph isomorphism.

A two-letter rewriting rule over a given alphabet Σ is a rewriting rule of the form $ab \to cd$ for letters $a, b, c, d \in \Sigma$. A two-letter string rewriting system is a collection R of two-letter string rewriting rules. The reachability problem for such a rewriting system R is the problem of determining whether a given string $x \in \Sigma^n$ can be transformed into a given string $y \in \Sigma^n$ by the application of a sequence of two-letter rewriting rules of R. On the other hand, in the graph isomorphism problem, we are given two graphs, G and G', and the goal is to determine whether there exists a bijection φ from the vertex set of G to the vertex set of G' in such a way that an edge $\{u, v\}$ belongs to G if and only if the edge $\{\varphi(u), \varphi(v)\}$ belongs to G'.

In order to describe more precisely the connections between two-letter term rewriting and graph isomorphism, we briefly discuss the notion of slices and unit decompositions. A *slice* is a graph $\mathbf{S}$ where the vertices are partitioned into a center C and special in-frontier I and out-frontier O that can be used for composition. A slice $\mathbf{S}_1$ can be glued to a slice $\mathbf{S}_2$ if the out-frontier of $\mathbf{S}_1$ can be coherently matched with the in-frontier of $\mathbf{S}_2$. In this case, the gluing gives rise to a bigger slice $\mathbf{S}_1 \circ \mathbf{S}_2$ which is obtained by matching the out-frontier of $\mathbf{S}_1$ with the in-frontier of $\mathbf{S}_2$. A *unit slice* is a slice with a unique vertex in the center. Any slice $\mathbf{S}$ can be decomposed into a sequence of unit slices. More specifically, a *unit decomposition* is a sequence $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2 \dots \mathbf{S}_n$ of unit slices with the property that for each $i \in [n-1]$, $\mathbf{S}_i$ can be glued to the slice $\mathbf{S}_{i+1}$. The result of gluing the unit slices in $\mathbf{U}$ is a slice $\mathring{\mathbf{U}}$ with n center vertices. Conversely, any slice $\mathbf{S}$ with n center vertices can be written as a unit decomposition $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2 \dots \mathbf{S}_n$ with the property that $\mathring{\mathbf{U}}$ is isomorphic to $\mathbf{S}$.

An important remark connecting unit decompositions and the notion of cutwidth is that if a slice $\mathbf{S}$ has cutwidth w, then $\mathbf{S}$ can be decomposed into a unit decomposition $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2 \dots \mathbf{S}_n$ where for each $i \in [n]$, $\mathbf{S}_i$ has at most w vertices in each frontier except for the in-frontier of $\mathbf{S}_1$ and the out-frontier of $\mathbf{S}_n$. Therefore, if we let $\mathbf{\Sigma}(w)$ denote the set of all unit slices with frontiers of size at most w, then any graph G with n vertices of cutwidth at most w can be written as a word (unit decomposition) of length n over the alphabet $\mathbf{\Sigma}(w)$. In this chapter, for each $w \in \mathbb{N}$, we introduce a suitable two-letter string rewriting system $R(w)$ over the alphabet $\mathbf{\Sigma}(w)$ with the following property: if $\mathbf{U}$ and $\mathbf{U}'$ are two unit decompositions over $\mathbf{\Sigma}(w)$ and if $\mathbf{U}$ can be transformed into $\mathbf{U}'$ using the rewriting rules in $R(w)$, then the graphs $\mathring{\mathbf{U}}$ and $\mathring{\mathbf{U}}'$ are isomorphic. Our second main result is a partial converse for this property. More precisely, we show that given two unit decompositions $\mathbf{U}$ and $\mathbf{U}'$ over $\mathbf{\Sigma}(w)$, if the graphs $\mathring{\mathbf{U}}$ and $\mathring{\mathbf{U}}'$ are isomorphic, then each of these unit decompositions can be transformed into one another by the application of rewriting rules from the string rewriting system $R(2w)$ (Theorem 98).

The proof of Theorem 98 is heavily based on Theorem 88. An important feature of this proof is that, given an isomorphism from $\mathring{\mathbf{U}}$ to $\mathring{\mathbf{U}}'$, one can construct a sequence of rewriting steps transforming $\mathbf{U}$ into $\mathbf{U}'$. Conversely, given any such a sequence, we are able to construct an isomorphism from $\mathring{\mathbf{U}}$ to $\mathring{\mathbf{U}}'$. This result, together with the fact that unit decompositions of minimum cutwidth can be approximated in FPT time, implies that the graph isomorphism problem for graphs of cutwidth at most w is FPT-equivalent to the reachability problem for $R(2w)$ (Theorem 100).

Another width parameter for linear orders that has been studied in the context of graph theory is the vertex separation number of a graph [Ellis et al., 1994]. This parameter may be seen as an order theoretic interpretation of the notion of pathwidth. The techniques used to prove Theorem 88 can be generalized to prove that reconfiguration of linear orders of vertex separation number w can always be achieved in width at most $2w$ (Theorem 101). While we do not provide a string-rewriting interpretation of this result, we do state it formally in Section 8.4 since this result may be of independent interest in the field of reconfiguration.

Structure of this Work

This work is divided into three parts. In Part I, the current part, we introduce the basic concepts used in Part II. Chapter 2 contains the basic definitions and notations for fundamental notions that will be used throughout this thesis such as sets, graphs and partial orders. In Chapter 3, we highlight some basic concepts of complexity theory used in Part II such as the notion of NP-completeness and of fixed-parameterized tractability. We illustrate these notions using the Vertex Cover problem as an example. Related to parameterized complexity, Chapter 4 lists the definitions and gives the background to several width measures for graphs, partial orders and strings used in Part II. In the last chapter of Part I, Chapter 5, we give formal definitions of the problems studied in Part II. We also discuss related work concerning these problems. The problems we consider span over a variety of fields of computer science such as ordering, graph drawing, social choice, strings, reconfiguration, graph theory and rewriting systems.

Part II contains the results that we introduced above. They have been published in [Arrighi et al., 2020, Arrighi et al., 2021c, Arrighi et al., 2021a]. This part is organized as follow. First, we look at OSCM, GbS and KRA using CO as the common generalization. In Chapter 6, we design an FPT algorithm for CO parameterized by the interval width of the input partial order and give a reduction for each problem to PCO, a restriction of CO. Then, in Chapter 7, we focus on KRA and CO, and look at those problems from the point of view of diversity. We define the diverse versions of both problems and give a multiparametric algorithm parameterized by the interval width and several diversity parameters. We end this part with Chapter 8. In this chapter, we look at reconfiguration problems under width constraints. Given a graph, we consider the problem of transforming a linear order of its vertices into another linear order in such a way that every intermediate step have bounded cutwidth or vertex separation number. Using our result in this chapter, we draw a connection between the graph isomorphism problem and a particular two-letter rewriting system.

Part III consists of the full versions of two published papers whose results do not fit in the framework of ordering problems. These papers are discussed in the next subsection.

Results of Part III

Part III consists of two research papers, [Arrighi and de Oliveira Oliveira, 2021] and [Arrighi et al., 2021b] that do not fit in the line of research described in the first two parts of this thesis. Here we give a short overview of the main results obtained in each paper.

Three is Enough for Steiner Tree.
In Chapter 9, we present the work in [Arrighi and de Oliveira Oliveira, 2021]. In this work, we study the Steiner Tree problem in graphs. To be able to define this problem, we first introduce the notion of Steiner trees. Let G be an undirected edge-weighted graph and let $S \subseteq V(G)$ be a subset of vertices of G whose elements are called *terminals.* A *Steiner tree* in G is a subgraph T of G such that T is a tree and $S \subseteq V(T)$. We note that T may contain non-terminal vertices. The cost of a tree, $\mathfrak{c}(T)$, is the sum of the costs of its edges.

Problem name: Steiner Tree
Given: A graph G, a set of terminals S, and an integer $k \in \mathbb{N}$.
Output: Is there a subtree T of G such that $S \subseteq V(T)$ and $\mathfrak{c}(T) \leq k$?

Intuitively, in the Steiner tree problem for graphs, the goal is to find a minimum-weight tree in G whose nodes span all terminals in S. This is a fundamental NP-hard problem [Karp, 1972], which has been studied since the seventies [Hakimi, 1971] and which has found applications in several fields of research such as planning [Keyder and Geffner, 2009], social networks [Lappas et al., 2009], sensor networks [Lee and Younis, 2010], community detection [Chiang et al., 2013], VLSI circuit design [Joobbani, 2012], as well as in numerous applications in industry [Cheng et al., 2004].

In this paper, we introduce a new simple combinatorial heuristic for the Steiner tree problem that works by replacing sub-trees of a prospective solution with Steiner trees on three terminals. We also note that this optimization procedure can also be used to improve the weight of sub-optimal Steiner trees output by other solvers.

To validate our new heuristic, we implemented a solver in C++ and benchmark it against several state of the art solvers for the Steiner tree problem on well known data sets. These solvers implement several paradigms, such as genetic algorithms, linear programming algorithms, local search algorithms as well as algorithms with approximation guarantees. The data sets were obtained from a variety of sources, such as established real-world benchmarks for the Steiner tree problem, data sets of common use in the field of road networks, and a synthetic data set where instances are generated at random.

Our experimental results have shown that our algorithm fits well the category of a general purpose Steiner tree heuristic since it was able to obtain good solutions in all benchmarked datasets when compared with other solvers. We note that the best solver in some datasets was built upon a state-of-the-art mixed-integer programming package. In some other datasets, the best solver was based on genetic algorithms. On the other hand, our algorithm essentially consists in the application of a single simple replacement routine that is applied multiple times until the time limit is reached. Still, the solutions obtained by our solvers were very competitive, often being the second best in the benchmarks and with a very small ratio $(v - b)/b$ where v is the weight of our solution and b a known lower bound or the weight of the best solver. It is also worth noting that our algorithm was able to handle graphs with millions of vertices, while most of the other solvers failed in all these big instances. Finally, it is worth noting that one possible application of our Steiner tree improvement sub-routine is as a black-box that can be used to improve the solution output by other solvers.

Intersection Non-emptiness for Star-Free Language Classes.
In Chapter 10, we present the work in [Arrighi et al., 2021b]. Here, we study the complexity of the INTERSECTION NON-EMPTINESS problem, one of the most fundamental and well studied problems in the interplay between algorithms, complexity theory, and automata theory [Kozen, 1977, Kasai and Iwata, 1985, Lange and Rossmanith, 1992, Wareham, 2000, Karakostas et al., 2003, Wehar, 2014, Fernau and Krebs, 2017, Wehar, 2016].

Problem name: INTERSECTION NON-EMPTINESS
Given: Finite automata $A_i = (Q_i, \Sigma, \delta_i, q_{(0,i)}, F_i)$, for $1 \leq i \leq m$.
Output: Is there a word w that is accepted by all A_i, i.e., is $\bigcap_{i=1}^{m} L(A_i) \neq \emptyset$?

In this paper, we analyze the complexity of the INTERSECTION NON-EMPTINESS problem when the input automata belong to some specific classes of languages belonging to two infinite hierarchies of languages, namely the Straubing-Thérien hierarchy [Place and Zeitoun, 2019, Straubing, 1981, Straubing, 1985, Thérien, 1981] and the Cohen-Brzozowski dot-depth hierarchy [Brzozowski, 1976, Cohen and Brzozowski, 1971, Place

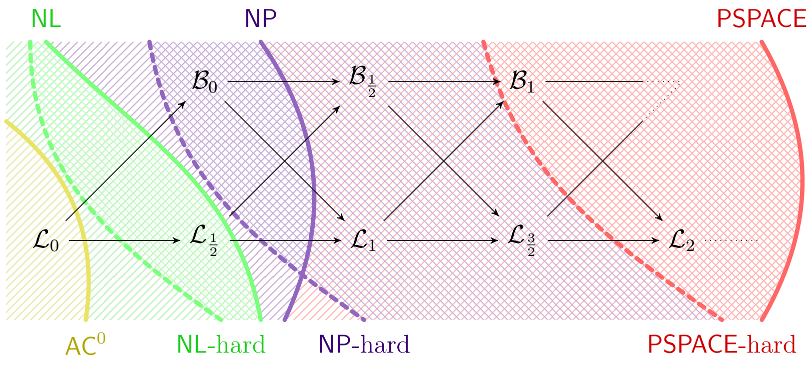


Figure 1.1: Complexity landscape of the Intersection Non-emptiness problem for NFAs for the lower levels of the Straubing-Thérien and dot-depth hierarchies.

and Zeitoun, 2019]. Those two hierarchies define very restricted classes of language in the sense that both hierarchies are included in the class of star-free languages. We show the complexity ranges from containment in AC^0 for the lowest level to PSPACE-completeness already in low levels of either hierarchies as depicted in Figure 1.1.

In more detail, we show that the Intersection Non-emptiness for NFAs and DFAs accepting languages from the level 1/2 of the Straubing-Thérien hierarchy are NL-complete and L-complete, respectively, under AC^0 reductions. Additionally, this completeness result holds even in the case of unary languages. This result is optimal in the sense that the problem becomes NP-hard even if we allow only a single DFA to accept a language from $\mathcal{L}_1$, and require all the others to accept languages from $\mathcal{L}_{1/2}$.

Subsequently, we analyze the complexity of Intersection Non-emptiness when all input automata are assumed to accept languages from one of the levels of $\mathcal{B}_0$ or $\mathcal{B}_{1/2}$ of the dot-depth hierarchy, or from the levels $\mathcal{L}_1$ or $\mathcal{L}_{3/2}$ of the Straubing-Thérien hierarchy. It is worth noting that NP-hardness follows straightforwardly from the fact that Intersection Non-emptiness for DFAs accepting finite languages is already NP-hard [Rampersad and Shallit, 2010]. Containment in NP, on the other hand, is a more delicate issue, and here the representation of the input automaton plays an important role. A characterization of languages in $\mathcal{L}_{3/2}$ in terms of languages accepted by partially ordered NFAs [Schwentick et al., 2001] is crucial for us. We show that the Intersection Non-emptiness is in NP for DFAs and partially ordered NFAs accepting languages from one of the levels of $\mathcal{B}_0$ or $\mathcal{B}_{1/2}$ of the dot-depth hierarchy, or from the levels $\mathcal{L}_1$ or $\mathcal{L}_{3/2}$ of the Straubing-Thérien hierarchy. For general NFAs, we show that the Intersection Non-emptiness problem is contained in NP for languages from the level $\mathcal{B}_0$ of the dot-depth hierarchy. The exact complexity for the level $\mathcal{B}_{1/2}$ of the dot-depth hierarchy or for the levels $\mathcal{L}_1$ or $\mathcal{L}_{3/2}$ of the Straubing-Thérien hierarchy is open.

Interestingly, we show that the proof technique used to prove this last result does not generalize to the context of NFAs. To prove this, we carefully design a sequence $(L_n)_{n\in\mathbb{N}_{\geq 1}}$ of languages over a binary alphabet such that for every $n \in \mathbb{N}_{\geq 1}$, the language L_n can be accepted by an NFA of size n, but any partially ordered NFA accepting L_n has size $2^{\Omega(\sqrt{n})}$. To the best of our knowledge, this is the first exponential separation between the state complexity of general NFAs and that of partially ordered NFAs. While this result does not exclude the possibility that Intersection Non-emptiness for languages in $\mathcal{L}_{3/2}$ represented by general NFAs is in NP, it gives some indication that proving such a containment requires substantially new techniques.

Finally, we show that Intersection Non-emptiness for both DFAs and for NFAs is already PSPACE-complete if all accepting languages are from the level $\mathcal{B}_1$ of the dot-depth hierarchy or from the level $\mathcal{L}_2$ of the Straubing-Thérien hierarchy.

Chapter 2

Fundamentals

In this chapter, we collect some basic notions, definitions and notation that we will use throughout this thesis.

2.1 Miscellaneous

We let $\mathbb{N}$ denote the set of natural numbers, including 0, and $\mathbb{N}_{>0}$ denote the set of positive natural numbers. For each $n \in \mathbb{N}_{>0}$, we let $[n] = \{1, \ldots, n\}$ denote the discrete interval of the first n positive integers. As a degenerate case, we let $[0] = \emptyset$. For each $n \in \mathbb{N}$, we let $[n]_0 = \{0\} \cup [n]$.

Given a finite set S, the size of S is the number of elements contained in S. We let $|S|$ denote the size of S. We let $\mathcal{P}(S)$ denote the set of all subsets of S. For each $k \in \mathbb{N}$, we let $\mathcal{P}(S, k)$ and $\mathcal{P}(S, \leq k)$ denote the sets of subsets of S of size exactly k and at most k, respectively.

Given two sets A and B, we let $A \cup B = \{x \mid x \in A \lor x \in B\}$ be the union of A and B, $A \cap B = \{x \mid x \in A \land x \in B\}$ be the intersection of A and B, and $A \setminus B = \{x \mid x \in A \land x \notin B\}$ be the set of elements of A not in B. We write $A \subseteq B$ for the set inclusion and $A \subset B$ for the strict set inclusion.

2.2 Graphs

Throughout this thesis, we work with undirected graphs without loops. Unless otherwise stated, graphs are simple. A (simple) *graph* G is a pair (V, E), where V is a finite set and $E \subseteq \mathcal{P}(V, 2)$ is a finite set, consisting of two-elementary sets of elements from V. Elements of V are called *vertices* and elements of E are called *edges*. Given an edge $e = \{u, v\} \in E$, we call u and v the endpoints of e. Given a vertex v, let $N(v) \doteq \{u \mid u \in V, \{v, u\} \in E\}$ be the neighbourhood of v. In the case of a simple graph, E can be seen as a binary relation over the vertices. In contrast, a *multigraph* is a graph in which there can be more than one edge between two vertices. In this case, E is a multiset. Given a graph G, we let $V(G)$ denote the *set of vertices* of G and $E(G)$ denote the *set of edges* of G.

An *isomorphism* from a graph G to a graph G' is a bijection $\varphi : V(G) \to V(G')$ such that for each $v, u \in V(G)$, $\{v, u\} \in E(G)$ if and only if $\{\varphi(v), \varphi(u)\} \in E(G')$. If such an isomorphism exists, we call G *isomorphic* to G'.

Given a subset $S \subseteq V(G)$, we let $G[S]$ be the *subgraph* of G *induced* by S. More precisely, $V(G[S]) = S$ and $E(G[S]) = E(G) \cap \mathcal{P}(S, 2)$. Let H be a graph, we say that G contains H as an induced subgraph if there exist a subset of vertices $S \subseteq V$ of G such that H is isomorphic to $G[S]$. We say that G is H-free or that G excludes H, if G does not contain H as an induced subgraph.

Interval graphs. Let $\mathcal{I} \doteq \{I_i \mid i \in [n]\}$ be a finite set of intervals over the reals. The *intersection graph* associated with $\mathcal{I}$ is the graph $G_{\mathcal{I}}$ with vertex set $V(G_{\mathcal{I}}) = \mathcal{I}$ and edge set $E(G_{\mathcal{I}}) = \{\{I_i, I_j\} \in \mathcal{P}(\mathcal{I}, 2) \mid I_i \cap I_j \neq \emptyset\}$.

Definition 1 (Interval graph). *A graph $G = (V, E)$ is an* interval graph *if there exists a set of interval $\mathcal{I} = \{I_u \mid u \in V\}$ such that G is isomorphic to the intersection graph $G_{\mathcal{I}}$ of $\mathcal{I}$.*

Figure 2.1 shows an example of interval graph with an interval representation.

2.3 Partial Orders

Let V be a set and ρ be a binary relation over V. We say that ρ is *reflexive* if for each $x \in V$, $(x, x) \in \rho$, antisymmetric if for each $x, y \in V$, $(x, y) \in \rho$ and $(y, x) \in \rho$ imply $x = y$ and transitive if for each $x, y, z \in V$, $(x, y) \in \rho$ and $(y, z) \in \rho$ imply $(x, z) \in \rho$. We call a relation σ *irreflexive* if for each $x \in V$, $(x, x) \notin \sigma$.

A *partial order* on V is a reflexive, anti-symmetric and transitive binary relation $\rho \subseteq V \times V$. We call a partial order ρ a *linear order* if additionally, for each $(x, y) \in V \times V$, either $(x, y) \in \rho$ or $(y, x) \in \rho$ holds. Linear orders are sometimes called total orders. A

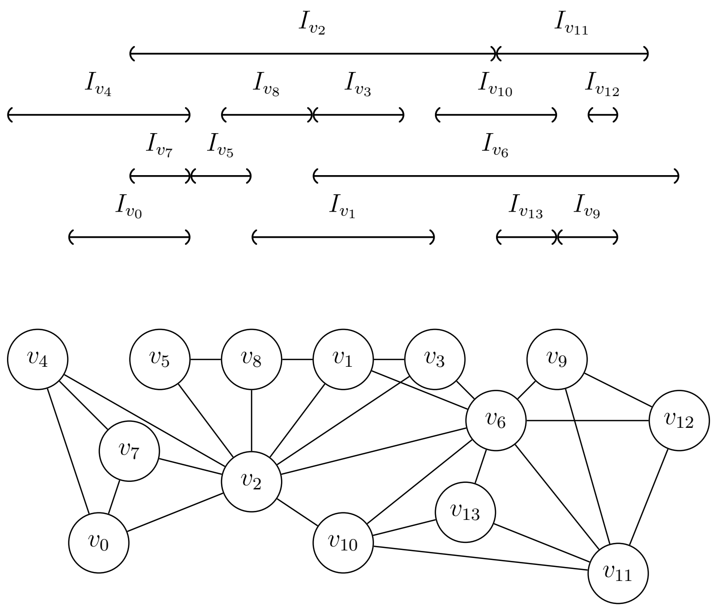


Figure 2.1: Example of an interval graph with an interval representation.

strict partial order on V is an irreflexive, antisymmetric and transitive binary relation $\sigma \subseteq V \times V$. By adding the identity relation I_V, $\rho \doteq \sigma \cup I_V$ becomes a partial order, and conversely, from a partial order ρ on V, we can define $\sigma \doteq \rho \setminus I_V$ as a strict partial order. If $\rho \subseteq V \times V$ is a partial order, then $<_\rho$ denotes the corresponding strict order. Hence, we will occasionally use the term linear order also for the corresponding strict order, often denoted as $<_\rho$ for reasons of clarity. Note that for a finite base set V, we can specify a linear order $\leq_f$ by a bijection $f : [|V|] \to V$, with the understanding that $f(i) \leq_f f(j)$ if and only if $i \leq j$, i.e., if the number i is smaller than the number j. Such a bijection f is also called a *ranking* in the following. Conversely, any linear order τ on Σ defines a bijection $f_\tau : [|V|] \to V$.

Let $\rho \subset V \times V$ be a partial order (respectively a strict partial order). For each $(x, y) \in \rho$, we say that x is smaller (respectively strictly smaller) than y, and denote this fact by $x \leq_\rho y$ (resp. $x <_\rho y$).

Given two partial orders ρ and τ on the same set V, we say that τ is an *extension* of ρ or that ρ is a *suborder* of τ if for each two elements x and y in V, $x \leq_\rho y$ implies that $x \leq_\tau y$. We denote this by $\rho \subseteq \tau$. If τ is also a linear order on V, then we say that τ is a *linear extension* of ρ. Given a set V, a (strict) partial order ρ on V and a subset $S \subseteq V$, we let $\min_\rho(S)$ be the set of minimal elements of S with respect to ρ and $\max_\rho(S)$ be the set of maximal elements of S with respect to ρ. For a linear order τ on V, the set

$\min_\tau(S)$ (respectively $\max_\tau(S)$) contains only one element, the minimum (respectively maximum) element in S with respect to τ. In this case, we identify, the sets $\min_\tau(S)$ and $\max_\tau(S)$ with their unique element. In other words, for a linear order τ, $\min_\tau(S)$ (resp. $\max_\tau(S)$) denotes the minimum (respectively maximum) element in S with respect to τ.

Given a set V, a subset $T \subseteq V$ and a (strict) partial order $\rho \subseteq V \times V$, we let $\rho|_T \doteq \rho \cap T \times T$ be the restriction of ρ to T. Let τ be a linear order of T. We say that τ is a *linear extension of* ρ *on* T if τ is a linear extension of $\rho|_T$. We define $\mathsf{Lin}(\rho, T)$ to be the set of linear extensions of ρ on T.

Given a binary relation α, we denote by $\mathbf{tc}(\alpha)$ the transitive closure of α.

Interval Orders. We recall the notions of an *interval order* as defined, e.g., in [Habib and Möhring, 1994].

Definition 2 (Interval order)**.** *An* interval order *is a strict partial order* $\iota \subseteq V \times V$ *over a set* V *whose elements* $v \in V$ *can be represented by half-open intervals* $I_v = [l_v, r_v)$ *on the reals with* $(u, v) \in \iota \iff r_u \leq l_v$. *We call* $\{I_v \mid v \in V\}$ *an* interval representation *of* ι.

For more information on interval orders, we refer to textbooks and survey articles such as [Fishburn, 1985, Trotter, 1997].

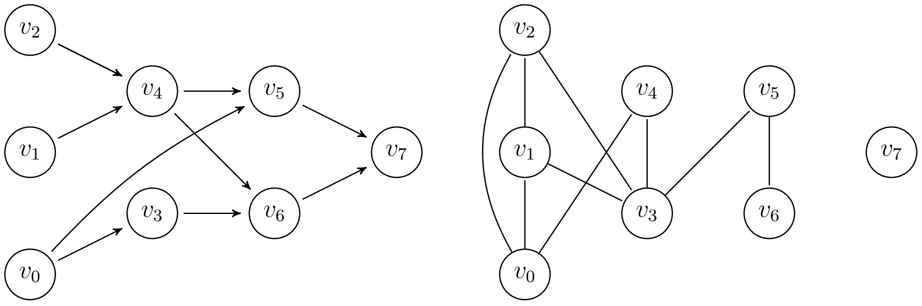
2.4 Graphs and Partial Orders

Next, we study partial orders through the lens of graph theory by using the concept of a *cocomparability graph.*

Definition 3 (Cocomparability graph)**.** *Given a (strict) partial order* $\rho \subseteq V \times V$*, we call the undirected graph* $G_\rho \doteq (V, E)$ *with* $E \doteq \{\{u, v\} \in \mathcal{P}(V, 2) \mid u \neq v, (u, v) \notin \rho, (v, u) \notin \rho\}$ *the* cocomparability graph *of* ρ.

Figure 2.2 shows an example of partial order with its cocomparability graph.

Let ι be an interval order on V and $\{I_v \mid v \in V\}$ be an interval representation of this order. The cocomparability graph G_ι of an interval order is an interval graph. Furthermore, G_ι is the intersection graph of $\{I_v \mid v \in V\}$. It was shown in [Gilmore and Hoffman, 1964a] that interval graphs are exactly the cocomparability graphs that do not contain an induced cycle of length four.



(a) A partial order ρ (b) Cocomparability graph of ρ

Figure 2.2: Example of an partial order and its cocomparability graph. For clarity, in the partial order, transitive edges are not shown.

Remark 4. Habib and Möhring [Habib and Möhring, 1994] define cocomparability graphs for strict partial orders σ on V, i.e., for transitive irreflexive relations. By the known tight connections between partial orders and strict partial orders, one sees that the cocomparability graph defined in [Habib and Möhring, 1994] is the same object as the one according to our definition. In particular, we also use the notation G_σ to denote the cocomparability graph of a strict partial order σ.

Chapter 3

Computational Complexity

Computational complexity theory classifies problems according to the resources needed by an algorithm to solve them. Formally, the classification is made using a model of computation called *Turing machine.* As we do not need the concept of Turing machine in the rest of the thesis, we will just talk about algorithms.

In this section, we give some notations and definitions of the complexity classes that are used in this thesis. We assume familiarities with the computational complexity theory and the big $\mathcal{O}$ notation, we refer the reader to [Arora and Barak, 2009, Sipser, 1997] for a more detailed introductions on computational complexity theory and to [Papadimitriou, 2007, Garey and Johnson, 1979, Page 6] for an introduction to the big $\mathcal{O}$ notation.

3.1 Classical Time Complexity Classes: P and NP

In this section we recall the useful definitions of classical complexity theory for this thesis. First, we start with the notion of decision problems. A *decision problem* is a problem with two possible answers, YES or NO. For example, the question, "Does the input graph have a vertex cover of size at most k?" is a decision problem. Given a fixed finite alphabet Σ, a *decision problem* is formally defined as a language $L \subseteq \Sigma^*$. An *instance* or *input* of a problem L is a word $x \in \Sigma^*$. Intuitively, L is the set of positive instances of the problem. An *algorithm* that solves L is a procedure that given an input $x \in \Sigma^*$ determines if $x \in L$ or not.

Definition 5 (Time Bounds)**.** *Given an algorithm $\mathcal{A}$ and a function $f : \mathbb{N} \to \mathbb{N}$. If $\forall n \in \mathbb{N}, \forall x \in \Sigma^n$, $\mathcal{A}$ terminates in less than $f(n)$ steps on input x, we say that $\mathcal{A}$ is* time bounded *by f. Or similarly, that the* running time *of $\mathcal{A}$ is bounded by f.*

We express time bounds using the big $\mathcal{O}$ notation. In addition to the classical big $\mathcal{O}$ notation, we will also use $\mathcal{O}^\star$-notation. $\mathcal{O}^\star$ suppresses the polynomial factor. For example, $\mathcal{O}(n^n \cdot n^{\mathcal{O}(1)}) \subseteq \mathcal{O}^\star(n^n)$ or $\mathcal{O}(k \cdot 2^k \cdot n^{\mathcal{O}(1)}) \subseteq \mathcal{O}^\star(2^k)$. This notation is used to highlight the dependency in the parameter for FPT algorithms. The class FPT and parameterized complexity is introduced in Section 3.2. We now recall the definition of the class P of problems solvable in (deterministic) polynomial time and the class NP of problems solvable in non-deterministic polynomial time.

Definition 6 (P: Deterministic Polynomial-Time)**.** *The class* P *is the class of computational problems* $L \subseteq \Sigma^*$ *for which there exist an algorithm* $\mathcal{A}$ *solving* L*, which running time is bounded by a polynomial function* f*.*

Definition 7 (NP: Non-deterministic Polynomial-Time)**.** *The class* NP *is the class of computational problems* $L \subseteq \Sigma^*$ *for which there exist a non-deterministic algorithm* $\mathcal{A}$ *solving* L*, which running time is bounded by a polynomial function* f*.*

Next, we recall the notion of hardness with respect to a complexity class. In our case, we are interested in NP-hardness. This notion is based on the concept of a reduction. Intuitively, a reduction expresses that if we can solve a computational problem L_1 using an algorithm for a computational problem L_2, then L_1 is easier than L_2.

Definition 8 (Polynomial-Time Many-One Reduction)**.** *Let* $L_1 \subseteq \Sigma^*$ *and* $L_2 \subseteq \Gamma^*$ *be two computational problems. A* polynomial-time many-one reduction *from* L_1 *to* L_2*, is a function* $f : \Sigma^* \to \Gamma^*$ *that can be computed in polynomial time such that*

$$\forall x \in \Sigma^*, x \in L_1 \Leftrightarrow f(x) \in L_2.$$

The polynomial-time many-one reduction is not the only notion of reduction used in complexity theory, but it is a useful definition for working with problems in NP. Using reductions, we can order problems. We note $L_1 \leq L_2$ if L_1 reduces to L_2. Given a problem L if, for every $L_i \in \mathsf{NP}$, $\mathsf{L}_i \leq L$, this means that every problem in NP reduces to L, then we call L NP-hard. If in addition $L \in \mathsf{NP}$, then L is called NP-complete.

Example 9 (Vertex Cover)**.** To illustrate the introduced concepts, we use the Vertex Cover problem (VC for short) as an example throughout this chapter.

Problem name: Vertex Cover (VC)
Given: A graph G and a non-negative integer k.
Output: Does there exist a set $S \subseteq V(G)$, such that $G[V(G) \setminus S]$ is edgeless?

The Vertex Cover problem is a well known and well-studied problem in the field of parameterized complexity. First, VC is an example of an NP-complete problem [Karp, 1972]. The hardness can be shown by a reduction from the 3-SAT problem or from the Clique problem as in [Karp, 1972]. Note that the SAT problem, asking for the satisfiability of a Boolean formula, was the first problem to be shown NP-complete by a direct reduction form the halting problem for polynomial-time bounded Turing machines, performed by Cook [Cook, 1971]. An algorithm to show membership in NP works as follow, it first non-deterministically guesses a set of k vertices that cover all edges and then verifies that this set indeed covers all the edges. We will have a deeper look in the complexity of VC in the following when we introducing the other concepts of this chapter.

3.2 Parameterized Complexity

Many problems with practical applications are NP-complete. Unless P = NP, those problems cannot be solved by an algorithm running in polynomial time. This means that we cannot have exact and efficient algorithms to solve them. Once we have proven that a problem is hard, there are several approaches to go beyond this hardness. One can use heuristics which are algorithms that work well in practice but without any formal guarantee on the solution found, or approximation algorithms that have a formal guarantee how far is the solution given from the optimal one, or one can look at variations of the problem that may be easier to solve. The method we are interested in here is to consider the parameterized complexity of a problem. The theory of *parameterized complexity* was introduced by Downey and Fellows [Downey and Fellows, 1999]. It extends the framework of complexity theory to go beyond NP-hardness results. The key idea is to remark that classical complexity theory looks at the worse case scenario. In some sense classical complexity theory tries to answer the following question: if an adversary gives us the worse possible instance of a problem, how much time or memory do we need to solve it? From a practical point of view, we want to solve some specific instances of the problem, not the worse possible ones. The idea of Downey and Fellows is to consider additional parameters in addition to the size of the input. Therefore, parameterized algorithms can be used to solve problems on instances for which the value of the parameter is small. For a more broad introduction to the field of parameterized complexity, we refer the reader to [Downey and Fellows, 2013, Cygan et al., 2015, Flum and Grohe, 2006]

In this thesis, we will only use the parameterized class called fixed-parameter tractable, FPT for short. To be able to define it, we first need to generalize the notion of computational problems, to *parameterized* computational problems.

Definition 10 (Parameterized computational problem). *Given a fixed finite alphabet Σ, a* parameterized computational problem *is a language $L \subseteq \Sigma^* \times \mathbb{N}^p$ for some $p \in \mathbb{N}_{>0}$. For an instance $(x, (k_1, k_2, \ldots, k_p)) \in L$, $(k_1, k_2, \ldots, k_p)$ are called the parameters of the instance.*

Traditionally, parameterized computational problems are defined with only one parameter, $p = 1$ in our definition. Both definitions are equivalent, from an instance $(x, (k_1, k_2, \ldots, k_p)) \in \Sigma^* \times \mathbb{N}^p$, one construct an instance with only one parameter by taking $(x, \sum_{i=1}^{p} k_i)$. In this thesis, some problems will naturally have several parameters, therefore we choose to explicitly state the definition with several parameters.

Example 11 (Parametrizations of the Vertex Cover problem). A computational problem can have several parameterizations. In the case of Verter Cover, for an instance (G, k), a parameter could be k, the maximum size of the vertex cover. Other parameters can come from structural properties of the input. For example, in the case of VC, parameters can be defined by a property of the input graph G. Examples of such parameters for graph problems are maximum degree of the input graph, treewidth or pathwidth of the input graph, or the length of the longest cycle in the input graph. For instance, the treewidth can be used as a parameter for VC. In this case, let L be the language defining graphs that have a vertex cover of size at most k, with parameter the treewidth. An instance $((G, k), w)$, with (G, k) being an instance of the Vertex Cover problem, and w being the parameter, belongs to the language L if G has treewidth at most w and G has a vertex cover of size at most k.

Now, we are ready to define the class of fixed-parameter tractable problems called FPT.

Definition 12 (FPT: Fixed-Parameter Tractable). *The class* FPT *is the class of parameterized computational problems $L \subseteq \Sigma^* \times \mathbb{N}^p$ for which there exist an algorithm $\mathcal{A}$, called a* fixed-parameter algorithm*, such that, $\forall n \in \mathbb{N}$ on input $(x, (k_1, k_2, \ldots, k_p)) \in \Sigma^n \times \mathbb{N}^p$, $\mathcal{A}$ is time bounded by $f(k_1, k_2, \ldots, k_p) \cdot n^c$, where f is any function and c a constant.*

We say that a fixed-parameter algorithm runs in FPT-time. We can also generalize the notion of reductions to parameterized computational problems.

Definition 13 (FPT Many-One Reduction). *Let $L_1 \subseteq \Sigma^* \times \mathbb{N}^p$ and $L_2 \subseteq \Gamma^* \times \mathbb{N}^q$ be two parameterized computational problems. A* FPT many-one reduction *from L_1 to L_2, is a function $\mathcal{R} : \Sigma^* \times \mathbb{N}^p \to \Gamma^* \times \mathbb{N}^q$ such that*

- *$\mathcal{R}(x, (k_1, k_2, \ldots, k_p))$ can be computed in time $f(k_1, k_2, \ldots, k_p) \cdot |x|^{\mathcal{O}(1)}$ for some function f.*

- $\forall(x,k) \in \Sigma^* \times \mathbb{N}^p, (x,(k_1,k_2,\dots,k_p)) \in L_1 \Leftrightarrow \mathcal{R}(x,(k_1,k_2,\dots,k_p)) \in L_2$,
- $\forall(x,(k_1,k_2,\dots,k_p)) \in \Sigma^* \times \mathbb{N}^p$, *if* $\mathcal{R}(x,(k_1,k_2,\dots,k_p)) = (x',(k'_1,k'_2,\dots,k'_q))$, *then* $\forall i \in [q], k'_i \leq g(k_1,k_2,\dots,k_p)$ *for some computable function* g.

Example 14 (FPT results for the Vertex Cover problem)**.** Parameterized results need to be specified for a particular parameter. In the case of Vertex Cover, we have seen in Example 11 several possible parameters. Here, we give results for some of them. We start with the parameter k, the size of the vertex cover. A simple branching algorithm can solve VC in time $2^k \cdot n^{\mathcal{O}(1)}$, where k is the size of the vertex cover. Therefore, the Vertex Cover problem parameterized by the size of the solution is in FPT. In the case of treewidth or pathwidth, a dynamic programming algorithm can solve VC in time $2^w \cdot n^{\mathcal{O}(1)}$, where w is the treewidth (respectively pathwidth) of the input graph. We give a more detailed description of the algorithm in the case of pathwidth in Example 18.

In classical complexity theory, conditional hardness results are often obtained under the assumption that P $\neq$ NP. In parameterized complexity theory, there exist several hypotheses commonly used to give conditional lower bounds. The one used in this thesis is called the Exponential Time Hypothesis, ETH for short, and was introduced by Impagliazzo and Paturi [Impagliazzo and Paturi, 2001]. It is based on an assumption concerning the 3-SAT problem. We refer the reader to [Lokshtanov et al., 2011] for more information.

Definition 15 (ETH: Exponential Time Hypothesis)**.** *There is an* $\epsilon > 0$ *such that* 3-SAT *on* n *variables cannot be solved in time* $\mathcal{O}^\star(2^{\epsilon n})$.

Example 16 (ETH lower bound for VC)**.** We can use the Exponential Time Hypothesis assumption to obtain a conditional lower bound for VC stating that one of the FPT-results we gave in Example 14 is optimal under ETH. Namely, under ETH, there is no algorithm running in time $2^{o(k)} \cdot n^{\mathcal{O}(1)}$ for VC parameterized by the size of the solution.

Chapter 4

Decompositions and Width Measures

Structural parameters are fundamental in the design of parameterized algorithms. Also the algorithms presented in this thesis will rely on structural parameters. Therefore, we present in this chapter different structural parameters based on width of decompositions. Those width measures will span from graphs over partial orders to strings.

4.1 Width Measures for Graphs

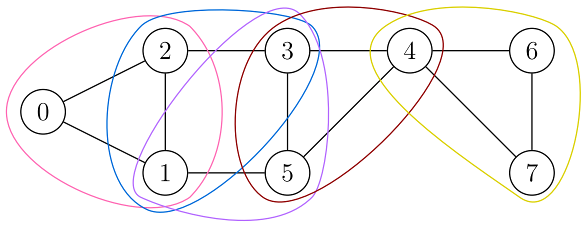
4.1.1 Treewidth and Pathwidth

We start our tour of width measures for undirected graphs by introducing the notions of *treewidth* and *pathwidth*. Treewidth is arguably one of the most studied and used width measures on undirected graphs. It also plays a crucial role in parameterized complexity theory [Cygan et al., 2015, Downey and Fellows, 2013]. Intuitively, the treewidth of a graph measures how far it is from being a tree. The pathwidth is the analogous notion from paths instead of trees. For an overview on the treewidth and pathwidth, we refer the reader to the survey by Bodlaender [Bodlaender, 2012].

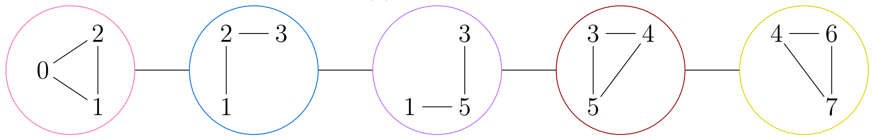
The definition of treewidth is based on the notion of tree decomposition.

Definition 17 (Tree decomposition)**.** *A* tree decomposition *of a graph $G = (V, E)$ is a pair $(\mathcal{T}, \mathcal{B})$ where $\mathcal{T}$ is a tree and $\mathcal{B} = \{B_t \subseteq V \mid t \in V(\mathcal{T})\}$ is a set of subsets of V called* bags*, such that the following conditions are satisfied.*

- $\bigcup_{t \in V(\mathcal{T})} B_t = V$.
- *For each edge $\{u, v\} \in E$, there is an $t \in V(\mathcal{T})$ such that $u, v \in B_t$.*



(a) A graph G.



(b) A path decomposition of G of with 2.

Figure 4.1: Example of a graph (4.1a) and its path decomposition of width 2 (4.1b)

- *For each $v \in V$, the subtree $\mathcal{T}[\{t \in V(\mathcal{T}) \mid v \in B_t\}]$ is connected.*

The *width* of a tree decomposition $(\mathcal{T}, \mathcal{B})$ is defined as $w((\mathcal{T}, \mathcal{B})) = \max_{t \in V(\mathcal{T})} |B_t| - 1$. The *treewidth*, $tw(G)$, of G is the minimum width of a tree decomposition of G.

A *path decomposition* is a tree decomposition $(\mathcal{T}, \mathcal{B})$ where $\mathcal{T}$ is a path. For simplicity, we write a path decomposition as a sequence of bags $D = (B_1, B_2, \ldots, B_l)$. The *width* of a path decomposition D is defined as $w(D) = \max_{p \in [l]} |B_p| - 1$. Analogously, the *pathwidth*, $pw(G)$, of G is the minimum width of a path decomposition of G.

In this thesis, we will focus on pathwidth and path decomposition for two reasons. First in Chapter 6 and Chapter 7, we are interested in cocomparability graphs associated to some partial order. For this class of graphs, Habib and Möhring [Habib and Möhring, 1994] showed that the notions of pathwidth and treewidth are equivalent. In Chapter 8, despite the fact that we do not use path decompositions to devise algorithms, we give give a structural result about pathwidth using an equivalent definition from Subsection 4.1.2 based on a linear order on the set of vertices. Figure 4.1 shows an example of a graph with its path decomposition.

Path Decompositions. Let $D = (B_1, B_2, \ldots, B_l)$ be a path decomposition of a graph G. We say that $[l]$ is the set of *positions* of D and that l is the *length* of D. For each position $p \in [l]$, we say that B_p is the *p-th bag* of D. For each position $p \in \{2, \ldots, l\}$, and for each vertex $v \in B_p \setminus B_{p-1}$, we say that B_p *introduces* v (or that v is *introduced* by B_p) and for each vertex $v \in B_{p-1} \setminus B_p$, we say that B_p *forgets* v (or that v is *forgotten* by B_p).

For a position $p \in [l]$, we write intro(p) (respectively forg(p)) for the set of all vertices introduced (respectively forgotten) by B_p, and we let $L_p = \bigcup_{1 \leq i \leq p} \text{forg}(p)$ be the set of vertices that have been forgotten up to position p. For each $p \in [l]$, $p > 1$, we say that B_p is an *introduce bag* if $B_p = B_{p-1} \cup \{v\}$ and that B_p is a *forget bag* if $B_p = B_{p-1} \setminus \{v\}$. We say that the path decomposition $D = (B_1, B_2, \ldots, B_l)$ is *nice* if for each $p \in [l]$, B_p is either an introduce bag or a forget bag, $|B_1| = 1$ and $B_r = \emptyset$. In a nice path decomposition, we say that $B_1 = \{v\}$ introduces v, and therefore B_1 is an introduce bag. It can be shown that, given any path decomposition $D = (B_1, B_2, \ldots, B_r)$ of width w of a graph G, one can construct in time $\mathcal{O}(l \cdot w(D))$ a nice path decomposition of G of width at most w. In a nice path decomposition, for every vertex of V, there is a bag that introduces it and a bag that forgets it, so the length of a nice path decomposition is $2 \cdot |V|$. For each position $p \in [l]$, we let $L_p = \bigcup_{1 \leq i \leq p-1} B_i \setminus B_p$ be the set of vertices that have been forgotten (**l**ost) up to position p.

Some complexity results. We give a few words on the complexity of deciding if a graph has pathwidth at most w. We call this problem the Pathwidth problem. Deciding if a graph has pathwidth at most w is NP-complete [Lengauer, 1981]. From the parameterized complexity point of view, the Pathwidth problem can be solved in time $2^{\mathcal{O}(w^2)} \cdot n^{\mathcal{O}(1)}$ where w is the pathwidth of the input graph [Downey and Fellows, 1999].

We now demonstrate, how a nice path decomposition can be used to design FPT algorithms.

Example 18 (VC parameterized by pathwidth)**.** We continue our series of examples using the Vertex Cover problem. Here, we give an algorithm that, given a graph G and a nice path decomposition $D = (B_1, B_2, \ldots, B_l)$ of width w, solves VC in time $2^w \cdot |V(G)|^{\mathcal{O}(1)}$. Intuitively, the algorithm processes the path decomposition one bag at a time. For each bag B_p, the algorithm builds a set of minimal solutions for $G[\cup_{i \leq p} B_i]$ and stores enough information to process the next bag. Formally, for each position $p \in [l]$, the algorithm constructs a table T_p. For each $S \subseteq B_p$, $T_p(S)$ contains the size of a minimum vertex cover X of $G[\cup_{i \leq p} B_i]$ such that $X \cap B_p = S$. We use the convention that $T_p(S) = +\infty$, if S is not a vertex cover of $G[B_p]$. Note that, for each bag B_p, there are at most 2^w subsets S of B_p. Then, the smallest entry in T_l, gives the size of a minimum vertex cover of G.

As D is a nice path decomposition, to describe the algorithm, we need to specify its behaviour on the first bag, for an introduce bag and for a forget bag.

- For $B_1 = \{v\}$, there are two cases $T_1(\emptyset) = 0$ and $T_1(B_1) = 1$.
- For a position $p > 1$, given the table T_{p-1}, we explain how to build T_p.

- If B_p forgets v, then, for each $S \subseteq B_p$, $T_p(S) = \min(T_{p-1}(S), T_{p-1}(S \cup \{v\}))$.
- If B_p introduces v, then, for each $S \subseteq B_p$, if $v \notin S$ and S is a vertex cover of B_p, then we set $T_p(S) = T_{p-1}(S)$, otherwise, if S is not a vertex cover, we set $T_p(S) = +\infty$. If, on the other hand, $v \in S$, then we set $T_p(S) = T_{p-1}(S \setminus \{v\}) + 1$.

For each bag B_p and subset $S \subseteq B_p$, the algorithm computes $T_p(S)$ in time $|V(G)|^{\mathcal{O}(1)}$. As D is a nice path decomposition, it follows that $l \in \mathcal{O}(|V(G)|)$. So the total running time of the algorithm is $2^w \cdot |V(G)|^{\mathcal{O}(1)}$.

One can easily check that the algorithm will build a minimum vertex cover after processing the last bag B_l.

4.1.2 Vertex Separation Number

Here, we define the vertex separation number of a graph, a notion that is equivalent to pathwidth [Kinnersley, 1992]. The *vertex separation number* is a width measure for graphs based on a specific linear order of the vertices of the graph.

Let G be an n-vertex graph with vertex set $V(G)$ and edge set $E(G)$. Given sets $S, S' \subseteq V(G)$, we let $V(G, S, S') = \{u \in S \ : \ \exists v \in S' \colon \{u, v\} \in E(G)\}$ be the set of vertices in S that are adjacent to some vertex in S'. As a special case, we define $V(G, S) = V(G, S, V(G) \setminus S)$.

Definition 19 (Vertex Separation Number). *Let G be an n-vertex undirected graph with vertex set $V(G)$ and edge set $E(G)$. Let $\tau : [n] \to V(G)$ be a linear order on the vertices of G. For each $p \in [n]$, we let*

$$\mathrm{vsn}(G, \tau, p) = |V(G, \tau([p-1]))| = |\{l \in [p-1] \mid \exists r \geq p \text{ such that } \{\tau(l), \tau(r)\} \in E(G)\}|.$$

The vertex separation number *of the linear order τ is defined as* $\mathrm{vsn}(G, \tau) = \max_{p \in [n]} \mathrm{vsn}(G, \tau, p)$. *Finally, the* vertex separation number *of G is defined as* $\mathrm{vsn}(G) = \min_\tau \mathrm{vsn}(G, \tau)$, *where τ ranges over all linear orders on the vertex set V.*

4.1.3 Cutwidth

One of the classical layout parameters for graphs is the so called *cutwidth* of a graph. Intuitively, the cutwidth of a graph describes the maximum number of edges connecting

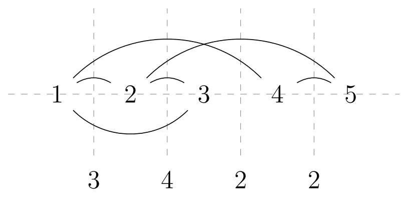


Figure 4.2: Visual representation of the cutwidth of a graph. For simplicity, τ is the identity function. In this example, the cutwidth is 4.

vertices of any prefix to the complement suffix of a linear order on the vertices of the graph. Cutwidth turned out to be a useful parameter in parameterized algorithms and is used among other things in circuit layout [Adolphson and Hu, 1973, Makedon and Sudborough, 1989], network reliability [Karger, 2001], graph drawing [Mutzel, 1995] and information retrieval [Botafogo, 1993]. For more detailed information, we refer the reader to [Díaz et al., 2002, Petit, 2011].

Let G be an n-vertex graph. Given sets $S, S' \subseteq V(G)$, we let $E(G, S, S') = \{\{u, v\} \in E(G) \ : \ u \in S, \ v \in S'\}$ be the set of edges with one endpoint in S and the other endpoint in S'. As a special case, we define $E(G, S) = E(G, S, V(G) \setminus S)$. The following two properties will be of importance in Chapter 8.

- **Monotonicity property:** If $T \subseteq S$ and $T' \subseteq S'$, then $E(G, T, T') \subseteq E(G, S, S')$.
- **Linearity property:** If $\{S_1, S_2\}$ is a partition of S, then $\{E(G, S_1, S'), E(G, S_2, S')\}$ is a partition of $E(G, S, S')$.

Let $\tau : [n] \to V(G)$ be a linear order on the vertices of G. For each $p \in [n]$, we let $\mathrm{cw}(G, \tau, p) = |E(G, \tau([p-1]))|$ be the number of edges that have one endpoint in the first $p-1$ vertices of the linear order τ and the other endpoint in the remaining vertices. $E(G, \tau([p-1]))$ is called the *cut* at position p and $\mathrm{cw}(G, \tau, p)$ is the *size of the cut* at position p.

Definition 20 (Cutwidth)**.** *The* cutwidth *of the linear order τ is defined as* $\mathrm{cw}(G, \tau) = \max_{p \in [n]} \mathrm{cw}(G, \tau, p)$. *The* cutwidth *of the graph G is defined as* $\mathrm{cw}(G) = \min_\tau \mathrm{cw}(G, \tau)$, *where τ ranges over all linear orders on the vertex set $V(G)$.*

Given a graph G, the cutwidth of a given linear order τ can be intuitively understood by drawing G in a specific way on the plane. First, the vertices of G are placed on a horizontal line following the order given by τ. Then, edges are drawn as curves between the point representing their endpoints. Now, if we draw a vertical line between the $(p-1)$-th and p-th vertices in τ, then the number of edges that intersect this vertical line corresponds to $\mathrm{cw}(G, \tau, p)$. Figure 4.2 gives an example of such a visual representation of the cutwidth.

We give a few words on the complexity of deciding if a graph has cutwidth at most w. We call this problem the CUTWIDTH problem, it is also known as the MINIMUM CUT LINEAR ARRANGEMENT problem in the literature. Deciding if a graph has cutwidth at most k is NP-complete [Garey and Johnson, 1979]. From the parameterized complexity point of view, the CUTWIDTH problem can be solved in time $2^{\mathcal{O}(w^2 \log w)} \cdot n$ where w is the cutwidth of the input graph [Giannopoulou et al., 2019]. Recently, Casel *et al.* [Casel et al., 2019] relate the cutwidth of a graph to it pathwidths. The relation can be used to give a better approximation algorithm for cutwidth.

4.2 Width Measures for Partial Orders

Next, we look at a width measure for partial orders called *interval width*, introduced by Habib and Möhring [Habib and Möhring, 1994]. Intuitively, this parameter measures how far a partial order is from a linear order. Even if this width measure can be defined purely in terms of partial orders, we will see that this notion is related to a width measure on graph.

Definition 21 (Interval width). *The* interval width *of a partial order $\rho \subseteq V \times V$ is defined as* $\mathrm{iw}(\rho) \doteq \min\{w(\iota) | \iota$ *interval order*$, \iota \subseteq \rho\}$*, where $w(\iota)$ is the maximum size of an antichain of ι.*

The definition of the interval width of a partial order is similar to the definition of pathwidth using the size of the biggest clique of interval graphs. And the resemblance does not stop here, Habib and Möhring [Habib and Möhring, 1994] show that the interval width of a partial order ρ is equal to the pathwidth of the cocomparability graph G_ρ of ρ plus 1.

Lemma 22 (Theorem 1.2 in [Habib and Möhring, 1994]). *Let ρ be a partial order. Then, $pw(G_\rho) = iw(\rho)$.*

Hence, similar to the interval width, the pathwidth of the cocomparability graph of a partial order may be regarded as a measure of how close the order is from being a linear order. The cocomparability graph of a linear order τ on n elements is the graph with n vertices and no edges. This graph has pathwidth 0. On the other hand, if τ is a partial order where all n elements are unrelated, then the cocomparability graph of τ is

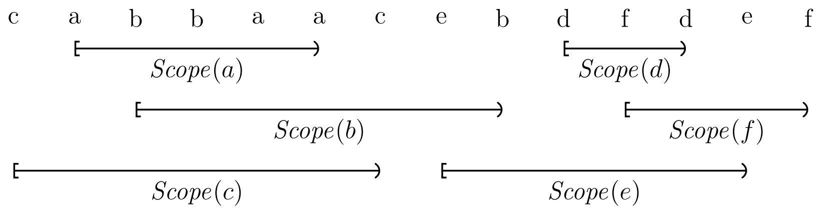


Figure 4.3: $w = cabbaacebdfdef$ has scope coincidence degree 3. The intervals show the scope of each letter. The set of intervals can be interpreted as the interval representation of some interval graph G_w. The pathwidth of G_w is equal to $SCD(w) - 1 = 2$.

the n-clique, which has pathwidth $n - 1$ (the highest possible pathwidth in an n-vertex graph).

Remark 23. In [Habib and Möhring, 1994], *interval width* of strict partial orders was defined. However, as interval orders are strict partial orders and hence are irreflexive, a partial order ρ extends an interval order ι if and only if the corresponding strict partial order $\sigma = \rho \setminus I_V$ extends ι. Hence, the interval width of ρ equals the interval width of σ.

4.3 Width Measures for Strings

Finally, we turn our attention to width measures for strings by adapting a parameter called *scope coincidence degree.* The concept of scope coincidence degree (SCD for short) was introduced in [Reidenbach and Schmid, 2013] for patterns, which are strings over two disjoint alphabets, where only the alphabet of variables was used to measure the SCD of patterns. We adapt the definition from [Reidenbach and Schmid, 2014] in the following to strings over a single alphabet.

Definition 24 (Scope Coincidence Degree)**.** *Given a string $w \in \Sigma^*$, and a letter $a \in \Sigma$, the* scope *of a, denoted $Scope(a)$ is the set of positions in $\{1, \dots, |w|\}$ between the minimum position and the maximum position in which a occurs. For each position i, we let the incidence set of i to be $Inc(i) = \{a \in \Sigma \ : \ i \in Scope(a)\}$. Now, the scope coincidence degree is the number of overlapping scopes for all letters. In other words, we have that $SCD(w) = \max_i |Inc(i)|$.*

Given a string $w \in \Sigma^*$, we can define an interval graph G_w where $V(G_w) = \Sigma$ and the representation of $V(G_w)$ with intervals is given by the scope of the letters in w. The pathwidth of G_w is one less than the scope coincidence degree of w. This connection with pathwidth will be used in a reduction later in Section 6.6. Figure 4.3 illustrates this connection for the word $w = cabbaacebdfdef$.

The connection between the scope coincidence degree of a string w and the pathwidth of a interval graph, implies that the scope coincide degree can be computed in linear times.

Despite the fact that we will be only interested in the scope coincidence degree as a width measure for strings in the following, we quickly want to mention that there exist also other interesting width measures for strings such as the *locality number.* Recently, Casel *et al.* [Casel et al., 2019] related the locality number to some width measure for graphs, namely cutwidth and the pathwidth.

Chapter 5

Problem Definitions

In this chapter, we define the main problems of interest for this thesis which will be studied in detail in Chapter 6, Chapter 7 and Chapter 8. The problems span over a variety of different fields of computer science, such as ordering, graph drawing, social choice, strings, reconfiguration, graph theory and rewriting systems. All these problems may seem very different, but they all have a direct or indirect link to linear orders which is the topic of this thesis.

5.1 Completion of an Ordering (CO) and its Variations

In this section, we introduce the Completion of an Ordering problem and two variations of this problem. The Completion of an Ordering problem is a generalization of the Positive Completion of an Ordering (PCO) problem which was originally introduced in [Dujmovic et al., 2003, Sec. 8] and in more details in [Fernau, 2005, Sec. 6.4].

Problem name: Completion of an Ordering (CO)
Given: A partial order $\rho \subseteq V \times V$ over a set V, a cost function $\mathfrak{c} : V \times V \to \mathbb{N}$, and a non-negative integer $k \in \mathbb{N}$.
Output: Is there a linear order $\tau \supseteq \rho$ with $\mathfrak{c}(\tau \setminus \rho) = \sum_{(x,y)\in\tau\setminus\rho} \mathfrak{c}(x, y) \leq k$?

Intuitively, given a partial order ρ and a cost function $\mathfrak{c}$, the goal is to find a linear extension of ρ incurring a cost of at most k. The only difference between CO and the original PCO problem introduced in [Dujmovic et al., 2003, Fernau, 2005] is that, in the latter, the cost function needs to satisfy the following condition: for every pair $(x, y) \in C \times C$ such that x and y are incomparable in ρ, the cost of (x, y) is strictly positive ($\mathfrak{c}(x, y) > 0$). In CO, for such a pair, the cost can be zero ($\mathfrak{c}(x, y) \geq 0$). Formally, the Positive Completion of an Ordering problem is defined as follow.

Problem name: Positive Completion of an Ordering (PCO)
Given: A partial order $\rho \subseteq V \times V$ over a set V, a cost function $\mathfrak{c} : V \times V \to \mathbb{N}$ satisfying $\forall x, y \in V : ((x, y) \notin \rho \wedge (y, x) \notin \rho) \implies \mathfrak{c}(x, y) > 0$, and a non-negative integer $k \in \mathbb{N}$.
Output: Is there a linear order $\tau \supseteq \iota$ with $\mathfrak{c}(\tau \setminus \iota) \leq k$?

Let us shortly discuss the *cost parameter* k. By the result of Dujmovic, Fernau and Kaufmann [Dujmovic et al., 2003] (for details, see [Fernau, 2005]), PCO can be solved in time $\mathcal{O}^\star(1.52^k)$ and admits a linear-size kernel. The best-known algorithm for PCO, whose running time is $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k}\log(k))})$, was obtained in [Fernau et al., 2014] by relating PCO to the Feedback Arc Set in Tournaments problem, or FAST for short, that allows for subexponential algorithms due to [Alon et al., 2009]. In Subsection 6.3.2, we are going to improve on these algorithms by presenting an algorithm that runs in time $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ and that is relatively straightforward to implement. As branching algorithms are sometimes better in practice, we also present in Subsection 6.3.4 a branching algorithm that runs in time $\mathcal{O}^\star(1.42^k)$, improving on the one explained in [Fernau, 2005]. Our algorithms are based on the interval width of ρ.

When we later reduce the introduced problems to each other, PCO on interval orders appears naturally, and we can use the extra structure to improve the running time of our algorithm.

Problem name: Positive Completion of an Interval Ordering (PCIO)
Given: An interval order $\iota \subseteq V \times V$ over a set V, a cost function $\mathfrak{c} : V \times V \to \mathbb{N}$ satisfying $\forall x, y \in V : ((x, y) \notin \iota \wedge (y, x) \notin \iota) \implies \mathfrak{c}(x, y) > 0$, and a non-negative integer $k \in \mathbb{N}$.
Output: Is there a linear order $\tau \supseteq \iota$ with $\mathfrak{c}(\tau \setminus \iota) \leq k$?

This variation has two more restrictions compared to CO: the cost between two incomparable elements must not be zero and the partial order is an interval order. Those two restrictions will enable us to have a finer analysis of the complexity of our algorithm.

5.2 One-Sided Crossing Minimization (OSCM)

Next, we define and discuss the One-Sided Crossing Minimization problem from the field of graph drawing. Given a bipartite graph G with bipartition (V_1, V_2), a *two-layer drawing* of G is a drawing such that vertices of V_1 and V_2 are placed on two parallel lines and edges are represented as straight lines between the vertices. A two-layer drawing can be specified by two linear orders τ_1 of V_1 and τ_2 of V_2. A crossing in a two-layer drawing is a pair of edges that intersect each other in a point that is not a vertex. The number of crossings is defined by the order of V_1 and V_2 on the lines. The One-Sided Crossing Minimization problem consists of ordering vertices of one part of the bipartite graph, for instance V_2, given a linear order of the other part, for instance V_1, that minimizes the number of crossings. This problem is a key sub-problem for drawing hierarchical graphs [Bastert and Matuszewski, 1999, Battista et al., 1999, Healy and Nikolov, 2013, Mutzel, 2009] or producing row-based VLSI layouts [Sechen, 2012, Stallmann et al., 2001].

Problem name: One-Sided Crossing Minimization (OSCM)
Given: A bipartite graph $G = (V_1, V_2, E)$, a linear order τ_1 on V_1 and $k \in \mathbb{N}$
Output: Is there a linear order τ_2 on V_2 such that, in the two-layer drawing specified by (τ_1, τ_2), at most k edge crossings occur?

The problem is known to be NP-complete [Eades and Wormald, 1994] even in sparse graphs [Muñoz et al., 2001] and FPT in the number of edge crossings k [Dujmovic et al., 2008, Dujmovic and Whitesides, 2004, Fernau et al., 2014], including sub-exponential algorithms. In [Fernau et al., 2014], a sub-exponential algorithm that runs in time $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k}\log k)})$ is presented which has been improved to $\mathcal{O}(2^{\sqrt{2k}} + n)$ in [Kobayashi and Tamaki, 2015]. The two-sided variant of the problem (where the permutation of both sides is variable) is also FPT in the number of crossings [Kobayashi et al., 2014]. A similar problem of deleting up to k edges in order to get a graph that can be drawn without crossings is NP-complete (in both variants: where the order on one or both layers is free to choose) and FPT in k (in both variants) [Dujmovic et al., 2006]. For a broader overview, we refer to some survey papers on graph crossing numbers [Schaefer, 2012, Vrt'o, 2008]. Note that OSCM is a cornerstone of algorithms dealing with the so-called *Sugiyama* *approach* to hierarchical graph drawing, see [Healy and Nikolov, 2013, Sugiyama et al., 1981].

5.3 Kemeny Rank Aggregation (KRA)

We are now shifting our focus to the field of social choice. There, preference lists are extensively used in social science surveys and voting systems to capture information about choice. The problem of combining several preference lists into a single one, called the aggregation, was initially discussed by Kemeny in [Kemeny, 1959]. This approach aims at minimizing the total disagreement (formalized below) between the several input rankings and their aggregation. The idea itself has not only applications in (the theory of) elections in the context of social sciences, say, on a committee, but has also been suggested as a means of designing meta-search engines [Dwork et al., 2001]. It has been also shown by Young and Levenglick [Young and Levenglick, 1978] that the aggregation method proposed by Kemeny is in fact the only one satisfying a number of natural requirements on such aggregations.

More precisely, in Kemeny Rank Aggregation we are given a set Π of rankings (also called *votes*) over a set of alternatives C (also called *candidates*), and a positive integer k, and are asked for a ranking π of C, such that the sum of the *Kendall-Tau distances* (or, KT-distances for short) of π to all the votes, called its *Kemeny score*, is at most k. A ranking π that gives the smallest Kemeny score is called a *Kemeny consensus*. The KT-distance between two rankings π_1 and π_2 is the number of pairs of candidates that are ordered differently in the two rankings and is denoted by KT-dist(π_1, π_2). Hence, if $\pi_1, \pi_2 : [|C|] \to C$, KT-dist$(\pi_1, \pi_2) = |\{(c, c') \in C \times C \mid c <_{\pi_1} c' \wedge c' <_{\pi_2} c\}|$. Observe that the Kendall-Tau distance can be seen as the 'bubble sort' distance, i.e., the number of pairwise adjacent transpositions needed to transform one ranking into the other. More formally, this leads to the following problem.

Problem name: Kemeny Rank Aggregation (KRA)
Given: A list of votes Π over a set of candidates C and a non-negative integer k
Output: Is there a ranking π on C such that the sum of the KT-distances of π to all the votes is at most k.

Hence, given rankings $\pi_1, \ldots, \pi_m$ of C and a non-negative integer k, the question is if there exists a ranking $\pi : [|C|] \to C$ such that $\sum_{i=1}^{m}$ KT-dist$(\pi, \pi_i) \leq k$.

Parameterizations of KRA The problem Kemeny Rank Aggregation is known to be NP-complete [Bartholdi et al., 1989], even if the input contains only four votes [Dwork et al., 2001].[1] For this reason, KRA has been studied from the perspective of parameterized complexity theory under a variety of parameterizations. Below, we consider two prominent parameterizations for this problem.

The first parameter we consider is the cost of a solution. Simjour [Simjour, 2009] obtained an algorithm for the problem that runs in time $\mathcal{O}^\star(1.403^k)$, where k upper-bounds the sum of the KT-distances of the solution π from all the votes. There are also subexponential algorithms for Kemeny Rank Aggregation under this parameterization: Karpinski and Schudy [Karpinski and Schudy, 2010] obtained an algorithm for Kemeny Rank Aggregation that runs in $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ time, while the algorithm of Fernau *et al.* [Fernau et al., 2014], based on a different methodology, runs in $\mathcal{O}^\star(k^{\mathcal{O}(\sqrt{k})})$ time. Both algorithms hide some constant factor in the $\mathcal{O}$-notation in the exponent that is not that clear from the expositions, but clearly worse than what we obtain in Chapter 6. Our considerations are also valid for the *weighted Kemeny score*, a modification suggested in [Betzler et al., 2009] that assigns positive weights to the voters. We can add some comments on conditional lower bounds of KRA by bringing together facts from different parts of the literature.

Theorem 25. KRA *on instances with only $m = 4$ votes on some candidate set C and some integer k bounding the sum of the Kendall-Tau distances to a solution can be solved neither in time $\mathcal{O}^\star\left(2^{o(|C|)}\right)$ nor in time $\mathcal{O}^\star\left(2^{o(\sqrt{k})}\right)$ unless ETH fails.*

Proof. The reduction of Dwork *et al.*, see footnote 1, starts out with an instance $G = (V, E)$ of Feedback Arc Set and produces an instance (C, Π) of Kemeny Rank Aggregation such that $|C| = |V| + |E|$ and $|\Pi| = 4$. Combined with the analysis by Kobayashi and Tamaki [Kobayashi and Tamaki, 2015] of the standard reduction chain from 3-SAT to Feedback Arc Set, we can note that the number of candidates of the KRA instance obtained from 3-SAT grows linearly in the number of variables and clauses. Hence, an algorithm solving KRA in time $\mathcal{O}^\star\left(2^{o(|C|)}\right)$ would break ETH. Moreover, as $k \leq |\Pi| \cdot |C|^2 = 4 \cdot |C|^2$ by observing the connection to bubble-sort, an algorithm running in time $\mathcal{O}^\star\left(2^{o(\sqrt{k})}\right)$ would also break ETH. □

This proves, in particular, the conditional optimality of the mentioned algorithm of Karpinski and Schudy. Recall that Simjour [Simjour, 2009] obtained an algorithm for Kemeny Rank Aggregation that runs in time $\mathcal{O}^\star(1.403^k)$. For moderate values of k, this might be better than the subexponential-time algorithm of Karpinski and Schudy.

[1] The proof of this fact is not contained in the conference paper [Dwork et al., 2001] but only appears in Appendix B of `http://www.wisdom.weizmann.ac.il/~naor/PAPERS/rank_www10.html`.

The second parameter we consider is the *unanimity width* of the set of votes, which is based on the notion of unanimity order of a set of votes [Charon and Hudry, 2007]. This parameter is defined in Section 6.5

5.4 Grouping by Swapping (GbS)

From the field of string problems, we discuss the problem of Grouping by Swapping (GbS for short). This problem asks whether a given string can be transformed by at most k interchanges of neighboring letters into a block format where all occurrences of a letter are adjacent to form one single block each. This problem has been mentioned in the famous list of NP-complete problems by Garey and Johnson in [Garey and Johnson, 1979]. Further algorithmic aspects are discussed in [Downey and Fellows, 2013, Wong and Reingold, 1991]. We show that GbS can be reduced to OSCM in a parameter-preserving way and hence inherits FPT-results shown above. We first discuss the problem GbS itself and then continue with the reductions.

Problem name: Grouping by Swapping (GbS)
Given: A finite alphabet Σ, a string $w \in \Sigma^*$, and a non-negative integer $k \in \mathbb{N}$.
Output: Is there a sequence of at most k adjacent swaps such that w is transformed into a string w' where all occurrences of each symbol are in a single block each?

Let us formalize this problem a bit more. If $w, w' \in \Sigma^*$ both have length n, we call w' a *permutation* of w if there exists a bijection $\pi : [n] \to [n]$ such that, for any $i \in [n]$, $w'[i] = w[\pi(i)]$. Slightly abusing notation, we will also write $w' = \pi(w)$. Special bijections are adjacent swaps $\sigma_i : [n] \to [n]$ (with $i \in [n-1]$) that act as the identity with two exceptions: $\sigma_i(i) = i + 1$ and $\sigma_i(i + 1) = i$. Swaps are a special case of transpositions and hence they are involutions. Interpreted as acting on words, swaps exchange neighboring letters. Every bijection $\pi : [n] \to [n]$ can be written as a composition of swaps (property $(*)$). Hence, given a permutation w' of w, we can ask to compute the *swap distance*, written $sd(w, w')$, which is the smallest number k of swaps σ_{i_1}, σ_{i_2}, $\ldots$, σ_{i_k} such that $w' = (\sigma_{i_1} \circ \sigma_{i_2} \circ \cdots \circ \sigma_{i_k})(w)$. Observe that sd can be viewed, for each mapping $g : \Sigma \to \mathbb{N}$, as a metric on the space of all words $w \in \Sigma^*$ with $g(a)$ occurrences of a for each letter $a \in \Sigma$. In particular, $sd(w, w') = sd(w', w)$ for all permutations w' of w. The swap distance between two words can be expressed as a special case of the *Kendall-Tau* distance [Kendall, 1938] (also called the *bubble-sort* distance or the *number of inversions*) between two linear orders. Let $w \in \Sigma^*$ be a word of length n and

w' be a permutation of w. Now, we will show how to transform the swap distance between w and w' into a Kendall-Tau distance between two linear orders. First, consider the function $g_w : \Sigma \to \mathbb{N}$ that counts the number of occurrences of each letter $a \in \Sigma$ in w. Clearly, $g_w = g_{w'}$. Consider the new alphabet $\Sigma_w = \{(a, i) \mid a \in \Sigma, i \in [g_w(a)]\}$. Interpret w and w' as words over Σ_w by replacing the first occurrence of a in w by $(a, 1)$, the second occurrence by $(a, 2)$, etc. This way, w, w' have the property that each letter of Σ_w occurs exactly once in w, w'. Hence, w and w' defines two linear order $<_w$ and $<_{w'}$ on Σ_w. Note that, in the original instance of GbS, to transform w into w' with a minimum number of swaps, a swap must not exchange two identical letters. This means that the relative order of the occurrences of a letter in w will never change when applying a sequence of swaps to get w'. Therefore, the Kendall-Tau distance between $<_w$ and $<_{w'}$ is equal to the swap distance between w and w'. Now, we can adapt algorithms that compute the Kendall-Tau distance between two linear orders, to compute the swap distance of two words. Lowrance and Wagner give in [Lowrance and Wagner, 1975] a dynamic programming algorithm that runs in quadratic time. The idea is to run the bubble-sort algorithm to sort w according to $<_{w'}$ and keep track of the number s of swaps performed by bubble-sort. Then, we have $s = sd(w, w')$. A classical textbook problem is to improve this quadratic algorithm to run in time $\mathcal{O}(n \cdot \log n)$ by adapting the algorithm to use merge-sort instead of bubble-sort. A more complex algorithm by Chan and Patrascu [Chan and Patrascu, 2010] compute the Kendall-Tau distance in time $\mathcal{O}(n \cdot \sqrt{\log n})$ (property (+)).

The picture changes if we add one more degrees of freedom by not defining the target permutation. Let us call $w' \in \Sigma^*$ to be in *block format* if there is a bijection $f : [|\Sigma|] \to \Sigma$ such that $w' \in f(1)^* f(2)^* \cdots f(|\Sigma|)^*$. Alternatively, we can view f as defining a linear order $<_f$ on Σ, and then the block format of w corresponding to f is the $<_f$-lexicographic smallest permutation of w. Conversely, any linear order τ on Σ defines a bijection $f : [|\Sigma|] \to \Sigma$. GbS now asks, given $w \in \Sigma^*$ and $k \geq 0$, if there is some permutation w' of w that is in block format and has swap distance at most k from w. As claimed in [Garey and Johnson, 1979], this variant is NP-complete. Unfortunately, the proof referenced by [Garey and Johnson, 1979] is hidden in private communication. We remedy this below by proving that GbS is NP-complete even for strings w where each letter occurs exactly four times. Let us start with two rather straightforward observations.

Lemma 26. *Any string w can be grouped into blocks using at most $|w|^2$ many swaps.*

Proof. We number the letters in w, such that for each $a \in \Sigma$, the subsequence of numbers induced by a in w forms a contiguous sequence of numbers. Applying the bubble-sort algorithm to sort this sequence of number obtained gives a sequence of swaps. Applying this sequence of swap to w gives a block string. Therefore the bubble-sort algorithm gives a solution to the GBS problem, and this solution does at most $|w|^2$ swaps. □

In fact, any permutation of w can be obtained by using at most $|w|^2$ many swaps, as can be seen by bubble-sort. This reasoning also shows $(*)$. We can use this observation to obtain our first (easy) FPT-result, to be improved on later.

Lemma 27. GBS *on strings $w \in \Sigma^n$ parameterized by $|\Sigma|$ can be solved in time $\mathcal{O}^\star(|\Sigma|!)$.*

Proof. Let (Σ, w, k) be an instance of GBS with $w \in \Sigma^n$. Let $g_w \colon \Sigma \to [n]$ be the function that maps each letter $a \in \Sigma$ to the number of occurrences of a in w. As the swap distance of two strings can be computed in less than quadratic time, we can test for all $|\Sigma|!$ bijections $f \colon [|\Sigma|] \to \Sigma$ whether $sd(w, f(1)^{g_w(f(1))} f(2)^{g_w(f(2))} \ldots f(|\Sigma|)^{g_w(f(|\Sigma|))}) \leq k$. □

We are now going to show that computing the swap distance can be done by considering the distance for pairs of letters, summing up the corresponding results. Note that the expression of the swap distance in Equation 5.1 in Lemma 28 have the same form as the expression of the cost of a solution for the PCO problem or as in Equation 6.1. To make this more precise, let $\Sigma' \subseteq \Sigma$ and consider the projection $p_{\Sigma,\Sigma'} : \Sigma \to \Sigma'$ that maps $a \mapsto a$ for $a \in \Sigma'$ and $a \mapsto \varepsilon$, the empty word, if $a \notin \Sigma'$, as a morphism $\Sigma^* \to (\Sigma')^*$.

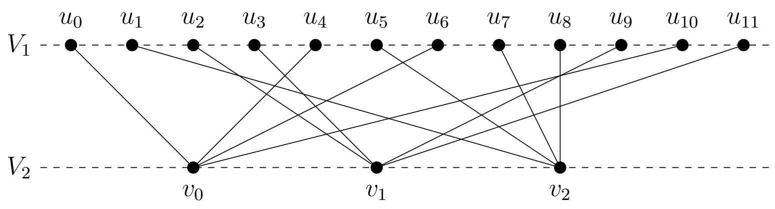
Lemma 28. *Let $w, w' \in \Sigma^*$ such that w' is a permutation of w. Let w' be in block format following the linear order τ on Σ. Then,*

$$sd(w, w') = \sum_{a,b \in \Sigma, a <_\tau b} sd(p_{\Sigma,\{a,b\}}(w), p_{\Sigma,\{a,b\}}(w')) . \tag{5.1}$$

Moreover, $p_{\Sigma,\{a,b\}}(w') = a^{|w|_a} b^{|w|_b}$ if $a <_\tau b$.

Proof. If $w, w' \in \Sigma^*$ such that w' is a permutation of w, i.e., $w' = \pi(w)$, then π can be written as the composition of $sd(w, w')$ many transpositions σ_i, leading to a sequence $w_0 = w$, $w_1 = \sigma_1(w_0)$, …, $w_i = \sigma_i(w_{i-1})$, …, $w' = w_{sd(w,w')}$. We can associate to σ_i a unique pair of letters $\ell_i = \{a, b\}$ that is actually swapped by this transposition. We can characterize ℓ_i by the property $p_{\Sigma,\ell_i}(w_{i-1}) \neq p_{\Sigma,\ell_i}(w_i)$. Hence, $sd(w, w') = \sum_{a,b \in \Sigma, a <_\tau b} |\{i \mid \ell_i = \{a, b\}\}|$. Moreover, if $a <_\tau b$, none of the transpositions will ever swap a factor ab, as otherwise the sequence of transpositions would not be minimal. Hence, $|\{i \mid \ell_i = \{a, b\}\}| = sd(p_{\Sigma,\{a,b\}}(w), p_{\Sigma,\{a,b\}}(w'))$. □

Discussing NP-completeness. Next, we prove NP-completeness of GBS, even for quite restricted instances, by making use of a somewhat similar result for OSCM, based on [Muñoz et al., 2001].



u_0 u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8 u_9 u_{10} u_{11}
V_1
V_2
v_0 v_1 v_2

Figure 5.1: Reduction from OSCM to GbS when all vertices in V_1 of the OSCM instance have degree one. In this example, the reduction gives the following instance of GbS: $\Sigma = \{v_0, v_1, v_2\}$ and $w = v_0v_2v_1v_1v_0v_2v_0v_2v_2v_1v_0v_1$

Theorem 29. GbS *is* NP*-complete, even if each letter has exactly 4 occurrences.*

Proof. We can guess and check a solution in polynomial time, hence membership in NP is clear. In order to show NP-hardness, we give a reduction from OSCM which is also NP-complete if each node in V_2 has degree four and each vertex in V_1 has degree one, i.e., if the graph is a forest of 4-stars [Muñoz et al., 2001]. Let $G = (V_1, V_2, E)$ be an instance of OSCM with order τ_1 on V_1, and integer k, such that all vertices in V_1 are of degree one and all vertices in V_2 are of degree four. We set $\Sigma = V_2 = \{v_1, v_2, \ldots, v_n\}$. Clearly, $|V_1| = 4n$. We construct $w \in \Sigma^{4n}$ (starting from the empty word) by going through the vertices in V_1, following the order τ_1. If the current vertex is adjacent to v_i, we concatenate v_i to w (See Figure 5.1). As the vertices in V_1 are of degree one, this assignment is unambiguous. Following [Eades and Wormald, 1994], for vertices $v_i, v_j \in V_2$, let $c_{v_iv_j}$ be the number of crossings between edges incident to v_i and edges incident to v_j when v_i is placed at the left of v_j. Lemma 3 in [Eades and Wormald, 1994] states (also confer Equation 6.1), referring to [Eades and Kelly, 1986], that for a linear order τ_2 on V_2, the number of crossings $\mathit{cross}(G, \tau_1, \tau_2)$ of the edges between V_1 in order τ_1 and V_2 in order τ_2 is $\mathit{cross}(G, \tau_1, \tau_2) = \sum_{v_i,v_j \in V_2, v_i <_{\tau_2} v_j} c_{v_iv_j}$. Clearly, for $v_i, v_j \in V_2$ the number of crossings $c_{v_iv_j}$ is equal to $\mathit{sd}(p_{\Sigma,\{v_i,v_j\}}(w), p_{\Sigma,\{v_i,v_j\}}(w_{\tau_2}))$, where w_{τ_2} is the τ_2-lexicographic smallest permutation of w. Combining this observation with Lemma 28, we obtain that for every linear order τ_2, $\mathit{sd}(w, w_{\tau_2}) = \mathit{cross}(G, \tau_1, \tau_2)$. □

In Section 6.6, we will show that, in a sense, the reduction presented in our NP-hardness result for GbS can be reversed. This also shows the following:

Remark 30. GbS is polynomial-time solvable when each letter occurs at most twice.

To summarize, GbS is NP-complete if each letter appears at most four times and solvable in polynomial time if each letter appears at most two times leaving open the natural question Can GbS instances be solved in polynomial time if each letter occurs at most thrice? Note that it is also open whether subcubic OSCM graph instances can be solved in polynomial time. Furthermore, within KRA, it is open if instances with three voters can be solved in polynomial time.

5.5 Reconfiguration Problems

In the field of reconfiguration, one is interested in studying relationships among solutions of a problem instance [Ito et al., 2011, Nishimura, 2018, Wrochna, 2018]. Here, by reconfiguration of one solution into another, we mean a sequence of steps where each step transforms a feasible solution into another. Three fundamental questions in this context are: (1) Is it the case that any two solutions can be reconfigured into each other? (2) Can any two solutions be reconfigured into each other in a polynomial number of steps? (3) Given two feasible solutions X and Y, can one find in polynomial time a reconfiguration sequence from X to Y?

In Chapter 8, we study two reconfiguration problems. In those two problems, we are given a graph G and two linear orders on the vertices of G of bounded width (cutwidth and vertex separation number) and we want to know if we can reconfigure one into the other respecting some width constraints. In this context, we use the following definition of linear orders. A linear order on a set V is a bijection $\tau : [|V|] \to V$. For each $i \in [|V|]$, $\tau(i)$ is the i-th elements in τ. This means that if $i < j$ then $\tau(i) <_\tau \tau(j)$. To formally define those two problems, we first need to define the reconfiguration rule.

Swap Operation. Given a set V of size n and a number $i \in [n-1]$, a *swap* at position i is an operation on linear orders that consist of exchanging the element at position i with the element at position $i+1$. More, precisely, given two linear orders $\tau, \tau' : [n] \to V$, τ' is obtained by applying a swap at position i to τ if and only if for every $j \in [n] \setminus \{i, i+1\}$, $\tau'(i) = \tau(i+1)$, and $\tau'(i+1) = \tau(i)$. Given linear orders $\tau, \tau' : [n] \to V$ of V and a number $i \in [n-1]$, we write $\tau \xrightarrow{i} \tau'$ to indicate that τ' is obtained from τ by *swapping* the order of the elements at positions i and $i+1$.

Linear Order Reconfiguration. We say that τ can be reconfigured into τ' in one swap, and denote this fact by $\tau \to \tau'$, if there exists some $i \in [n]$ such that $\tau \xrightarrow{i} \tau'$. We say that τ can be reconfigured into τ' in at most r swaps, and denote this fact by $\tau \to_r \tau'$, if there are numbers $r' \in [r]$, $i_1, \dots i_{r'} \in [n]$, and linear orders $\tau_0, \dots, \tau_{r'}$ such that

$$\tau = \tau_0 \xrightarrow{i_1} \tau_1 \xrightarrow{i_2} \cdots \xrightarrow{i_{r'}} \tau_{r'} = \tau'.$$

We call this sequence a *reconfiguration sequence* from τ to τ'. The mere existence of a (possibly empty) reconfiguration sequence from τ to τ' is also written as $\tau \to^* \tau'$.

Given two linear orders τ and τ' of the vertices of an n-vertex graph, τ can always be reconfigured into τ'. To see that, notice that the bubble-sort algorithm performs only swaps. Therefore running the bubble-sort algorithm to sort τ according to τ' gives a valid reconfiguration sequence from τ to τ'. This connection allows us to be more precise.

Lemma 31. *A linear order τ can always be reconfigured into another ordering τ' using a reconfiguration sequence of length at most n^2.*

An interesting question is to ask if τ can be reconfigured into τ' in such a way that every linear order appearing in the reconfiguration sequence has bounded width. We will look at this problem for cutwidth and vertex separation number.

5.5.1 Bounded Cutwidth Order Reconfiguration

For each $w \in \mathbb{N}$, and each n-vertex graph G, we let $\mathrm{CW}(G, w) = \{\tau : [n] \to V(G) : \mathrm{cw}(G, \tau) \leq w\}$ be the set of linear orders of $V(G)$ of cutwidth at most w. We say that τ can be *reconfigured* into τ' in cutwidth at most w if there is a *reconfiguration sequence* $\tau = \tau_0 \xrightarrow{i_1} \tau_1 \xrightarrow{i_2} \cdots \xrightarrow{i_r} \tau_r = \tau'$ such that for each $j \in \{0, \ldots, r\}$, $\tau_j \in \mathrm{CW}(G, w)$.

Problem name: Bounded Cutwidth Order Reconfiguration (BCOR)
Given: An n-vertex graph G, two linear orders of the vertex set of G $\tau, \tau' : [n] \to V(G)$, and a non-negative integer $w \in \mathbb{N}$.
Output: Is it true that τ can be reconfigured into τ' in cutwidth at most w?

5.5.2 Bounded Vertex Separation Number Order Reconfiguration

For each $w \in \mathbb{N}$ and each n-vertex graph G, let $\mathrm{VSN}(G, w) = \{\tau : [n] \to V(G) : \mathrm{vsn}(G, \tau) \leq w\}$ be the set of linear orders of $V(G)$ of vertex separation number at most w. We say that τ can be *reconfigured* into τ' in vertex separation number at most w if there is a *reconfiguration sequence* $\tau = \tau_0 \xrightarrow{i_1} \tau_1 \xrightarrow{i_2} \cdots \xrightarrow{i_r} \tau_r = \tau'$ such that for each $j \in [r]$, $\tau_j \in \mathrm{VSN}(G, w)$.

Problem name: Bounded Vertex Separation Number Order Reconfiguration
Given: An n-vertex graph G, two linear order $\tau, \tau' : [n] \to V(G)$ on the vertex set of G, and a non-negative integer $w \in \mathbb{N}$.
Output: Is it true that τ can be reconfigured into τ' in vertex separation number at most w?

5.6 Graph Isomorphism (GI)

Given two graphs G_1 and G_2, a *morphism* ϕ from G_1 to G_2 is mapping from $V(G_1)$ to $V(G_2)$ that preserve adjacency. In other words, for all edges $\{u, v\} \in E(G_1)$, we have $\{\phi(u), \phi(v)\} \in E(G_2)$. An *isomorphism* ϕ between G_1 and G_2 is a bijection between $V(G_1)$ and $V(G_2)$ such that ϕ and its inverse are morphism. We say that G_1 and G_2 are *isomorphic* if there exists an isomorphism between them. Intuitively, G_1 is isomorphic to G_2, if we can rename the vertices of G_1 to obtain G_2. The Graph Isomorphism problem asks if two given graphs are isomorphic or not. It is a famous problem in theoretical computer science because its exact complexity is a major open problem. More precisely, GI is not known either to be in P or to be NP-complete. It has applications in many fields such as computer vision [Conte et al., 2003, Gori et al., 2004, Abdulrahim and Misra, 1998, Messmer and Bunke, 1999], information retrieval [Berztiss, 1973], data mining [Washio and Motoda, 2003], VLSI layout validation [Ohlrich et al., 1993, Abadir and Ferguson, 1990, Pelz and Roettcher, 1991] or chemistry [Faulon, 1998]. For a recent overview on the Graph Isomorphism problem, we refer the reader to [Grohe and Neuen, 2021, Grohe and Schweitzer, 2020].

Problem name: Graph Isomorphism (GI)
Given: Two graphs G and G'.
Output: Is there an isomorphism from G to G'?

The Graph Isomorphism problem has been shown to be solvable in time $f(k) \cdot n^{\mathcal{O}(1)}$ (that is, FPT time) whenever the parameter k stands for eigenvalue multiplicity [Babai et al., 1982], treewidth [Lokshtanov et al., 2014], feedback vertex-set number [Kratsch and Schweitzer, 2010], or size of the largest color class [Furst et al., 1980] of the involved graphs. On the other hand, GI can be solved in time $f_1(k) \cdot n^{f_2(k)}$ (that is, in XP time), whenever the parameter k stands for genus [Miller, 1980], rankwidth [Grohe and

Schweitzer, 2015], maximum degree [Luks, 1982], size of an excluded topological subgraph [Grohe and Marx, 2015], or size of an excluded minor [Grohe, 2012]. We note that, in particular, Babai's algorithm and techniques have been recently used to improve the fastest FPT algorithm for graphs of treewidth at most k from $2^{\mathcal{O}(k^5 \cdot \log k)} \cdot n^{\mathcal{O}(1)}$ [Lokshtanov et al., 2014] to $2^{\mathcal{O}(k \cdot \mathsf{polylog}(k))} \cdot n^{\mathcal{O}(1)}$ [Grohe et al., 2018b], and for graphs of maximum degree d, the fastest XP-algorithm has been improved from $n^{\mathcal{O}(d/\log d)}$ [Babai et al., 1983] to $n^{\mathsf{polylog}(d)}$ [Grohe et al., 2018a]. Showing that isomorphism for graphs of cutwidth w can be solved in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ is still an open problem.

5.7 String Rewriting System

A *string rewriting system* is a pair (Σ, R) where Σ is a finite, non-empty set of symbols (an alphabet), and $R \subseteq \Sigma^* \times \Sigma^*$ is a binary relation over Σ^*. Elements of R are called *rewriting rules*. Given a rewriting rule $(u, v) \in R$, we say that u can be *rewritten* to v, and write $u \to v$. Given two strings $x, y \in \Sigma^*$, we say that x can be *transformed* into y using a rewriting rule $u \to v \in R$ at position $i \in \mathbb{N}_{>0}$ if there exists $p \in \Sigma^{i-1}$ and $s \in \Sigma^*$ such that $x = pus$ and $y = pvs$. We write $x \xrightarrow{i} y$ to denote that x can be transformed into y by the application of some rewriting rule at position i. We write $x \to y$ to denote that x can be transformed into y by the application of some rewriting rule at some position $i \in \mathbb{N}_{>0}$. We say that y is *reachable* from x if there is a sequence of strings $x = x_0, x_1, \ldots, x_m = y$ such that $x_{i-1} \to x_i$ for each $i \in [m]$. We write $x \to^* y$ to denote that y is reachable from x. We say that x and y are *R-equivalent* if $x \to^* y$ and $y \to^* x$.

Two-Letter String Rewriting System. A *two-letter string rewriting system* is a string rewriting system (Σ, R) where $R \subseteq \Sigma^2 \times \Sigma^2$ is a set of rewriting rules of the form $ab \to cd$. Let x and y be strings in Σ^n and $i \in [n-1]$. In this special case of two-letter string rewriting systems, x can be transformed into y by applying a rewriting rule $ab \to cd$ at position i if $x_i x_{i+1} = ab$, $y_i y_{i+1} = cd$ and $x_j = y_j$ for $j \notin \{i, i+1\}$.

Problem name: Reachability
Given: A finite alphabet Σ, a string rewriting system R, two string $x, y \in \Sigma^*$
Output: Is there a sequence of rewriting rules that transforms x into y?

Reachability is a central problem in the field of string rewriting [Book and Otto, 1993] and can also be studied under the light of term rewriting theory [Klop, 1990, Barendsen, 2003, Baader and Nipkow, 1999, Book and Otto, 1993]. The complexity of the reachability problem is highly dependent on the rewriting system R. For general rewriting systems, the problem becomes undecidable [Book and Otto, 1993]. In the case of two-letter rewriting, reachability can be solved in PSPACE since in this case, strings never grow in size. It is also not difficult to design two-letter rewriting systems for which the reachability problem is PSPACE-complete. Nevertheless, our results imply that for each $w \in \mathbb{N}$, the $R(2w)$-reachability problem for unit decompositions of length n and width at most w is reducible to the graph isomorphism problem. Therefore, it can be solved in time $n^{\mathsf{polylog}(n)}$, independently of k, using Babai's quasi-polynomial time algorithm for graph isomorphism [Babai, 2016]. An interesting question we leave unsolved is the complexity of $R(\alpha \cdot w)$-reachability for unit decompositions of width at most w when α is a rational number with $1 \leq \alpha < 2$. In particular, we do not know if there is such an α for which the reachability problem becomes PSPACE-hard.

Part II

Ordering-Related Problems Parameterized by Width

Chapter 6

Solving KRA, GbS, OSCM and PCO Using a DP Algorithm for CO

In this chapter, we use the Completion of an Ordering problem as a general framework to solve different problems such as Kemeny Rank Aggregation, Grouping by Swapping and One-Sided Crossing Minimization. The main result of this chapter is a dynamic programming algorithm to solve CO in single-exponential time parameterized by the pathwidth of the cocomparability graph of the input order (Theorem 37). To develop this algorithm, we introduce a new parameter based on path decomposition called consistent pathwidth. For this new parameter, we show that it coincides with pathwidth for cocomparability graphs (Lemma 35). We also give an algorithm to construct a consistent path decomposition of width $\mathcal{O}(w)$ in time $\mathcal{O}^\star(2^{\mathcal{O}(w)})$ (Lemma 36). The algorithm for CO then uses dynamic programming over a consistent path decomposition to solve CO.

Building on recent advances in the theory of C_k-free graphs [Chudnovsky et al., 2020], we establish an upper bound for the pathwidth of a cocomparability graph in terms of the number of edges of the graph. As a by-product of this result, we obtain the first algorithm running in time $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$ for PCO and PCIO (Theorem 41 and Theorem 50). Previously, the best algorithm for this problem parameterized by cost had asymptotic time complexity of $\mathcal{O}^\star(k^{\mathcal{O}(\sqrt{k})}) = \mathcal{O}^\star(2^{\sqrt{k}\log k})$. Therefore, we remove the log-factor in the exponent. Using the connection to KRA, we show that this is optimal under the Exponential Time Hypothesis (ETH) (Corollary 63).

Finally, we give the reductions from KRA to PCO and from OSCM and GbS to PCIO. Together with our result on PCO and PCIO, we match or improve the current best-known running times. For KRA, OSCM and GbS, we discuss the meaning of the pathwidth parameter in PCO when applied to instances given by the reductions. There, we show some connections to parameters specific to each problem and we are able to improve the running time with respect to those parameters (Corollary 65 and Corollary 69).

6.1 Consistent path decomposition

For the need of our main algorithm described in the next section (Section 6.2), we start by defining a suitable notion of consistency between a path decomposition and a given partial order. A path decomposition that respects the consistency property for a partial order ρ is called a *ρ-consistent path decomposition.* The same way the pathwidth of a graph is defined based on path decompositions, we define the notion of *ρ-consistent pathwidth*[1]. Even if the ρ-consistent pathwidth parameter is more restrictive than the classical pathwidth, we show that given a partial order ρ, the notions coincide for G_ρ, the cocomparability graph of ρ (Lemma 35). From an algorithmic point of view, we show that constructing a ρ-consistent path decomposition of the cocomparability graph G_ρ can be done in linear time for interval orders (Lemma 32), and, for general partial orders, it is fixed-parameter tractable parameterized by the pathwidth of G_ρ (Lemma 36).

Let $G = (V, E)$ be a graph, $\rho \subseteq V \times V$ be a (strict) partial order on the vertices of G and $D = (B_1, \dots, B_l)$ be a path decomposition of G. We say that D is *consistent* with ρ or, similarly, *ρ-consistent* if there is no pair of vertices $(x, y) \in \rho$ such that

$$\max(\{i \in [l] \mid y \in B_i\}) < \min(\{i \in [l] \mid x \in B_i\}).$$

In other words, if x is smaller than y in ρ, then y cannot be forgotten in D before x is introduced in D. The *width* of a ρ-consistent path decomposition D is the maximum size of a bag in D minus 1, $w(D) = \max_i\{|B_i| - 1\}$. The *$\rho$-consistent pathwidth* of G, denoted by $cpw(G, \rho)$, is the minimum width of a ρ-consistent path decomposition of G. We will be interested in particular in the consistent pathwidth $cpw(G_\rho, \rho)$ of the cocomparability graph G_ρ consistent with ρ.

Let us explain why our pathwidth measure can be seen as a *distance to triviality* parameterization in the context of CO. A trivial instance of CO is a linear order, as it has cost zero. Then, the cocomparability graph is an independent set and has consistent pathwidth 0.[2] In the opposite case, if the input partial order is empty, then the cocomparability graph is a clique and has consistent pathwidth $|V| - 1$. It is also worth noticing

[1]The notion of ρ-consistent pathwidth was first introduced in [Arrighi et al., 2020] but the problem of computing a ρ-consistent path decomposition was left open and solved in [Arrighi et al., 2021c].

[2]In Lemma 35, we show that consistent pathwidth is equal to pathwidth.

that it is NP-hard to determine the pathwidth of a cocomparability graph, together with an optimal path decomposition, as observed in by Habib and Möhring in [Habib and Möhring, 1994].

Our first lemma gives a way to build an ι-consistent path decomposition given an interval representation of an interval order ι. This construction is similar to the construction of a path decomposition given an interval representation of an interval graph.

Lemma 32. *Let ι be an interval order on V. One can construct a minimum width path decomposition of G_ι consistent with ι of width $w(\iota) - 1$ in time $\mathcal{O}(|V|)$.*

Proof. First, we build an interval representation $\{I_v \mid v \in V\}$ of ι which can be done in linear time. How to obtain interval representations for interval orders was first shown by Booth and Lueker in [Booth and Lueker, 1976]. Improved variants of finding interval representations are given in [Hsu and Ma, 1999, Corneil et al., 2009], also see [Corneil et al., 2013]. For space and further complexity considerations, confer [Köbler et al., 2015, Köbler et al., 2011].

From $\{I_v \mid v \in V\}$, we derive an ι-consistent path decomposition of minimum width as follow. For each element v in V, we let l_v and r_v be the left and right endpoints of I_v. For every point x on the real line $\mathbb{R}$ that corresponds to an endpoint of one or more intervals, we associate a bag $B_x = \{v \mid x \in I_v\}$. Then, we order the bags following the order of l_v and r_v on the real line (See Figure 6.1). For each element $v \in V$, $v \in B_{l_v}$. Given three bags B_x, B_y, B_z such that $x \le y \le z$, we have that $B_x \cap B_z = \{v \mid x \in I_v\} \cap \{v \mid z \in I_v\} = \{v \mid l_v \le x \le z < r_v\} \subseteq \{v \mid l_v \le y < r_v\} = B_y$. For each edge $(u, v) \in E(G_\iota)$, I_u and I_v intersect, therefore, we have either $l_v \in I_u$ or $l_u \in I_v$. If $l_v \in I_u$ then $u, v \in B_{l_v}$, similarly if $l_u \in I_v$ then $u, v \in B_{l_u}$. So this construction builds a path decomposition. We call this path decomposition D. As each bag corresponds to the set of vertices that intersect at a certain point on the real line, each bag is a clique, therefore, D is a minimum path decomposition. Now, we will show that D is consistent with ι. More precisely, we will show that for each pair $(u, v) \in \iota$, $\max\{x \in \mathbb{R} \mid u \in B_x\} < \min\{x \in \mathbb{R} \mid v \in B_x\}$. For each $(u, v) \in \iota$, we have $l_u < r_u \le l_v$, $\max\{x \in \mathbb{R} \mid u \in B_x\} < r_u \le l_v \le \min\{x \in \mathbb{R} \mid v \in B_x\}$. Therefore, D is consistent with ι. Note that each bag is a clique, therefore, this is a path decomposition of minimum width. A clique in G_ι is an antichain of ι and each antichain of ι forms a clique in G_ι. Therefore, we have that D has width $w(\iota) - 1$. □

We will refer to the decomposition built from an interval representation $\{I_v \mid v \in V\}$ as the path decomposition *derived* from the interval representation of ι.

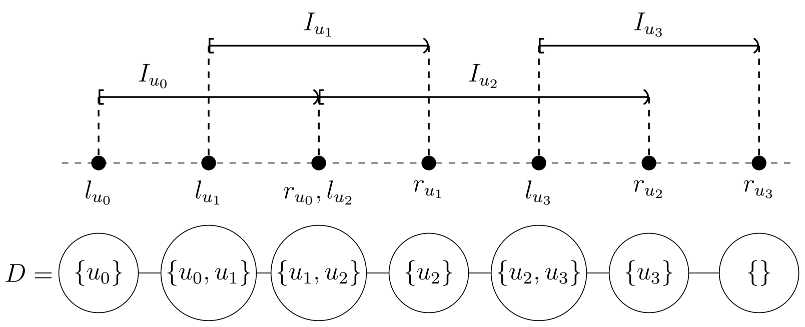


Figure 6.1: Consistent path decomposition of the cocomparability graph of an interval order given an interval representation. Given an interval order ι with its interval representation $I = \{I_{u_0}, I_{u_1}, I_{u_2}, I_{u_3}\}$, D is the path decomposition obtained in the proof of Lemma 32.

Nice path decompositions are convenient to describe and prove algorithms on path decompositions. To be able to use nice path decompositions in our setting we need to prove that we can build a nice path decomposition from a path decomposition while keeping the property of being consistent with some partial order.

Lemma 33. *Let $G = (V, E)$ be a graph. Given a partial order ρ on V and a path decomposition D of G of width w and length l that is consistent with ρ, one can construct in time $\mathcal{O}(w \cdot l)$ a nice path decomposition of width w that is consistent with ρ.*

Intuitively, the construction works as follow. Given a path decomposition D, one can get a nice path decomposition by introducing before each bag B several new bags that will forget one by one each vertex forgotten by B and introduce each new vertex in B one by one. If D is consistent with ρ, then the new path decomposition is also consistent with ρ.

Proof of Lemma 33. Let l be the length of D. First, for each $i \in [l-1]$, if $B_i = B_{i+1}$, B_{i+1} is removed from D. After removing all the duplicated bags D is still a valid path decomposition and consistent with ρ. For the rest of the proof, we assume that for each $i \in [l-1]$ $B_i \neq B_{i+1}$. For each bag B_i, let $F_i = B_i \setminus B_{i+1}$ be the set of vertices that will be forgotten ($F_l = B_l$) and $I_i = B_i \setminus B_{i-1}$ be the set of vertices that are introduced ($I_1 = B_1$). For each bag B_i, $|I_i| - 1$ bags are added just before B_i in the path decomposition to introduce the vertices in I_i one by one, and $|F_i|$ bags are added after B_i to forget the vertices in F_i one by one. This can be done in time $\mathcal{O}(w \cdot l)$. Let D' denote this new path decomposition, with bags $(B'_1, B'_2, \ldots, B'_{2|V|})$, i.e., D' has length $2|V|$, as each vertex is introduced and forgotten exactly once. By construction, D' is a nice path decomposition. As we only add new bags in the path decomposition, there is a mapping $f : [l] \to [2|V|]$

that embeds D into D', i.e., $B'_{f(i)} = B_i$ for all $i \in [l]$. Moreover, f is strictly monotone, i.e., for $i < j$ we have $f(i) < f(j)$, and conversely, if $f(i) < f(j)$, then $i < j$. For $k \in [2|V|]$, let $f_1(k) \doteq \max\{i \in [l] \mid f(i) \leq k\}$ and $f_2(k) \doteq \min\{i \in [l] \mid f(i) \geq k\}$. Observe that $f_1(k) \leq f_2(k)$, with equality only if $k = f(i)$ for some $i \in [l]$. In general, we find $(*)$ that $f(f_1(k)) \leq k \leq f(f_2(k))$ for all $k \in [2|V|]$.

Now, assume that D' is not consistent with ρ. Then, there is some $(x, y) \in \rho$ with

$$\max(D', y) \doteq \max\{i \in [2|V|] \mid y \in B'_i\} < \min(D', x) \doteq \min\{i \in [2|V|] \mid x \in B'_i\}\,.$$

Hence, $B'_{\max(D',y)}$ is the last bag in the path decomposition D' before y is forgotten. Therefore, $B_{f_1(\max(D',y))}$ is the last bag in the path decomposition D before y is forgotten. In line with our notations, let $\max(D, y) \doteq \max\{i \in [l] \mid y \in B_i\}$. We just showed that $\max(D, y) = f_1(\max(D', y))$. Similarly, $B'_{\min(D',x)}$ is the bag in the path decomposition D' where x is introduced. Therefore, $B_{f_2(\min(D',x))}$ is the bag in the path decomposition D in which x is introduced. In line with our notations, let $\min(D, x) \doteq \min\{i \in [l] \mid x \in B_i\}$. We just showed that $\min(D, x) = f_2(\min(D', x))$. By $\max(D', y) < \min(D', x)$ and by $(*)$, we infer that $\max(D, y) = f_1(\max(D', y)) < f_2(\min(D', x)) = \min(D, x)$, which means that ρ is not consistent with D, contradicting our assumptions. □

For the cocomparability graphs, we can further show the following three lemmas. First, given a partial order ρ, the notion of consistency behave well with respect to an extension of ρ in the case of the cocomparability graph G_ρ.

Lemma 34. *Let ρ be a partial order on a set V, G_ρ be the cocomparability graph of ρ and D be a path decomposition of G_ρ consistent with ρ, then D is consistent with any extension of ρ.*

Proof. Let τ be an extension of ρ. For every pair $(x, y) \in \tau$ with $x \neq y$, if $(x, y) \in \rho$ then $\max\{i \in [l] \mid y \in B_i\} \geq \min\{i \in [l] \mid x \in B_i\}$; otherwise, $(x, y) \notin \rho$, as $x \neq y$ and $(x, y) \in \tau$, $(y, x) \notin \rho$ because τ extends ρ; hence, x and y are not comparable with respect to ρ; therefore, there is an edge xy in the cocomparability graph G_ρ; so there exists a bag B_i such that $x, y \in B_i$ and therefore $\max\{i \in [l] \mid y \in B_i\} \geq \min\{i \in [l] \mid x \in B_i\}$. □

Next, we show that the notions of consistent pathwidth and (ordinary) pathwidth coincde when restricted to the class of cocomparability graph associated with one of its partial orders. This justifies that we can use consistent path decomposition to build an algorithm on partial order parameterized by the pathwidth of the cocomparability graph.

Lemma 35. *Let $G_\rho = (V, E)$ be the cocomparability graph of a partial order $\rho \subseteq V \times V$. Then, $pw(G_\rho) = cpw(G_\rho, \rho)$.*

Proof. By definition we have $pw(G_\rho) \leq cpw(G_\rho, \rho)$. We will show that $cpw(G_\rho, \rho) \leq \mathrm{iw}(\rho) - 1$ and use the fact that $pw(G_\rho) = \mathrm{iw}(\rho) - 1$ (Theorem 2.1 from [Habib and Möhring, 1994]). By definition of $\mathrm{iw}(\rho)$, we can find an interval order ι such that $\mathrm{iw}(\rho) = w(\iota)$ and $\iota \subseteq \rho$. Let $\{I_v \mid v \in V\}$ be an interval representation of ι. Then, G_ι is the intersection graph of $\{I_v \mid v \in V\}$. Then, by Lemma 32, the path decomposition D of G_ι derived from $\{I_v \mid v \in V\}$ is consistent with ι and has width $w(\iota) - 1$. From Lemma 34, we know that D is also consistent with the extension ρ of ι. We conclude $cpw(G_\rho, \rho) \leq cpw(G_\iota, \iota) = w(\iota) - 1 = \mathrm{iw}(\rho) - 1 = pw(G_\rho)$. □

To have a complete FPT algorithm parameterized by pathwidth, we show that for any partial order ρ, one can construct a ρ-consistent path decomposition of the cocomparability graph G_ρ in fixed-parameter tractable time parameterized by the pathwidth of the graph G_ρ.

Lemma 36. *Let $\rho \subseteq V \times V$ be a partial order and G_ρ be the cocomparability graph of ρ. Then one can construct a nice ρ-consistent path decomposition D of G_ρ of width $\mathcal{O}(pw(G_\rho))$ in time $2^{\mathcal{O}(pw(G_\rho))} \cdot |V|$.*

Proof. Let ρ be a partial order on a set V and G_ρ be the cocomparability graph of ρ. It has been shown in Theorem 2.1 of [Habib and Möhring, 1994] that any minimal triangulation H of G_ρ is not only a cocomparability graph, but also an interval graph. This result allows us to compute a ρ-consistent path decomposition of G_ρ as follows.

We start by computing a tree decomposition $\mathcal{T}$ of G_ρ of width at most $5 \cdot pw(G) + 4$ in time $2^{\mathcal{O}(pw(G))} \cdot |V|$ using the algorithm from [Bodlaender et al., 2016]. Subsequently, we construct a triangulation $H_\mathcal{T}$ of G_ρ by transforming each bag of the decomposition $\mathcal{T}$ into a clique. More precisely, we add an edge to vertices u and v in G_ρ if and only if u and v occur together in some bag. This operation clearly preserves treewidth, since the size of the bags does not increase. Therefore, the graph $H_\mathcal{T}$ is a triangulation of G_ρ of treewidth at most $5 \cdot pw(G) + 4$. Now, we successively delete edges from $H_\mathcal{T}$ until we get a minimal triangulation H of G_ρ. In other words, by removing any additional edge from H, we either get a graph that is not triangulated or that is not a supergraph of G_ρ. We have $E(G_\rho) \subseteq E(H) \subseteq E(H_\mathcal{T})$ and $E(H)$ is minimal with respect to inclusion. Therefore, $tw(H) \leq tw(H_\mathcal{T}) = w$. By [Habib and Möhring, 1994], we know that H is an interval graph.

Now, adding an edge $\{u, v\}$ to G_ρ is equivalent to removing the edge constructing the DAG $\rho \backslash \{(u, v)\}$. What is shown in [Habib and Möhring, 1994] is that the DAG $\rho \backslash \{(u, v) \mid \{u, v\} \in E(H) \backslash E(G_\rho)\}$ is actually a partial order ι. Note that when deleting edges from a partial order, the only axiom that can be broken is transitivity. So what this result is really saying is that by deleting the pairs corresponding to edges that are in H but not in G, we can indeed preserve transitivity. The crucial fact about this construction is that the partial order ι is actually an interval order, and therefore H is an interval supergraph of G_ρ.

Now, from the interval graph H, we derive a ρ-consistent path decomposition D of G_ρ. This construction is as follows. Given two maximal cliques X and Y of H, we say that X is smaller than Y if there exist vertices $x \in X$ and $y \in Y$ such that $(x, y) \in \iota$. This relation defines a linear order on the maximal cliques [Habib et al., 2000]. It follows from [Habib et al., 2000] that the sequence of maximal cliques obtained by ordering the maximal cliques of H according to the order above is a path decomposition of H. As the bags of the path decomposition follow the order above, this path decomposition is consistent with ι. Now, since any path decomposition of H is also a path decomposition of G_ρ, and as $\iota \subseteq \rho$, this path decomposition is also ρ-consistent.

We note that the process of finding all maximal cliques of an interval graph H can be realized in time linear in the size of H [Habib et al., 2000]. Note that since H has pathwidth $\mathcal{O}(pw(G_\rho))$, the number of edges of H is bounded by $\mathcal{O}(pw(G_\rho)^2 \cdot |V|)$. So the process of finding the maximal cliques in H takes time $\mathcal{O}(pw(G_\rho)^2 \cdot |V|)$. Finally, since the most time-expensive part of the process described above is the construction of the tree decomposition $\mathcal{T}$, we have that the whole process takes time $2^{\mathcal{O}(pw(G_\rho))} \cdot |V|$. $\square$

6.2 DP Algorithm for CO

Now, we use the newly introduced notion of consistent path decomposition and the fact that on cocomparability graphs, consistent pathwidth and pathwidth coincide to develop an algorithm for CO parameterized by pathwidth. This algorithm is based on a dynamic programming algorithm over a consistent path decomposition of the cocomparability graph. Using the additional restriction of PCO and PCIO, in Section 6.3, we will push the analysis of the complexity of our algorithm further and we will derive a subexponential algorithm parameterized by the cost of the solution and improve the running time of the currently best branching algorithm. We will extend our algorithm to solve KRA, GbS and OSCM using reductions from those problems to PCO or PCIO.

Theorem 37. *Given a partial order ρ over a set V and a cost function $\mathfrak{c} : V \times V \to \mathbb{N}$ one can solve an instance $(\rho, \mathfrak{c}, k)$ of the* CO *problem in time $\mathcal{O}(|V| \cdot w \cdot 2^w \cdot \log(k) + |V|^2 \cdot \log(k))$ where w is the pathwidth of the cocomparability graph of ρ.*

Given a partial order ρ, one can construct a ρ-consistent path decomposition of width

$\mathcal{O}(pw(G_\rho))$ in time $2^{\mathcal{O}(pw(G_\rho))} \cdot |V|$ using Lemma 36. The rest of the proof of Theorem 37 follows from the next lemma.

Lemma 38. *Given a partial order ρ over a set V, a cost function $\mathfrak{c} : V \times V \to \mathbb{N}$ and a ρ-consistent path decomposition D of the cocomparability graph G_ρ of width w, one can solve an instance $(\rho, \mathfrak{c}, k)$ of the* CO *problem in time $\mathcal{O}(|V| \cdot w \cdot 2^w \cdot \log(k) + |V|^2 \cdot \log(k))$.*

The remainder of this subsection is dedicated to the proof of Lemma 38.

Dynamic Programming Algorithm. Let $\rho \subseteq V \times V$ be a partial order on a set V, $\mathfrak{c} : V \times V \to \mathbb{N}$ be a cost function and S and T be two subsets of V such that for each pair $(s,t) \in S \times T$, $(t,s) \notin \rho$, in other words, no element in S is bigger than any element in T with respect to ρ. We define $\mathfrak{c}(S,T) = \sum_{(s,t)\in(S\times T)\setminus\rho} \mathfrak{c}(s,t)$, this is the cost of having the elements of S before (or smaller than) the elements of T. For every linear extension τ of ρ on T, we let $\mathfrak{c}(\tau) = \sum_{(a,b)\in\tau\setminus\rho} \mathfrak{c}(a,b)$ be the cost of τ. We define $\text{opt}(T) = \min\{\mathfrak{c}(\tau) \mid \tau \in \mathsf{Lin}(\rho, T)\}$. Our goal is to find $\text{opt}(V)$.

Let D be a ρ-consistent path decomposition of width w of the cocomparability graph G_ρ of ρ. By Lemma 33, we can assume without loss of generality that D is nice.

For each position $1 \le i \le 2 \cdot |V|$ in the path decomposition, we compute and store $\mathfrak{c}(L_i, \{v\})$ in table $T_i^{\mathfrak{c}}$, for every vertex $v \in B_i$ such that for each $u \in L_i$, $(v,u) \notin \rho$ and we also compute and store $\text{opt}(L_i \cup T)$ in table T_i^{opt} for each $T \subseteq B_i$. For every vertex $v \in B_i$ such that for each $u \in L_i$, $(v,u) \notin \rho$, $\mathfrak{c}(L_i, \{v\})$ is the cost of having v after the forgotten vertices at position i if this is compatible with ρ and for each $T \subseteq B_i$, $\text{opt}(L_i \cup T)$ is the minimum cost of a linear extension of ρ on $L_i \cup T$. We have $L_{2\cdot|V|} \cup B_{2\cdot|V|} = V$. Therefore, to find the minimum cost of a linear extension of ρ, it is enough to inductively construct these two tables. The induction basis is trivial. For the first bag of the path decomposition, we have $L_1 = \emptyset$ and $|B_1| = 1$, so we have $\mathfrak{c}(L_1, \{v\}) = 0$ for the only vertex $v \in B_1$ in table $T_1^{\mathfrak{c}}$ and $\text{opt}(L_i \cup T) = 0$ for both $T = \emptyset$ and $T = B_1$ in table T_1^{opt}. The following two lemmas explain the induction step of the algorithm.

Lemma 39. *Let $i \in [2, \ldots, 2 \cdot |V|]$. Given the table $T_{i-1}^{\mathfrak{c}}$ that lists the values of $\mathfrak{c}(L_{i-1}, \{v\})$ for every $v \in B_{i-1}$, one can compute $\mathfrak{c}(L_i, \{v\})$ for every $v \in B_i$ in time $w \cdot \log(k)$ in order to build the table $T_i^{\mathfrak{c}}$.*

Proof. We consider two cases. If B_i introduces a vertex v, then $L_i = L_{i-1}$ and $B_i = B_{i-1} \cup \{v\}$. So for $u \in B_i \setminus \{v\}$, $\mathfrak{c}(L_i, \{u\}) = \mathfrak{c}(L_{i-1}, \{u\})$, and, by the consistency of D with ρ, we have that $\forall x \in L_i : (x,v) \in \rho$, so $\mathfrak{c}(L_i, \{v\}) = 0$. If B_i forgets a vertex v, then $L_i = L_{i-1} \cup \{v\}$ and $B_i = B_{i-1} \setminus \{v\}$, so for $u \in B_i$ such that for each $u' \in L_i, (u, u') \notin \rho$, if $(v, u) \in \rho$, then $\mathfrak{c}(L_i, \{u\}) = \mathfrak{c}(L_{i-1} \cup \{v\}, \{u\}) = \mathfrak{c}(L_{i-1}, \{u\})$ otherwise, $\mathfrak{c}(L_i, \{u\}) = \mathfrak{c}(L_{i-1} \cup \{v\}, \{u\}) = \mathfrak{c}(L_{i-1}, \{u\}) + \mathfrak{c}(v, u)$. As the cost can be arbitrarily large, we need $\mathcal{O}(\log(k))$ time to perform the addition of two costs. □

Lemma 40. *Let $i \in [2, \dots, 2 \cdot |V|]$. Given a table $T_i^{\mathfrak{c}}$ that lists the values of $\mathfrak{c}(L_i, \{v\})$ for every $v \in B_i$ such that for each $u \in L_i$ $(v, u) \notin \rho$ and a table T_{i-1}^{opt} that lists the values of $\text{opt}(L_{i-1} \cup T)$ for every $T \subseteq B_{i-1}$, one can compute in $\mathcal{O}(w \cdot 2^w \cdot \log(k))$ time the value of $\text{opt}(L_i \cup T)$ for all $T \subseteq B_i$ in order to build the table T_i^{opt}.*

Proof. The cost can be arbitrarily large, therefore, the addition of two costs is done in time $\mathcal{O}(\log(k))$. First, we compute $\mathfrak{c}(T, \{v'\})$ for $v' \in B_i$ and for $T \subseteq B_i \setminus \{v'\}$, and store the values in an auxiliary table T^{aux}. This computation can be done in $\mathcal{O}(w \cdot 2^w \cdot \log(k))$ time. Now there are two cases:

- If B_i forgets a vertex v, then $L_i = L_{i-1} \cup \{v\}$; for each subset $T \subseteq B_i$, $\text{opt}(L_i \cup T) = \text{opt}(L_{i-1} \cup T \cup \{v\})$ and this value is in the table T_{i-1}^{opt}, as $T \cup \{v\} \subseteq B_{i-1}$.

- If B_i introduces a vertex v, then $L_i = L_{i-1}$ and $B_i = B_{i-1} \cup \{v\}$. Given a subset T of B_i, if $v \notin T$, then $\text{opt}(L_i \cup T)$ is already in the table T_{i-1}^{opt}. Suppose $v \in T$. For all $u \in L_i$, there is no edge between u and v in G_ρ, and as D is consistent with ρ, we have $(u, v) \in \rho$. So in any linear extension of ρ on $L_i \cup T$, the maximum element is a maximal element of T (with respect to ρ). Then we have, by testing all possible maximum elements v':

$$\begin{aligned}\text{opt}(L_i \cup T) &= \min_{v' \in \max_\rho(T)} \{\text{opt}(L_i \cup T \setminus \{v'\}) + \mathfrak{c}(L_i \cup T \setminus \{v'\}, \{v'\})\} \\ &= \min_{v' \in \max_\rho(T)} \{\text{opt}(L_i \cup T \setminus \{v'\}) + \mathfrak{c}(L_i, \{v'\}) + \mathfrak{c}(T \setminus \{v'\}, \{v'\})\}\end{aligned}$$

  where $\max_\rho(T) = \{v \in T \mid \forall u \in T, (v, u) \notin \rho\}$ is the set of maximal elements of T with respect to ρ. The second and third terms are in the tables $T_i^{\mathfrak{c}}$ and T^{aux}, respectively. If $v' = v$, then the first term can be looked up in table T_{i-1}^{opt}. By walking through all $T \subseteq B_i$ with increasing cardinality (recall that always $v \in T$), we can inductively compute $\text{opt}(L_i \cup T)$, as this provides the first term. As the inductive basis, consider $T = \{v\}$, in which case $\text{opt}(L_i \cup T) = \text{opt}(L_i \cup \{v\}) = \text{opt}(L_i)$. The first term is already in the table T_{i-1}^{opt}. The computation of T_i^{opt} can be done in time $\mathcal{O}(w \cdot 2^w \cdot \log(k))$.

This explains how to build the table T_i^{opt}. □

Since $L_{2\cdot|V|} \cup B_{2\cdot|V|} = V$, the dynamic programming algorithm can provide an optimal solution and runs in time $\mathcal{O}(|V| \cdot w \cdot 2^w \cdot \log(k))$. The size of the cost function given as input is $|V|^2 \cdot \log(k)$. Reading the cost function gives the second part of the running time. This proves Lemma 38.

6.3 Further Algorithmic Consequences

Next, we give some additional algorithmic results around CO. First, we show that CO and PCO stay NP-hard even if we restrict the cost of the missing relation in the partial order to $\{0, 1\}$ for CO and to $\{1, 2\}$ for PCO. Next, we analyse the algorithm for CO given in Section 6.2 when applied to PCO and PCIO and give a subexponential running time when the parameter is the cost of the solution (Theorem 41 and Theorem 50). Last, we give a kernel and a better branching algorithm for PCO (Theorem 52 and Theorem 55).

6.3.1 Tighter NP-hardness

The Feedback Arc Set on Tournaments problem, FAST for short, is a well-known NP-complete problem [Charbit et al., 2007]. A connection between FAST and PCO was exploited in [Fernau et al., 2014] to solve PCO using FAST. The relation between variants of FAST and CO range in both directions. We are now explaining a reverse reduction. We can even reduce from Constrained FAST, a generalization of FAST, introduced by van Zuylen and Williamson [Zuylen and Williamson, 2009] and Brandenburg *et al.* [Brandenburg et al., 2013]. It is defined as follows: The arc set of a given tournament graph is split into fixed arcs A_{fix} and free arcs A_{free}. The task is to remove at most k free arcs such that the resulting graph becomes acyclic. Constrained FAST can model FAST by assigning all the arcs of tournament to A_{free}. We know that every arc in A_{free} that contradicts the transitivity of A_{fix} needs to be removed. Therefore, we assume that A_{fix} gives a transitive relation on the set of vertices and that it is acyclic, so that it defines a partial order ρ on the vertex set V. By defining the following cost function, we can solve Constrained FAST with any CO algorithm: For arcs $(x, y) \in A_{\text{free}}$, we set $\mathfrak{c}(x, y) = 0$. For arcs (x, y) such that $(y, x) \in A_{\text{free}}$, we set $\mathfrak{c}(x, y) = 1$. By the tournament condition, for each edge $\{x, y\}$ of G_ρ, $\mathfrak{c}(x, y) \in \{0, 1\}$ and $\mathfrak{c}(y, x) \in \{0, 1\}$ are defined, with $\mathfrak{c}(x, y) + \mathfrak{c}(y, x) = 1$. As FAST is an NP-complete problem, this also shows NP-completeness for CO (even when the cost are restricted to 0 and 1) and similarly, by adding one to all arc costs, we obtain NP-completeness for PCO, even when the costs are restricted to 1 and 2.

6.3.2 Sub-exponential Algorithm for PCO

For special cases of PCO, such as those arising from KRA or from the graph-drawing problem OSCM, sub-exponential time algorithms have been known, i.e., algorithms with running times of the form $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$. In contrast, for the more general problem of PCO, only algorithms with running time $\mathcal{O}^\star(k^{\sqrt{k}})$ were known before, where k is the cost parameter [Fernau et al., 2014]. Here, we prove that PCO also admits algorithms of the form $\mathcal{O}^\star(2^{\mathcal{O}(\sqrt{k})})$, by making use of several structural insights for cocomparability graphs. More precisely, we prove the following theorem.

Theorem 41. *Given a partial order $\rho \subseteq V \times V$ and a cost function $\mathfrak{c} : V \times V \to \mathbb{N}$, one can solve a* PCO *instance $(\rho, \mathfrak{c}, k)$ in time $|V| \cdot 2^{\mathcal{O}(\sqrt{k})} + \mathcal{O}(|V|^2 \cdot \log(k))$.*

The remainder of this subsection is dedicated to the proof of Theorem 41.

In general graphs, treewidth is more expressive than pathwidth in the sense that any graph of pathwidth k has also treewidth at most k, but there exist graphs of treewidth k whose pathwidth is unbounded. Nevertheless, in the class of cocomparability graphs, treewidth and pathwidth coincide.

Lemma 42 (Theorem 1.2 in [Habib and Möhring, 1994]). *Let G be a cocomparability graph. Then, $pw(G) = tw(G)$.*

Let $t \geq 3$ be an integer, we write C_t for the cycle on t vertices and $C_{\geq t} = \{C_k \mid k \geq t\}$. We say that G is $C_{\geq t}$-free if G excludes all C_k as an induced subgraph for any $k \geq t$.

Lemma 43. *Cocomparability graphs are $C_{\geq 5}$-free.*

This fact is known but not that easy to track down in the literature. We refer to [Ghouila-Houri, 1964, Gilmore and Hoffman, 1964b, Gallai, 1967], which contain corresponding results on comparability graphs. We also refer to the textbook of Trotter [Trotter, 1992]. To keep the thesis self-contained, we present a short self-contained proof of the fact that cocomparability graphs are C_k free for $k \geq 5$. As a key notion, we consider bad triples in cocomparability orders. A *cocomparability order* of a cocomparability graph $G = (V, E)$ is a bijection $\sigma : V \to \{1, \ldots, |V|\}$ that linearly extends a transitive orientation ρ of G, meaning that $(x, y) \in \rho$ implies $\sigma(x) < \sigma(y)$. Given a graph G and a linear order σ of its vertices, a *bad triple* is three vertices x, y, z so that $\sigma(x) < \sigma(y) < \sigma(z)$, $xy \notin E$, $yz \notin E$, and $xz \in E$. Notice that if G is a cocomparability graph and σ a cocomparability order then (G, σ) has no bad triple. Namely, if x, y and y, z are comparable in some partial order ρ with $\sigma(x) < \sigma(y) < \sigma(z)$, then (as σ extends ρ) whenever $x <_\rho y$ and $y <_\rho z$ then $x <_\rho z$ is enforced by transitivity, ruling out a bad triple.

Lemma 44. *A cycle C_k on $k \geq 5$ vertices is not a cocomparability graph.*

Proof. Suppose for contradiction that $\sigma : V(C_k) \to \{1, \dots, k\}$ is a cocomparability order of C_k with edge set E_k. Define $\sigma^{-1}(i)$ to be the vertex x in C_k so that $\sigma(x) = i$. Let $u = \sigma^{-1}(1)$ and $v = \sigma^{-1}(k)$. If $uv \in E_k$, let x be any vertex non-adjacent to both u and v (such a vertex exists since $k \geq 5$), we have that u, x, v is a bad triple. We conclude that $uv \notin E_k$. Let P and Q be the two paths that connect u and v in C_k, excluding u, v. Without loss of generality, $|V(P)| \geq |V(Q)|$. Since $k \geq 5$, we have that the path P contains an edge $\{p, q\}$ so that $\{p, q\} \cap \{u, v\} = \emptyset$. Without loss of generality, $\sigma(p) < \sigma(q)$.

First, suppose that there is a vertex $\ell \in V(Q)$ so that $\sigma(p) < \sigma(\ell) < \sigma(q)$. Then, p, ℓ, q is a bad triple. Hence, such a vertex cannot exist. It follows that some edge $\{a, b\}$ of $E(C_k[V(Q) \cup \{u, v\}])$ is such that $\sigma(a) < \sigma(p)$ and $\sigma(q) < \sigma(b)$. Since the path Q has at least one internal vertex, we have that $|\{a, b\} \cap \{u, v\}| \leq 1$, and therefore (since each of u and v has at most one neighbor in $\{p, q\}$) we have $|N(\{a, b\}) \cap \{p, q\}| \leq 1$. However, if $p \notin N(\{a, b\})$, then a, p, b is a bad triple. If $q \notin N(\{a, b\})$, then a, q, b is a bad triple. But $|N(\{a, b\}) \cap \{p, q\}| \leq 1$ implies that one of the two former cases must hold, yielding the desired contradiction. □

Finally, as the proof is based on the notion of bad triples and as bad triples in induced subgraphs are also bad triples in the whole graph, this proves Lemma 43.

The following statement is put forward in [Chudnovsky et al., 2020, Theorem 1.5].

Lemma 45 (Theorem 1.5 in [Chudnovsky et al., 2020])**.** *A $C_{\geq t}$-free graph with maximum degree Δ has treewidth bounded by $\mathcal{O}(t \cdot \Delta)$. Furthermore, a tree decomposition of this width can be computed in polynomial time.*

From Lemma 45, we can prove the following lemma.

Lemma 46. *Let G be a $C_{\geq 5}$-free graph, let m be the number of edges of G. Then, we have $m = \Omega(tw(G)^2)$.*

Proof. A graph on m edges has at most $2\sqrt{m}$ vertices of degree at least $\sqrt{m}$. Let G' be the graph obtained from G by removing all vertices of degree at least $\sqrt{m}$. Since G is $C_{\geq 5}$-free, so is G'. Therefore, by applying Lemma 45, we get that $tw(G') = \mathcal{O}(\sqrt{m})$. The removal of a vertex reduces the treewidth by at most 1. Hence, we have $tw(G) - 2\sqrt{m} \leq tw(G') = \mathcal{O}(\sqrt{m})$. This implies that $tw(G) = \mathcal{O}(\sqrt{m})$, or conversely, that $m = \Omega(tw(G)^2)$. □

By combining Lemma 46 with Lemma 42, we get:

Lemma 47. *Let G be a cocomparability graph and let m be the number of edges of G. Then, $m = \Omega(pw(G)^2)$.*

Now, in a PCO instance, each edge contributes at least 1 to the cost of any solution. Therefore, if a solution has cost at most k, then the cocomparability graph of the input partial order can have at most k edges. Therefore, this observation, together with Lemma 47 yields the following lemma.

Lemma 48. *Let $(\rho, \mathfrak{c}, k)$ be an YES-instance of* PCO*. Then, $pw(G_\rho) = \mathcal{O}(\sqrt{k})$.*

To get the running time of Theorem 41, we need to either compute a ρ-consistent path decomposition of width at most $\mathcal{O}(\sqrt{k})$, or to trigger an early rejection. For this, we will use the following lemma which is based on Lemma 36.

Lemma 49. *There is an algorithm running in time $2^{\mathcal{O}(\sqrt{k})} \cdot |V|$ that takes an instance $(\rho, \mathfrak{c}, k)$ of* PCO *as input, and either constructs a ρ-consistent path decomposition of the graph G_ρ of width $\mathcal{O}(\sqrt{k})$, or determines that this instance is a NO-instance.*

Now we are ready to prove the statement of Theorem 41. Given an instance $(\rho, \mathfrak{c}, k)$ of PCO, we apply the algorithm given by Lemma 49. This algorithm either determines that the instance is a NO-instance, or constructs a ρ-consistent path decomposition D of G_ρ of width $\mathcal{O}(\sqrt{k})$. In the first case, we are done and simply answer NO. Otherwise, we give both the instance $(\rho, \mathfrak{c}, k)$ and the decomposition D to the algorithm stated in Lemma 38 to determine in time $|V| \cdot 2^{\mathcal{O}(\sqrt{k})} + \mathcal{O}(|V|^2 \cdot \log(k))$ whether $(\rho, \mathfrak{c}, k)$ is a YES- or a NO-instance of PCO. It is worth noting that in case this is a YES-instance, the algorithm also constructs a linear extension of ρ of cost at most k. This concludes the proof of Theorem 41.

6.3.3 Completing an interval order is easier

Recall that PCIO has one more restriction compared to PCO, the input partial order is an interval order. As such, Theorem 41, given in the previous subsection, gives a sub-exponential algorithm for PCIO parameterized by the cost of the linear extension. Working with an interval order instead of a partial order allows us a better analysis of the complexity of the algorithm. Therefore, we get an explicit and better bound for our dynamic programming algorithm.

Theorem 50. *An instance $(\iota, \mathfrak{c}, k)$ of the* PCIO *problem is solvable in time $\mathcal{O}(k \cdot 2^{\sqrt{2k}} \cdot \log(k) + |V|^2 \cdot \log(k))$.*

The following is an outline of our algorithm, called DP-PCIO.

1. Construct G_ι, if G_ι has more than k edges then stop with "NO". This can be done in time $|V|^2$. This is justified, because $\mathfrak{c}(x, y) > 0$ for each incomparable pair $\{x, y\}$.
2. Construct a nice path decomposition D consistent with ι. If the width of D is more than $\sqrt{2k}$, then stop with "NO", as a large clique was detected.
3. Compute $\text{opt}(V)$ by a dynamic programming algorithm based on the path decomposition D. If the current optimum solution is bigger than k, then stop with "NO". If the computation is successful and $\text{opt}(V) \leq k$, then answer "YES". Otherwise, answer "NO".

To apply our dynamic programming algorithm of Lemma 38, we need a consistent path decomposition. Let $D = (B_1, \ldots, B_{2|V|})$ be the nice path decomposition of G_ι consistent with ι that we got by applying Lemma 33 on the path decomposition of Lemma 32. By construction, each bag in D is a clique.

We are now proving Theorem 50 with the help of the following technical lemma.

Lemma 51. *Assume that G_ι has at most k edges. Let $H = \lceil\sqrt{2k}\rceil + 1$ and, for each $2 \leq h \leq H$, let $c_h \doteq |\{i \mid h = |B_i|\}|$. Then, we have $c_h \leq k/(h-1) - h/2 + 1$.*

Proof. Let $h \in \{2, \ldots, H\}$. Let $i_1 < i_2 < \ldots < i_{c_h}$ be the elements of $\{i \mid h = |B_i|\}$. To prove the claim, we will associate some edges of G_ι to each bag B_{i_j}. Each edge will be associated to at most one bag but will not necessarily belong to that bag. As B_{i_1} is a clique, we associate the $h(h-1)/2$ edges of the clique to it. Now, we will associate at least $h-1$ edges to each of the remaining bags. If $B_{i_j} \neq B_{i_{j-1}}$, then there is a vertex v in $B_{i_j} \setminus B_{i_{j-1}}$, because both $B_{i_{j-1}}$ and B_{i_j} are (different) cliques. Then, we associate the $h-1$ edges incident to v in B_{i_j} to B_{i_j}. If $B_{i_j} = B_{i_{j-1}}$, then $B_{i_{j-1}+1}$ cannot forget any vertices from $B_{i_{j-1}}$, so $B_{i_{j-1}+1}$ introduces a vertex v and we associate the h edges adjacent to v in $B_{i_{j-1}+1}$ to B_{i_j}. So G_ι contains at least $h(h-1)/2 + (h-1)(c_h - 1)$ many edges. The costs for incomparable pairs are non-zero, so each edge contributes at least one in the final cost of any linear order. Therefore, we have $h(h-1)/2 + (h-1)(c_h-1) \leq k$. ☐

Proof of Theorem 50. The first step of our algorithm (constructing the cocomparability graph $G_\iota = (V, E)$) takes $\mathcal{O}(|V|^2)$ time. Now, we know that $|E| \leq k$, if not we abort with NO. The second step takes $\mathcal{O}(|V| + |\iota|) \subseteq \mathcal{O}(|V|^2)$ time by Lemma 32. We have a nice path decomposition D thereafter that is consistent with ι. As bags induce cliques, having at least $\sqrt{2k}+1$ many vertices in a bag means having at least $(\sqrt{2k}+1)\sqrt{2k}/2 > k$

many edges. As soon as we detect such a situation, the computation is aborted with NO. Finally, upon doing the dynamic programming, we can check that the current costs are not bigger than k.

Let H, $h \in \{2, \ldots, H\}$ and c_h be as defined in Lemma 51. For each bag of size h, the running time of the algorithm based on Lemma 39 and Lemma 40 is $h2^h \log(k)$. Therefore, the total running time of the dynamic programming part of the algorithm is $\mathcal{O}(\sum_{h=2}^{H} c_h h 2^h \log(k) + |V|)$ By Lemma 51, we have

$$\begin{aligned}
\sum_{h=2}^{H} c_h h 2^h &\leq \sum_{h=2}^{H} (\frac{k}{h-1} - \frac{h}{2} + 1) h 2^h \\
&\leq k \cdot \sum_{h=2}^{H} \frac{h}{h-1} \cdot 2^h + \sum_{h=2}^{H} (1 - \frac{h}{2}) \cdot h \cdot 2^h \\
&\leq k \cdot \sum_{h=2}^{H} \frac{h}{h-1} \cdot 2^h && (1 - \frac{h}{2}) \leq 0 \\
&\leq 2k \cdot \sum_{h=2}^{H} 2^h && \frac{h}{h-1} \leq 2 \\
&\leq 2k 2^{H+1} \\
&= \mathcal{O}(k 2^{\sqrt{2k}})
\end{aligned}$$

Reading the interval order ι in the input takes time $\mathcal{O}(|V|^2 \cdot \log(k))$. This gives the second part of the running time. □

6.3.4 Some other FPT results for Positive Completion of an Ordering

We collect here two FPT results for Positive Completion of an Ordering.

1. We provide a small kernel for PCO of size $1.5k$ (Theorem 52).
2. We give a better search tree algorithm for PCO running in time $\mathcal{O}^\star(\sqrt{2}^k)$ (Theorem 55).

Kernel for PCO

In the following, let ρ be a partial order that should be extended to a linear order. Let G_ρ be the corresponding cocomparability graph. We also consider k as the standard parameter, upper-bounding the linearization cost.

We observe a *kernelization scheme* for Positive Completion of an Ordering, called RRPCOq in the following. For any *fixed* $q > 1$ do: For each connected component C of G_ρ with $|C| \leq q$ vertices, solve PCO optimally on the induced graph $G_\rho[C]$. The reduced instance will see the orders between all pairs of vertices from C settled, and the parameter will be reduced accordingly.

We also (clearly) have the following rule RRPCOcleanup: If $v \in V$ such that for all $u \in V$, either $(u, v) \in \rho$ or $(v, u) \in \rho$ have been settled, in other words, v is an isolated vertex in G_ρ, then we can remove v from V, ρ, and G_ρ. Notice that if rule RRPCOq was used to resolve some vertex set C, then RRPCOcleanup will delete all vertices of C from the instance. Namely, as C was a connected component of G_ρ, each vertex $v \in C$ and each vertex $u \notin C$ were comparable.

After exhaustively applying these rules, every connected component C of G_ρ has at least $q + 1$ many vertices and thus at least q edges. This means that at least q arcs with a weight of at least one per arc have to be added per component, therefore there are at most k/q many components. The more vertices (and hence edges) there are per component in G_ρ, the fewer components there are. The worst case is indeed when all components have $q + 1$ many vertices, so that overall we have at most $k\frac{q+1}{q}$ many vertices in the instance, an expression approaching k for growing q. Considering q as a fixed constant, the running time of this kernelization is still linear in the number of vertices of the graph respectively the size of the base set.

Theorem 52. *Fix some* $1 < \alpha \leq 1.5$. *Then, each instance of* Positive Completion of an Ordering *admits a problem kernel of size* αk.

Remark 53. Notice that the two rules RRPCOq and RRPCOcleanup do not destroy special structures on the orders, so that we obtain in particular linear-size kernels for Positive Completion of an Interval Ordering.

Branching algorithm

Finally, we show how our considerations also help to improve the running time of a simple branching algorithm. The algorithm works as follows: it picks an edge in the co-comparability graph and considers orienting it both ways. As long as there are *profitable edges* that cause at least a cost of two in each branch, we keep on branching. Costs can also be implicitly caused, as we modify the partial order ρ and hence transitivity must be maintained. We can use Theorem 50 when there are no more profitable edges because of the following lemma.

Lemma 54. *After exhaustively branching at all profitable edges, G_ρ is an interval graph.*

This is the key to the following improvement on the branching algorithm described in [Fernau, 2005]. Notice that in practice, branching algorithms tend to be faster at least for small parameter values, due to the smaller constants in the basis of the (sub-)exponential functions that upper-bound the running times.

Theorem 55. PCO *can be solved in time $\mathcal{O}^\star(\sqrt{2}^k)$ by a branching algorithm.*

No, we will formally describe and explain this branching algorithm. See Algorithm 1 for a detailed description of the algorithm. It is a blend of the branching algorithm explained in [Fernau, 2005] and our findings on interval graphs. In order to understand it, we will need the following definitions.

- If $R \subseteq V \times V$ is a relation, then R^* denotes is reflexive transitive closure.
- We are growing a partial order $\rho \subseteq V \times V$ by adding further pairs (a, b). When adding a pair, not only the cost $\mathfrak{c}(a, b)$ has to be considered, as often $|(\rho \cup \{(a, b)\})^* \setminus \rho| > 1$ by transitivity. We therefore set $\mathfrak{c}^*_\rho(a, b) \doteq \mathfrak{c}((\rho \cup \{(a, b)\})^* \setminus \rho)$.
- We call (a branching at) a pair of elements $\{a, b\}$ where $\{(a, b), (b, a)\} \cap \rho = \emptyset$ (i.e., a, b are incomparable) a *profitable edge* (in the cocomparability graph) if $\mathfrak{c}^*_\rho(a, b) \geq 2$ and $\mathfrak{c}^*_\rho(b, a) \geq 2$.

We are now going to show that Positive Completion of an Ordering can be solved in time $\mathcal{O}^\star(\sqrt{2}^k)$.

Proof of Theorem 55. By definition, a profitable edge yields a branching vector of at worst $(2, 2)$ and hence the claimed running time for the pure branching part of the algorithm. Furthermore, in case that G_ρ is an interval graph, our problem can be solved in time $\mathcal{O}^\star(2^{\sqrt{2k}}) \subseteq \mathcal{O}^\star(\sqrt{2}^k)$ due to Theorem 50. This explains the running time.

To prove the correctness, we have to prove Lemma 54. □

Proof of Lemma 54. For the sake of contradiction, assume G_ρ is not an interval graph. Then, it contains an induced C_4. Recall that interval graphs are exactly the cocomparability graphs that do not contain an induced C_4 [Gilmore and Hoffman, 1964a]. Without loss of generality, this means that there are four elements a, b, c, d with $\{(a, c), (b, d)\} \subseteq \rho$, and these are the only pairs from $\{a, b, c, d\}$ related by ρ. Now, consider branching on the pair $\{a, b\}$. If (a, b) is fixed, then transitivity will enforce (a, d). Hence, $\mathfrak{c}^*_\rho(a, b) \geq 2$. By symmetry, fixing (b, a) also fixes (b, c), so that $\mathfrak{c}^*_\rho(b, a) \geq 2$. Hence, $\{a, b\}$ is a profitable edge that cannot exist after the assumed exhaustive branching. □

Algorithm 1 A branching algorithm for POSITIVE COMPLETION OF AN ORDERING, called PCO

Input(s): A partial order $\rho \subseteq V \times V$ over a set V, a cost function $\mathfrak{c} : V \times V \to \mathbb{N}_{>0}$, and an integer k

Output(s): YES iff the given PCO instance has a solution of size at most k

```
    repeat
        Exhaustively apply the reduction rules, RRPCO$q$ and RRPCOcleanup, adjusting
        $\rho$, $\mathfrak{c}$, and $k$ accordingly.
        Determine the elements whose order is settled by transitivity and adjust accord-
        ingly.
    until there are no more changes
 5: if $k < 0$ then
        return NO.
    else if there is a profitable edge $\{a, b\}$ then
        if PCO$((\rho \cup \{(a, b)\})^*, k - \mathfrak{c}^*_\rho(a, b), \mathfrak{c})$ then
            return YES
10:     else
            return PCO$((\rho \cup \{(b, a)\})^*, k - \mathfrak{c}^*_\rho(b, a), \mathfrak{c})$
        end if
    else
        Compute an optimum path decomposition for the resulting cocomparability graph
        which is an interval graph
15:     Solve the rest of the instance using Theorem 50
        return YES or NO accordingly.
    end if
```

6.4 Reduction from OSCM to PCIO

Now, we show that OSCM can be reduced to PCIO, starting with a simple remark.

Remark 56. Isolated vertices in V_2, do not interact with the drawing and can be placed anywhere in an optimal linear order of V_2.

From here, we assume that V_2 does not contain any isolated vertices. Similar to [Fernau, 2005, Fernau et al., 2014], we can model OSCM instances as PCIO instances.

Lemma 57. *Given an instance (G, τ_1, k) of* OSCM*, one can construct in polynomial time an equivalent instance $(\iota, \mathfrak{c}, k)$ of* PCIO*.*

Proof. Let a bipartite graph $G = (V_1, V_2, E)$, a linear order τ_1 on V_1, and an integer k be an instance of the OSCM problem. From this instance, we construct an equivalent instance $(\iota,\ \mathfrak{c}, k)$ of the PCIO problem. This reduction is similar to an algorithm from Kobayashi and Tamaki in [Kobayashi and Tamaki, 2015]; also confer [Fernau, 2005, Fernau et al., 2014] for such type of reductions.

First, we set $V = V_2$. Given a linear order τ_2 on V_2, we define the *cost* of τ_2, $\mathfrak{c}(\tau_2)$, as the number of edge crossing in the two-layer drawing of G following τ_1 and τ_2. For every u and v from V_2, we define the *cost* of (u, v), $\mathfrak{c}(u, v)$, as the number of edge crossing between edges incident to u and edges incident to v when u is placed before v in a two-layer drawing of G. We remark that, similarly to the defining equation for PCO,

$$\mathfrak{c}(\tau_2) = \sum_{u <_{\tau_2} v} \mathfrak{c}(u, v)\,. \tag{6.1}$$

Let u and v be two distinct vertices in V_2. $\mathfrak{c}(u, v) = 0$ and $\mathfrak{c}(v, u) = 0$ if and only if $N(u) = \{x\} = N(v)$, as V_2 contains no isolated vertices.

Now we define a partial order ι on V_2 as follows: if $\mathfrak{c}(u, v) = 0$ and $\mathfrak{c}(v, u) > 0$, then $(u, v) \in \iota$ and if $\mathfrak{c}(u, v) = 0$ and $\mathfrak{c}(v, u) = 0$, then we choose either $(u, v) \in \iota$ or $(v, u) \in \iota$ arbitrarily. Given a linear extension τ_2 of ι, by [Dujmovic and Whitesides, 2004, Lemma 1], [Kobayashi and Tamaki, 2015, Lemma 2] and [Kobayashi and Tamaki, 2015, Corollary 1], we have that the number of edge crossing in the drawing defined by τ_1 and τ_2 is equal to the cost of τ.

Now, we will show that ι is an interval order by giving an interval representation of it. To do that, we will define a set X of points and a linear order τ_X on X. Then, using those points on the real line, we will define an interval for each vertex in V_2.

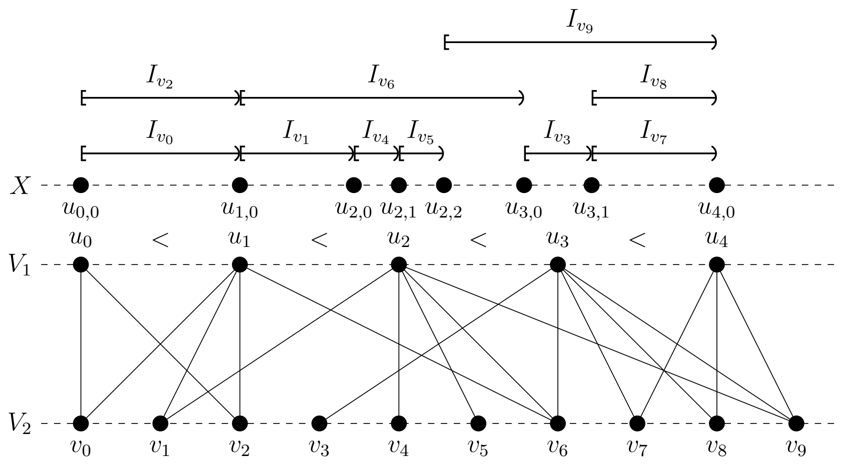


Figure 6.2: Reduction from OSCM to PCO: Example of the construction of the interval representation of the partial order associated with an OSCM instance.

Figure 6.2 illustrates this construction on an example. For each vertex u in V_1, let $V_u \doteq \{v_1, \ldots, v_r\} \subseteq N(u)$ be the set of vertices in the neighborhood of u of degree 1 such that $v_1 <_\iota v_2 <_\iota \cdots <_\iota v_r$. We say that v_i is the i-th neighbor of u of degree 1. We define $X_u \doteq \{u_0, \ldots, u_r\}$, a set of points associated to u. Note that there is one more point than neighbors of degree 1. The points in X_u are ordered as follows: $u_0 <_{\tau_X} u_1 <_{\tau_X} \cdots <_{\tau_X} u_r$. Then, we set $X = \bigcup_{u \in V_1} X_u$. The order in τ_X between points in sets X_u and X_v follows τ_1, as $u, v \in V_1$. Given a vertex v in V_2, we will define the half-open interval I_v associated to v. If v has degree 1, then let u be its neighbor and let i be its position in V_u: we define $I_v \doteq [u_{i-1}, u_i)$. If $v \in V_2$ has degree at least 2, then let $l = \min_{\tau_1}\{u \in V_1 \mid (u, v) \in E\}$ and $r = \max_{\tau_1}\{u \in V_1 \mid (u, v) \in E\}$, then we define $I_v \doteq [l_{|V_l|}, r_0)$. Note that in that case $r \neq l$. One can verify that $\{I_v \mid v \in V_2\}$ is an interval representation of ι. □

As PCIO has not been formally studied in the literature, (basically apart from results in [Fernau, 2005, Fernau et al., 2014]), let us draw an important consequence from the previous lemma (also see the discussion in the beginning of Subsection 6.3.1). Note that the following corollary leaves some room for improvement with respect to the set of allowed weights.

Corollary 58. Positive Completion of an Interval Ordering *is* NP*-complete, even when restricted to instances $(\iota, \mathfrak{c}, k)$ where the arc weights are within the set $\{1, 2, \ldots, 16\}$.*

Proof. PCIO is a special case of CO, therefore it is contained in NP. We use the reduction from OSCM as described in the previous lemma to prove the claim. Due to

[Muñoz et al., 2001], we know that OSCM-instances are NP-hard even if all vertices in V_2 have at most four neighbors. Therefore, considering there is at most four edges incident to $u \in V_2$ and there is at most four edges incident to $v \in V_2$, they cannot incur more than 16 edge crossings, which naturally upper-bounds the cost $\mathfrak{c}(u, v)$. □

Remark 59. We can use Theorem 50 to immediately deduce an algorithm for OSCM matching the running time $\mathcal{O}^\star(2^{\sqrt{2k}})$ of the best published algorithm for OSCM, which is due to Kobayashi and Tamaki [Kobayashi and Tamaki, 2015]. We could also use the PCO-kernelization as a kernelization procedure for OSCM.

Remark 60. Çakiroglu *et al.* [Çakiroglu et al., 2009] studied the variation where edges (if existing) have positive weights, and the cost of an edge crossing is obtained by the product of the weights of the crossing edges. This modification (with applications in automatic graph drawing) can also be modeled by PCIO so that we inherit an $\mathcal{O}^\star(2^{\sqrt{2k}})$ algorithm for the standard parameter k.

6.5 Reduction from KRA to PCO

Now we will show that KRA can be encoded into PCO. Let (Π, C) be an instance of Kemeny Rank Aggregation with m votes $\Pi = (\pi_1, \ldots, \pi_m)$ over n candidates C. From this instance, we construct an equivalent instance of the PCO problem $(\rho, \mathfrak{c})$ with base set $V = C$. For every pair of candidates c_1 and c_2, we define the *cost* of (c_1, c_2), $\mathfrak{c}(c_1, c_2)$, as the number of votes that do not order c_1 before c_2. More formally, $\mathfrak{c}(c_1, c_2) = |\{i \in [m] \mid c_2 <_{\pi_i} c_1\}|$.

Lemma 61. *Given two candidates c_1 and c_2, if, for every vote $\pi_i \in \Pi$, we have $c_1 <_{\pi_i} c_2$ then for every Kemeny consensus π, $c_1 <_\pi c_2$.*

Using different terminology, a proof of this lemma can be found in [Monjardet, 1973, Théorème 3]. To make the thesis self-contained, we provide a short proof below.

Proof. Let (Π, C) be an instance of KRA such that for each vote $\pi_i \in \Pi$, we have $c_1 <_{\pi_i} c_2$ for some candidates $c_1, c_2 \in C$. Let π be a consensus such that $c_2 <_\pi c_1$. We will show that modifying π by moving c_1 just before c_2 or c_2 just after c_1 we can strictly reduce the cost of π. We split the impact of this modification on the cost in two. First changing the order of c_1 and c_2 in π will reduce the cost by m. Then we will look at candidates between c_1 and c_2 in π. Let $K_1 = \sum_{c_2 <_\pi c_k <_\pi c_1} |\{i \in [m] \mid c_k <_{\pi_i} c_1\}|$ be the increase of the cost for moving c_1 before every candidates between c_1 and c_2. As $c_1 <_{\pi_i} c_2$ for all i, we have $K_1 = \sum_{c_2 <_\pi c_k <_\pi c_1} |\{i \in [m] \mid c_k <_{\pi_i} c_2\}|$ which is the decrease of the cost for moving c_2 after every candidates between c_1 and c_2. Similarly, let $K_2 = \sum_{c_2 <_\pi c_k <_\pi c_1} |\{i \in [m] \mid c_2 <_{\pi_i} c_k\}| = \sum_{c_2 <_\pi c_1 <_\pi c_k} |\{i \in [m] \mid c_k <_{\pi_i} c_1\}|$ be the increase of the cost for moving c_2 after every candidates between c_1 and c_2 and also the decrease of the cost for moving c_1 before c_2. Exchanging the order of c_1 and c_2 decrease the cost by m. Therefore, if $K_1 \leq K_2$, then moving c_1 before c_2 strictly reduces the cost of π, otherwise moving c_2 after c_1 strictly reduces the cost of π. □

Now, we define the partial order ρ as follows: $(c_1, c_2) \in \rho$ if and only if $\mathfrak{c}(c_1, c_2) = 0$. Hence, $<_\rho = \bigcap_{i=1}^{m} <_{\pi_i}$ is the *unanimity order* [Charon and Hudry, 2007]. The *unanimity width* of a set of votes Π is defined as the pathwidth of the cocomparability graph of the unanimity order ρ.

By Lemma 61, a ranking π, which is a linear order of the candidates, is a Kemeny consensus if and only if π is a linear extension of ρ of minimum cost. The Kemeny score of π is[3]

$$\sum_{i=1}^{m} \text{KT-dist}(\pi, \pi_i) = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{n} [c_j <_{\pi_i} c_k \wedge c_k <_\pi c_j] = \sum_{j=1}^{n} \sum_{k=1}^{n} \mathfrak{c}(c_k, c_j)[c_k <_\pi c_j] \quad (6.2)$$

and is equal to the cost of the linear extension given by π according to its definition. These considerations prove that we can translate our algorithmic results for PCO to KRA.

Remark 62. Our reduction works even if votes are reflexive and antisymmetric relations instead of linear orders.

Using the reduction from KRA to PCO and Theorem 25, we can conclude that our subexponential time algorithm parameterized by the cost of a solution for PCO in Theorem 41 is optimal under ETH.

Corollary 63. PCO *on instances with cost only in* $\{1, 2, 3\}$ *and some integer* k *bounding the cost of a linear extension can be solved neither in time* $\mathcal{O}^\star\left(2^{o(|V|)}\right)$ *nor in time* $\mathcal{O}^\star\left(2^{o(\sqrt{k})}\right)$ *unless* ETH *fails.*

Pathwidth in Kemeny Rank Aggregation. Now we will discuss the meaning of the pathwidth measure from the PCO problem applied to KRA. For KRA, several measures have been studied in the context of parameterized complexity, Betzler et al. [Betzler et al., 2009] introduced the notion of *maximum range of candidate positions*. For an election (Π, C), the *range* $r(c)$ of a candidate c is defined as

[3]Recall the bracket notation: if p is a logical proposition, then $[p]$ yields 1 if p is true and else, $[p]$ yields 0 if p is false.

$r(c) \doteq \max_{i,j\in[m]} |\pi_i^{-1}(c) - \pi_j^{-1}(c)| + 1$. If $\Pi(c) \doteq \{i \in [|C|] \colon \exists \pi \in \Pi : \pi(i) = c\}$ denotes the set of positions the candidate c received in election (Π, C), then $r(c) = \max(\Pi(c)) - \min(\Pi(c)) + 1$. The *maximum range* $r_{\max}$ of an election is given by $r_{\max} \doteq \max_{c\in C} r(c)$. Betzler et al. [Betzler et al., 2009] proved that KRA can be solved in time $\mathcal{O}(32^{r_{\max}} \cdot (r_{\max}^2 \cdot |C| + r_{\max} \cdot |C|^2 \log |C| \cdot m) + m^2 \cdot |C| \log |C|) = \mathcal{O}^\star(2^{5r_{\max}})$.

We can relate the pathwidth measure from the PCO problem applied to KRA with the maximum range of candidate positions.

Lemma 64. *Given an election* (Π, C)*, let* w *be the consistent pathwidth associated to the election and* $r_{\max}$ *be the maximum range of the election. We have* $w \leq 2 \cdot r_{\max} - 2$.

Proof. Let ρ be the partial order defined by the election. To prove this statement, we will construct an interval order ι such that $\iota \subseteq \rho$, and $w(\iota) \leq 2 \cdot r_{\max}$. To each candidate $c \in C$, we associate the interval $I_c \doteq [\min(\Pi(c)) - 1, \max(\Pi(c))\,)$. (We subtract one from the left border to avoid empty intervals.) We let ι be the interval order associated with the interval representation $\{I_c \mid c \in C\}$. By Lemma 61, we have that ρ is an extension of the interval order ι. Each interval I_c has length at most $r_{\max}$. Thus, there are at most $2 \cdot r_{\max} - 2$ intervals that intersect at one point in the interval representation. Hence, $w(\iota) \leq 2 \cdot r_{\max} - 2$. □

In this proof, $\{I_c \mid c \in C\}$ is an interval representation of an interval order ι such that ι is a suborder of ρ and $w(\iota) \leq 2 \cdot r_{\max} - 2$. By Lemma 32, one can use this construction to build a ρ-consistent path decomposition of width at most $2 \cdot r_{\max} - 2$ given an election (Π, C). Hence, Lemma 38 yields the following noticeable improvement to the mentioned result of Betzler et al. [Betzler et al., 2009]:

Corollary 65. KRA *can be solved in time* $\mathcal{O}(|C| \cdot r_{\max} \cdot 2^{2r_{\max}} + m \cdot |C|^2) = \mathcal{O}^\star(2^{2r_{\max}})$.

6.6 Reduction from GbS to OSCM

In this section, we will show that GbS can be encoded into OSCM. With the same idea as in the proof of Theorem 29, we can also reduce GbS to OSCM by representing the string w as the ordered vertex set V_1 and Σ as the vertex set V_2 (see Figure 6.3). More precisely, let n be the length of w and interpret w as a mapping from $[n]$ into Σ. Moreover, set $V_1 = [n]$ with the usual linear order $<_{\tau_1} \doteq <$ on $[n]$. Let $V_2 = \Sigma$ and connect $a \in V_2$ to $i \in [n]$ if and only if $w(i) = a$. This defines the bipartite graph $G = (V_1, V_2, E)$ with linear order τ_1 on V_1. Now, the GbS instance (w, k) is a YES-instance if and only if the constructed OSCM instance (G, τ_1, k) is a YES-instance. As OSCM is solvable in

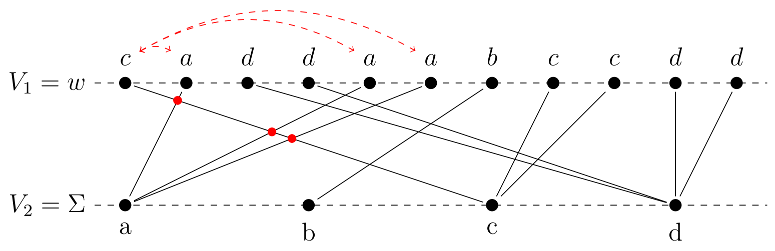


Figure 6.3: Reduction from GBS to OSCM: Given an alphabet $\Sigma = \{a, b, c, d\}$ and $w = caddaabccdd$, this bipartite graph is the OSCM instance obtained by the reduction. V_1 correspond to the set of position in w, V_2 correspond to Σ and there is an edge between the position i and a letter a if and only if $w[i] = a$. The number of crossings between edges adjacent to a and edges adjacent to c corresponds to the number of swaps needed between the letter a and c in w to put all the occurrences a before all the occurrences of c.

polynomial time if the vertices in V_2 have degrees at most two, this implies that GBS is solvable in polynomial time if each letter has at most two appearances, see Remark 30 and the following comments.

Together with the reduction proving Theorem 29, we see that GBS can be viewed as exactly the special case of OSCM where all vertices of V_1 are of degree one, so that the instance becomes a forest of stars with centers in V_2. We make the algorithmic consequences of this connection explicit, each time giving references to the literature on OSCM. Both for the parameterized complexity as for the approximation version (MINGBS), we exhibit two types of algorithms, as in each case, it could be that in practice the seemingly worse (but simpler) algorithm outperforms the more advanced and theoretically better algorithm.

Corollary 66. GBS *on strings of length n over the alphabet Σ, parameterized by the number k of swaps, can be solved (in polynomial space) in time $\mathcal{O}^\star(1.4656^k)$ by [Dujmovic et al., 2008] or (in exponential space) in time $\mathcal{O}^\star(2^{\sqrt{2k}})$ by [Kobayashi and Tamaki, 2015] or Remark 59.*

Fernau *et al.* [Fernau et al., 2014] and Kobayashi and Tamaki [Kobayashi and Tamaki, 2015] also obtained OSCM lower bound results, based on [Muñoz et al., 2001], assuming ETH. By the proof of Theorem 29, we can strengthen them in the following way:

Corollary 67. GBS *on strings of length n over the alphabet Σ, parameterized by the number k of swaps, can be solved neither in time $\mathcal{O}^\star(2^{o(n)})$ nor in time $\mathcal{O}^\star(2^{o(|\Sigma|)})$ nor in time $\mathcal{O}^\star(2^{o(\sqrt{k})})$, unless ETH fails, even if each letter has exactly 4 occurrences.*

Proof. As analyzed by Kobayashi and Tamaki [Kobayashi and Tamaki, 2015], the standard chain of reductions, starting from a 3-SAT instance and ending by the reduction in [Muñoz et al., 2001], produces a collection of 4-stars and can hence be interpreted as a GBS instance where each letter occurs four times. As shown in [Kobayashi and Tamaki, 2015] by a closer look at the construction from [Muñoz et al., 2001], the number of vertices and edges of the OSCM instance obtained from 3-SAT grows linearly in the number of variables and clauses. Hence, an algorithm solving GBS in time $\mathcal{O}^\star(2^{o(n)})$ or in time $\mathcal{O}^\star(2^{o(|\Sigma|)})$ would break ETH. Moreover, as $k \leq n^2$ by Lemma 26, also a GBS algorithm running in time $\mathcal{O}^\star(2^{o(\sqrt{k})})$ breaks ETH. □

Pathwidth in Grouping by Swapping. Our reduction from GBS to OSCM first turns $w \in \Sigma^*$ into a bipartite graph $G = (V_1, V_2, E)$ with $V_1 = [|w|]$ and $V_2 = \Sigma$. Lemma 57 then produces an equivalent PCIO-instance with an associated partial order ι_w on V_2 that is an interval order. For two letters $a, b \in \Sigma$, $(a, b) \in \iota_w$ means that the last occurrence of a in w comes before the first occurrence of b in w. The cocomparability graph G_{ι_w} is isomorphic to the graph G_w defined in Section 4.3. Obviously, $SCD(w)$ is the maximum size of an anti-chain in ι_w. Hence, the previously mentioned results of Habib and Möhring imply, together with Lemma 35:

Lemma 68. $SCD(w) = pw(G_{\rho_w}) + 1 = cpw(G_{\rho_w}, \rho_w) + 1.$

Lemma 38 has therefore the following algorithmic consequences for the string parameter *SCD*. To the best of our knowledge, this is the first algorithmic exploit of this string parameter.

Corollary 69. GBS *can be solved in time* $\mathcal{O}^\star(SCD(w)2^{SCD(w)})$.

As the scope coincidence degree of a word $w \in \Sigma^*$ is upper-bounded by $|\Sigma|$, we also obtain the following result for the parameter $|\Sigma|$ that improves on Lemma 27.

Corollary 70. GBS *can be solved in time* $\mathcal{O}^\star(|\Sigma|2^{|\Sigma|})$.

There is another graph-theoretic interpretation of the scope coincidence degree presented by Reidenbach and Schmid in [Reidenbach and Schmid, 2014] for patterns. It relates to our setting as follows. To a string $w \in \Sigma^n$, we associate its *Gaifman graph* Γ_w with vertex set $[n]$ and edges $(i, i+1)$ for $i \in [n-1]$, as well as the edge sets $E_a = \{(\min(\mathit{Scope}(a)), j) \mid j \in \mathit{Scope}(a)\}$ (disregarding loops) for each $a \in \Sigma$. According to [Reidenbach and Schmid, 2014, Lemma 15], $pw(\Gamma_w) \leq SCD(w) + 1$. It might be interesting to further link the pathwidths of Γ_w and of G_{ρ_w}. Do they differ by exactly two?

We could also try to link the notions developed for GbS in the context of OSCM. For instance, the Gaifman graph Γ of an OSCM instance (G, τ_1, k) with $G = (V_1, V_2, E)$ could be, assuming $V_1 = [n]$ and $<_{\tau_1} = <$, a graph with vertex set V_1 and edges $(i, i+1)$ for $i \in [n-1]$, as well as edge sets $E_x = \{(\min_< N(x), y) \mid (x, y) \in E\}$ for all $x \in V_2$. Possibly, $pw(G) = pw(\Gamma)$, at least up to a small constant. This could also mean that the pathwidth of G and of G_ρ, where ρ is the partial order associated to (G, τ_1), are the same up to a constant. This would mean that the pathwidth of G is upper-bounded by a term in $\mathcal{O}(\sqrt{k})$ if (G, τ_1, k) is a Yes-instance. This in itself would be an interesting observation.

Inspired by the considerations on the *range of a candidate* in KRA, the *maximum scope* $s_{\max} \doteq \max_{a \in \Sigma} |Scope(a)|$ could be another parameterization for GbS. Similar to Lemma 64, one can show that GbS, parameterized by $s_{\max}$, is in FPT. It would also be meaningful to interpret this parameter in the context of OSCM for graph visualization reasons, as long distances between neighbors of a vertex make graphs hard to read, because the edges of a graph should not be drawn with very long lines, and also the angles might become too small to be distinguished.

6.7 Discussions

Finally, we explain some further connections and future lines of research.

Different types of partial orders. It would be interesting to have a closer look to different types of partial orders in the context of PCO. For instance, the papers of Brandenburg and Gleißer [Brandenburg and Gleißner, 2016] or Hudry [Hudry, 2008] list quite a lot of different types of partial orders (in the context of rank aggregation problems). We can also view this research as a starting point to systematically look at decision problems related to partial orders from the viewpoint of parameterized complexity. Then, [Bouchitté and Habib, 1987] might be a good starting point.

Related problems, popular within Operations Research. In the Operations Research community, there has also been lots of studies of the *linear ordering polytope*. Regarding the problems studied in this chapter, [Buchheim et al., 2010] might be a good starting point. Likewise, the so-called Optimal Linear Extension Problem has been considered in the literature [Liu et al., 2011]. However, only the costs of the immediate neighborhood in the target linear order are considered, similar to the famous

Travelling Salesperson Problem,[4] while we sum up all costs associated to pairs (x, y) with $x < y$ in the final linear order $<$.

Putting additional constraints: a theme arising in Graph Drawing and in Order Theory. Forster [Forster, 2004] argues that the Constrained OSCM problem, where a partial order on V_2 is given in addition, that should be extended to a linear order (as before), has quite some applications. This can be clearly modeled as an instance of CO, but some further research is needed to conclude the same type of results as we did for OSCM with the interval order approach. This might relate to earlier (systematic) research on the realizability of constraints on interval orders, see [Pe'er and Shamir, 1997a, Pe'er and Shamir, 1997b]. In particular the distance constraints might be indeed interesting for graph drawing purposes, as the neighbor vertices should not stretch out too much.

Remarks on Approximation. For the minimization problem related to PCO, a PTAS is known according to [Fernau et al., 2014]. Our reasoning immediately implies the existence of PTAS for OSCM, KRA and GbS. In view of the tedious factor-1.4664 approximation for OSCM presented in [Nagamochi, 2005], this shows again the strength of looking at these specific problems from a wider perspective. We also refer to earlier PTAS publications such as [Karpinski and Schudy, 2010, Kenyon-Mathieu and Schudy, 2007, Kobayashi and Tamaki, 2015].

Comments on Approximation and Heuristics. We suggest that the tight connections that we found between GbS and OSCM should also be interesting in the development and analysis of (heuristic) algorithms for both problems. In this context, it is interesting to observe that Wong and Reingold [Wong and Reingold, 1991] proposed a median heuristic for computing a solution to a given GbS instance. They proved that on random instances, this heuristic is at most 10% off from the optimum (in expectation). Moreover, the larger random instances are picked, the smaller is the relative error of the median heuristic (in expectation). Incidentally, the same (median) heuristic was suggested by Eades and Wormalds [Eades and Wormald, 1994] some years later for OSCM. They proved that this heuristic is a factor-3 approximation, but did not go into a randomized analysis. Our translation of GbS into OSCM proves the following.

Corollary 71. *The median heuristic gives a factor-3 approximation for* GbS.

[4]The difference between cycles (tours) and paths do not matter for the involved algorithms that much.

Proof. Given an instance of GbS, we can translate it to an instance of OSCM, yielding a forest of stars. Now, the effect of the median heuristic for this OSCM instance is the same as the effect on the original GbS instance. We know that the obtained linear order is a factor-3 approximation for OSCM, which means that the median heuristic linear order of vertices of the OSCM instance creates not more than three times as many crossings as the optimum linear order. As this optimum linear order also corresponds to an optimum grouping of the original GbS instance, the claimed factor-3 approximation for GbS follows. □

The computational experiments undertaken by Martí and Laguna [Martí and Laguna, 2003] seem to indicate that many heuristics for OSCM work rather poorly on sparse graphs, but are quite good on dense graphs. From a practical point of view, sparse graphs are far more interesting, as one would not draw dense graphs with many crossing anyway for visualization purposes. Also, the OSCM-instances that one obtains from GbS-instances by our reductions are relatively sparse, as they have many vertices of degree one. It would be interesting to revisit experimental OSCM-settings for the forest instances obtained by our reductions.

Chapter 7

Diversity of Solutions for CO and KRA

In this chapter, we investigate the impact of the notions of diversity of solutions and fixed parameter tractability theory in the context of social choice and order theory. In particular, we focus on the Kemeny Partial Rank Aggregation (KPRA) problem.

When studying a given computational problem from the point of view of solution diversity, it is crucial to have in hand a notion of distance between solutions for that problem. The *diversity* of a set of solutions S is then defined as the sum of distances between pairs of solutions in S. We denote this measure by d. Intuitively, diversity is a global measure for how representative a set of solutions is among the space of solutions. Three natural parameters can be used to quantify how good a diverse set of solutions is: the number r of solutions in the set, the maximum distance δ between the cost of a solution in the set and the cost of an optimal solution (we call this parameter the *solution imperfection* of the set), and the minimum *required* distance s between any two solutions in the set. This last parameter is also known in the literature as the *scatteredness* of S [Galle, 1989]. Intuitively, the parameter r is expected to be small because in practical applications we do not want to overwhelm the user with an excessive number of choices. The parameter δ is expected to be small because while we do want S to be diverse, we do not want to allow solutions of bad quality. Finally, in the context of our work, solution diversity is formalized by the parameter d, and we only use the scatteredness parameter s to enforce that one cannot achieve high diversity by copying a given solution an arbitrary number of times. For this, it is enough to require $s = 1$. It is worth noting that it is possible to have s very small (say $s = 1$), but d very large, since some pairs of solutions in the set may be very far apart from one another.

The main result of this chapter is a multiparametric algorithm for Diverse KRA over partially ordered votes that runs in time $f(w, r, \delta, s) \cdot d \cdot n \cdot \log(n^2 \cdot m)$ where n is the number of candidates, m is the number of votes, r, δ, s and d are the parameters discussed above, and w is the *unanimity width* of the votes. That is to say, the pathwidth of the cocomparability graph of the unanimity order of the input votes (Corollary 84). Intuitively, this width measure is a quantification of the amount of disagreement between the votes.

As in Chapter 6, we use Completion of an Ordering as a framework to solve the diverse version of KPRA. For this, we define the diverse version of CO and develop an FPT algorithm to solve it. Even if we focus on KPRA, our algorithm can be directly applied to OSCM, and GbS using the reduction defined in Chapter 6.

7.1 A Notion of Diversity for KRA

The notion of diversity of solutions for computationally hard problems has been considered under a variety of frameworks. In this section, we define the Kemeny Partial Rank Aggregation problem which is a generalization of KRA to work on partial order for the votes instead of linear orders. Similar generalizations have been studied in the literature for votes with ties, also called bucket order [Hemaspaandra et al., 2005]. Then, we define a notion of diversity for the Kemeny Partial Rank Aggregation problem which is analogous to the notion of diversity of vertex sets used by [Baste et al., 2020]. We give a series of examples to illustrate the generalization to partial order and the use of diversity. To be able to use the Completion of an Ordering problem to solve Diverse-KPRA, we extend the reduction from KRA to PCO and discuss different reduction schemes depending the type of solutions we want.

7.1.1 Generalization of KRA

Let C be a finite set, which in this section will denote a set of candidates, or alternatives. A *partial vote*[1] on C is a partial order on C. The KT-distance between two partial votes π_1 and π_2, denoted by KT-dist(π_1, π_2), is the number of pairs of candidates that are

[1]The literature is not clear about these notions. In [Hemaspaandra et al., 2005], Hemaspaandra et al. call partial votes that allow ties in linear orders *preference rankings* and explain that this originally goes back to Kemeny. Allowing partial orders (as we do it here) is even more general. The authors of [Hemaspaandra et al., 2005] also consider a different Kendall-Tau distance for preference rankings compared to our setting.

ordered differently in the two partial votes.

$$\text{KT-dist}(\pi_1, \pi_2) = |\{(c, c') \in C \times C \mid c <_{\pi_1} c' \wedge c' <_{\pi_2} c\}|$$

Note that, if in a vote π_i, a pair of candidates (c_1, c_2) is unordered then this pair will never induce a cost in the KT-distance. In other words, π_i agree with any order of c_1 and c_2. Observe that the definition of the KT-distance is the same as in Section 5.3 but now extended for partial orders. Given a linear order π over a set of candidates C and a set Π of votes over C, we recall that the *Kemeny score* of π with respect to Π is defined as the sum of the Kendall-Tau distances between π and each vote in Π. In this chapter, we consider the following generalization of KRA.

Problem name: Kemeny Partial Rank Aggregation (KPRA)
Given: A list of partial votes Π over a set of candidates C, a non-negative integer k.
Output: Is there a linear order π on C such that the sum of the KT-distances to π from all the partial votes is at most k?

Hence, given partial votes $\pi_1, \ldots, \pi_m$ of C and a non-negative integer k, the question is if there exists a linear order $\pi \subseteq C \times C$ such that $\sum_{i=1}^{m} \text{KT-dist}(\pi, \pi_i) \leq k$.

Definition 72. *Given a set* Π *of partial votes, the* unanimity order *of* Π *is simply the partial order* ρ *obtained as the intersection of all partial orders in* Π*. In other words, a candidate* c_1 *has higher precedence than a candidate* c_2 *in* ρ *if and only if* c_1 *precedes* c_2 *in each vote in* Π.

As a consequence, the more disagreements there are among the voters with respect to the relative orders of pairs of candidates, the denser the cocomparability graph of ρ will be and therefore the greater its pathwidth will be. Therefore, the pathwidth of the cocomparability graph of the unanimity order of Π may be seen as a quantification of the amount of disagreement among the votes in Π.

7.1.2 Diverse Kemeny Partial Rank Aggregation

Next, we define a notion of diversity for the Kemeny Partial Rank Aggregation problem which is analogous to the notion of diversity of vertex sets used by [Baste et al., 2020]. More precisely, if R is a set of linear orders, then we define the Kendall-Tau diversity of R as the sum of Kendall-Tau distances between votes in the set R.

$$\text{KT-Div}(R) = \sum_{\pi_1,\pi_2 \in R} \text{KT-dist}(\pi_1, \pi_2)$$

We note that in the restricted case of the KRA problem where all votes are linear orders, the requirements that a set of solutions is at the same time diverse and only contains rankings with small Kemeny scores are clashing. The problem is that the very existence of two distinct rankings with small Kemeny scores is an impossible task. If two candidates c_1 and c_2 occur with the order (c_1, c_2) in one of the solutions and with the order (c_2, c_1) in the other solution, then at least one of these solutions will have a Kemeny score of at least half the number of votes. However, this opposition between diversity and small Kemeny score is not present in the setting where votes are allowed to be partial. The generalization to partial votes is one possible way to circumvent this conflict of desiderata. Another way we will be looking at is not to consider the cost of the solutions directly but the difference between the cost of solutions and the cost of an optimal solution. In this case, we can have diversity and a small difference between the cost and the cost of an optimal solution.

Problem name: DIVERSE KEMENY PARTIAL RANK AGGREGATION (DIVERSE-KRA)
Given: A list of partial votes Π over a set of candidates C, and k, r, $d \in \mathbb{N}$.
Output: Is there a set $R = \{\pi_1, \ldots, \pi_r\}$ of linear orders on C such that the Kemeny score for each order π_i is at most k and KT-Div$(R) \geq d$?

The mentioned NP-hardness of KRA immediately translates to NP-hardness results of KPRA and DIVERSE-KPRA, in the latter case by setting $r = 1$ and $d = 0$.

7.1.3 Some Discussion on K(P)RA

In this subsection, we briefly describe some toy applications where the notion of diversity can be naturally combined with the notion of Kemeny Rank Aggregation, both in the totally ordered setting and in the partially ordered setting.

A Natural Class of Partial Orders of Low Width. A *k-bucket order* [Fagin et al., 2004] is a partial order ρ where the set of vertices can be partitioned into a sequence of clusters $C_1, \ldots, C_m$, each of size at most k, where for each $i \in [m-1]$, all elements in

C_i precede all elements in C_{i+1}. In the context of democratic scheduling, an unanimity order that is a k-cluster order corresponds to the situation where the tasks to be executed are split into work-packages (the clusters), each containing at most k tasks, where the order of execution of the work-packages is agreed on, but the order inside each cluster is not. It is easy to see that the cocomparability graph of a k-cluster order has pathwidth at most k. Additionally, for each r and k, there are sets of votes whose unanimity order ρ is a k-cluster order such that there is a set S containing r linear extensions of ρ, each of which has optimal Kemeny score, and such that S has maximum diversity.

A Concrete Application of Bucket Orderings. Consider an election with 5 candidates A, B, C, D, E for 3 positions and 100 voters. If voters are forced to put (strict) linear orders, then it could be that there might be 50 votes like $A < B < C < D < E$ and 50 votes like $A < B < D < C < E$. There are two optimum Kemeny solutions, each of them coinciding with the two types of votes that were cast. But even if these two solutions are put into a diverse set of solutions, then the distance between these to votes is one. However, the sum of the Kemeny-Tau distances of any of the two optimal solutions to all given votes is 50. Hence, although the votes do agree to quite some extent, the resulting numbers are relatively big. But are these strict linear orders really expressing the opinions of the voters? This has been discussed in the social choice literature, and there is some evidence that many people do not have a strict preference among *all* candidates, but ranking them in groups is more realistic. This is our main motivation to introduce the KPRA model.

For instance, it could be that 20 voters do not care about the ranking of A versus B, but they would rank them all above C, D, E, without caring too much about their sequence, either. Another 10 voters might not care about the exact sequence of A and B, nor about the sequence of C and E, but they clearly put A and B before D which is in turn ahead of C and E. In shorter notation, we get $A = B < D < C = E$. There might be more different votes, as altogether summarized in the following table:

type I	20 voters	$A = B < C = D = E$
type II	10 voters	$A = B < D < C = E$
type III	10 voters	$A = B = C < D = E$
type IV	40 voters	$A = B = C = D < E$
type V	20 voters	$A < B < C = D = E$

This election could be turned into the first example if all voters would have been forced to commit themselves to linear orders. Obviously, the 50 type II and type IV voters are compatible with $A < B < D < C < E$, while all but the type II voters are compatible with $A < B < C < D < E$.

Hence, when viewing this as an instance of KPRA, the ranking $A < B < C < D < E$ would clearly win, as its Kemeny score is just 10. (Hemaspaandra et al. [Hemaspaandra et al., 2005] would attribute a much higher value here.) However, the diversity between $A < B < D < C < E$ and $A < B < C < D < E$ stays one. It might be also possible to consider the solution $B < A < C < D < E$ now, or even $A < B < C < E < D$. While in the model where we required all votes to be linear orders, only the mentioned two solutions would make sense in a diverse solution that should not be too expensive in terms of costs, here it might be possible to look for three or four solutions in the diverse set. This is another aspect that makes our model interesting for election problems.

Diversity in Search Rankings. Search engines are part of our everyday live. But who is really following the links presented by the search engine beyond the first few pages that are displayed on the user's screen? Therefore, it is crucial that important and interesting information is put on the very first pages. Usually, search engines consider some sort of relevance measure to rank the answers. Clearly, the search engine knows a bit more. For instance, is it important to display different hits from the same domain? Rather, it would be better for the user to see "really different" hits on the first page. Our concept of diversity could be implemented on two levels here: Either, we build a meta-search engine that collects the rankings of answers from different search engines (on the same question) and tries to come up with a diverse set of rankings that could help build the first couple of pages. Or, we consider different ranking functions within a search engine itself; in this second scenario, there could be also ties, so that our more general framework would apply.

Diversity in Team Formation. An organization wants to form a team/committee to perform some tasks and there are several candidates. But the committee will have only a few members (say three). To choose the committee, the organization will pick a rank with a sufficiently good score and select the first three candidates of that particular rank. The intuition is that candidates that appear in the first three positions of some rank with a good score have enough legitimacy to take the role. On the other hand, it may be important for the organization to have some liberty to choose which rank they will be using due to external factors, such as political, social, or affinity factors. For instance, if in one of the sufficiently good rankings the first three candidates are male, and in another sufficiently good ranking we have two females and one male, then it may be better to pick the latter one for gender equality reasons.

7.1.4 Reducing KPRA to CO

The reduction from KPRA to CO is similar to the one we gave in Section 6.5. We will first describe the reduction and then discuss the difference. Given an instance (Π, C) of KPRA with partial votes $\Pi = (\pi_1, \ldots, \pi_m)$ and candidates $C = \{c_1, \ldots, c_n\}$, we construct an instance $(\rho, \mathfrak{c})$ of CO by defining the cost function $\mathfrak{c} : C \times C \to \mathbb{N}$ as follows. For every pair of candidates (c, c'), we define its *cost*, $\mathfrak{c}(c, c')$, as the number of votes that order c' before c. More formally, $\mathfrak{c}(c, c') = |\{i \in [m] \mid c' <_{\pi_i} c\}|$. With this reduction, it is straightforward to check that a given linear order π of the candidates has Kemeny score

$$\begin{aligned}\sum_{i=1}^{m} \text{KT-dist}(\pi, \pi_i) &= \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{n} [c_j <_{\pi_i} c_k \wedge c_k <_{\pi} c_j] \\ &= \sum_{j=1}^{n} \sum_{k=1}^{n} \mathfrak{c}(c_k, c_j)[c_k <_{\pi} c_j].\end{aligned}$$

Here, for a logical proposition p, we use the bracket notation $[p]$ to denote the integer 1 if p is true or the integer 0 if p is false.

The main difference between this reduction and the reduction from KRA to PCO given in Section 6.5 is the definition of the partial order. In Section 6.5, as we are just looking for an optimal solution, the goal for the partial order ρ is to be as complete as possible such that ρ is provably a suborder of an optimal solution. Here, we are no longer looking for an optimal solution, but for a diverse set of good solutions. Therefore, ρ can be chosen with respect to what kind of diversity we want. Any partial order that is the suborder of an optimal solution can be chosen for ρ for our algorithm to work. We give two examples of partial orders and their implications with respect to the possible set of solutions.

As a first example we can use the same construction as in Section 6.5, given two candidates c_1 and c_2, $(c_1, c_2) \in \rho$ if and only if $\mathfrak{c}(c_1, c_2) = 0$. In the case of partial orders, if in every vote c_1 and c_2 are incomparable, then $\mathfrak{c}(c_1, c_2) = 0$ and $\mathfrak{c}(c_2, c_1) = 0$. In this case, we need to break the tie or we do not fix the order of c_1 and c_2 in ρ. Intuitively, this partial order corresponds to the following rule, the order between every two candidates must be the choice of a vote. In other words, the algorithm cannot output a ranking such that $c_1 < c_2$ if no one wants it and some vote want the opposite.

Another option is to use the unanimity order of the votes as the partial order ρ. In this case, if each vote contains (c_1, c_2) for some candidates c_1 and c_2, then $(c_1, c_2) \in \rho$. In other words, if everyone agrees then the algorithm cannot go against this choice. In the case of partial orders, if some votes do not order c_1 and c_2 and the rest choose $c_1 < c_2$, then $\mathfrak{c}(c_1, c_2) = 0$ but $(c_1, c_2) \notin \rho$. This order is a suborder of the previous one, which means that there are more possibilities for diverse solutions but the cocomparability may have a bigger pathwidth.

Note that both choices are equivalent in the case of linear orders for the votes.

If we choose a partial order that does not include the unanimity order, then the algorithm can output a ranking that disagrees with every vote on a pair of candidates. Therefore, in the following, we assume that the partial order contains at least the unanimity order.

We also note that since our reduction is solution preserving, it is also immediate that it is diversity preserving. In other words, R is a set of solutions of diversity d for an instance of KPRA if and only if it is also a set of solutions of diversity d for the corresponding instance of CO.

7.2 DP Algorithms for Diverse CO

To solve the generalization of KRA for partial orders and in the context of diversity, we will use the reduction from KPRA to the Completion of an Ordering problem. First, we define the Diverse Completion of an Ordering problem, Diverse-CO for short. Then, we give an FPT algorithm for Diverse-CO (Theorem 83). Then, we apply this algorithm to solve the Diverse Kemeny Partial Rank Aggregation problem (Corollary 84).

7.2.1 A Diversity Measure for CO.

We note that the notion of Kendall-Tau diversity introduced in Section 7.1 can also be used as a notion of diversity for CO, i.e., given a set R of (not necessarily optimal) solutions for a given instance $(\rho, \mathfrak{c})$ of CO, we let KT-Div(R) be the diversity of this set.

Problem name: DIVERSE COMPLETION OF AN ORDERING (DIVERSE-CO)
Given: A partial order $\rho \subseteq V \times V$ over a set V, a cost function $\mathfrak{c} : V \times V \to \mathbb{N}$, and non-negative integers $k, r, d \in \mathbb{N}$.
Output: Is there a set $R = \{\tau_1, \ldots, \tau_r\}$ of linear extensions of ρ such that $\mathfrak{c}(\tau_i \setminus \rho) \leq k$ for each $i \in [r]$, and KT-Div$(R) \geq d$?

In this chapter, we devise a fixed parameter tractable algorithm for DIVERSE CO parameterized by solution imperfection δ, number of solutions r, scatteredness s, and pathwidth w of the cocomparability graph of the input instance. Given our reduction that preserves solution and parameters from KPRA to CO introduced in Subsection 7.1.4, this algorithm immediately implies that DIVERSE KRA is fixed parameter tractable when parameterized by solution imperfection, number of solutions, scatteredness, and unanimity width.

7.2.2 Dynamic Programming Algorithm

Next, we develop an algorithm to solve DIVERSE-CO. To ease the description and proof of the algorithm, we split the description into two parts. First, we give an algorithm to find one optimal solution (Theorem 78). Then, we extend this algorithm to solve DIVERSE-CO (Theorem 83). The algorithm for one solution is quite similar to the one described in Section 6.2. It is also based on a dynamic programming algorithm over a consistent path decomposition. But the running time is worse. The dependency in the parameter is factorial instead of single exponential. The main reason is that we need to keep more information in each bag in order to extend it for DIVERSE-CO.

Let ρ be a partial order and $D = (B_1, B_2, \ldots, B_l)$ be a nice ρ-consistent path decomposition of G_ρ of width w. For each $p \in [l]$, let $\mathcal{P}_p$ be the set of pairs of the form (S, τ) where S is a subset of B_p that contains vertices introduced by B_p (intro$(p) \subseteq S \subseteq B_p$), $\tau \supseteq \rho|_S$ is a linear extension of the restriction $\rho|_S \doteq S \times S \cap \rho$ of ρ to S.

Definition 73. *Let $p \in [l]$, $\delta \in \mathbb{N}$, and $f : \mathcal{P}_p \to \mathbb{N}$. Then, we let $\mathcal{T}_p(f, \delta)$ be the set of all triples of the form (S, τ, γ), where $(S, \tau) \in \mathcal{P}_p$ and $f(S, \tau) \leq \gamma \leq f(S, \tau) + \delta$.*

Intuitively, the function f will be used by our dynamic programming algorithm to record the optimal values of partial solutions at each bag B_p when processing the path decomposition from left to right (see Theorem 78 and Theorem 83) and δ will be the allowed solution imperfection. In the case of a unique solution, this value will be 0 but this parameter will be useful in the diverse case as we allow sub-optimal linear extensions. The idea of the algorithm is to process each bag of the path decomposition one by one and for each bag, to incrementally construct a set of partial solutions. A partial solution up to the p-th bag is a linear order σ of $\bigcup_{j\leq p} B_j$. To extend σ, we need to insert vertices introduced by future bags in the current linear order. Note that newly introduced vertices have to be inserted in σ only after the already forgotten vertices in the p-th bag. If u will be introduced in a future bag and v is in some B_j with $j < p$ but not in B_p, then by the consistency of the path decomposition with respect to ρ, we have $v <_\rho u$. Therefore, in B_p, for each partial solution σ, we only need to remember the "last" part of σ, which are the vertices that are in B_p and after all forgotten vertices in σ. This means that for each bag B_p, we will construct and store a subset of $\mathcal{T}_p(f, \delta)$.

Remark 74. For each $p \in [l]$, $f : \mathcal{P}_p \to \mathbb{N}$ and $\delta \in \mathbb{N}$, the size of $\mathcal{T}_p(f, \delta)$ is bounded by $e \cdot (\delta + 1) \cdot (w + 1)!$.

Proof. Given a bag B_p at position p, the size of $\mathcal{T}_p(f, \delta)$ is bounded by:

$$(\delta+1)\cdot\sum_{0\leq i\leq |B_p|}\binom{|B_p|}{i}\cdot i! \leq e\cdot(\delta+1)\cdot|B_p|! \leq e\cdot(\delta+1)\cdot(w+1)!$$

where $\binom{|B_p|}{i}$ is the number of subsets of B_p of size i and $i!$ is the number of possible linear order on a set of size i. □

For each $p \in [l-1]$, $f : \mathcal{P}_p \to \mathbb{N}$ and $\delta \in \mathbb{N}$, we say that a triple $(S, \tau, \gamma) \in \mathcal{T}_p(f, \delta)$ is *compatible* with a triple $(S', \tau', \gamma') \in \mathcal{T}_{p+1}(f, \delta)$ if the following conditions are satisfied.

C1 If B_{p+1} forgets a vertex v then we have $S' = \{u \in S \mid v <_\tau u\}$. This means that S' is obtained from S by removing v and every vertex that is smaller than v in τ. Otherwise, B_{p+1} introduces a vertex v and we have $S' = S \cup \{v\}$, which means we add the new vertex v to S.

C2 $\tau|_{S\cap S'} = \tau'|_{S\cap S'}$, i.e., τ and τ' agree on $S \cap S'$.

C3 If B_{p+1} forgets v then we have $\gamma' = \gamma$. Otherwise, B_{p+1} introduces v and we have $\gamma' = \gamma + \sum_{u\in S, u<_{\tau'} v} \mathfrak{c}(u, v) + \sum_{u\in S, v<_{\tau'} u} \mathfrak{c}(v, u) + \sum_{u\in B_{p+1}\setminus S'} \mathfrak{c}(u, v)$. To compute γ', we add to γ the cost of adding v. The first two terms compute the cost of v in τ' and the last one computes the cost of placing v after the vertices in $B_p \setminus S$.

A *compatible sequence* for D is a sequence of triples $\mu = (S_1, \tau_1, \gamma_1) \cdots (S_l, \tau_l, \gamma_l)$ such that $S_1 = B_1$, $\gamma_1 = \mathfrak{c}(\tau_1)$ and for each $p \in [l]$, (S_p, τ_p, γ_p) is compatible with $(S_{p-1}, \tau_{p-1}, \gamma_{p-1})$.

Our interest in compatible sequences stems from the two following lemmas.

Lemma 75. *Let $\rho \subseteq V \times V$ be a partial order on V, $\mathfrak{c} : V \times V \to \mathbb{N}$ be a cost function, and D be a nice ρ-consistent path decomposition of the graph G_ρ. Let*

$$\mu = \mathfrak{t}_1 \dots \mathfrak{t}_l = (S_1, \tau_1, \gamma_1) \dots (S_l, \tau_l, \gamma_l)$$

be a compatible sequence for D. Then, the linear order $\pi = \mathbf{tc}(\rho \cup \tau_1 \cup \dots \cup \tau_l)$ is a linear extension of ρ of cost γ_l.

Proof. Lemma 75 follows straightforwardly by the following claim, which can be proved by induction on p.

▷ Claim 76. For each position $p \in [l]$, $\pi_p = \mathbf{tc}(\rho|_{L_p \cup B_p} \cup \tau_1 \cup \dots \cup \tau_p)$ is a linear extension of $\rho|_{L_p \cup B_p}$ of cost γ_p.

In the base case, $S_1 = B_1$ and by definition $\tau_1 = \pi_1$ is a linear extension of $\rho|_{B_1}$ and $\gamma_1 = \mathfrak{c}(\tau_1)$.

Now, let $p \in [l-1]$ be a position in D, and suppose that the claim holds for p. We show that it also holds for $p+1$.

If B_{p+1} is a forget bag, then by Item C1 and Item C2, we have that $\tau_{p+1} = \tau_p|_{S_{p+1}}$ and $\gamma_p = \gamma_{p+1}$. Therefore, the result follows directly by the induction hypothesis.

If B_{p+1} introduces a vertex v_{p+1}. We need to check that the transitive closure of $\rho|_{L_p \cup B_p} \cup \tau_1 \cup \dots \cup \tau_{p+1}$ defines a linear extension of $\rho|_{L_{p+1} \cup B_{p+1}}$. This means that π_{p+1} does not contain loops and each pair $u, v \in L_{p+1} \cup B_{p+1}$ is ordered by π_{p+1}. By Item C2, we have that $\mathbf{tc}(\rho \cup \tau_1 \cup \dots \cup \tau_p)|_{S_p \cap S_{p+1}} = \tau_p|_{S_p \cap S_{p+1}} = \tau_{p+1}|_{S_p \cap S_{p+1}}$. Therefore $\mathbf{tc}(\rho \cup \tau_1 \cup \dots \cup \tau_p)$ is compatible with τ_{p+1} and so there is no loop in π_{p+1}. Let $u, v \in L_{p+1} \cup B_{p+1}$. If $u, v \in L_p \cup B_p$, then, by the induction hypothesis, u and v are ordered by π_p and thus by π_{p+1}. Otherwise, let v be the vertex introduced by B_{p+1}. If $u, v \in S_{p+1}$, then u and v are ordered by τ_{p+1} and thus by π_{p+1}. If $u \in L_{p+1}$, by ρ-consistency of D, we have $u <_\rho v$ and thus $u <_{\pi_{p+1}} v$. If $u \in B_{p+1} \setminus S_{p+1}$, then, by Item C1, we have $u \leq_{\pi_p} \max_{\pi_p}(L_{p+1})$ and by ρ-consistency of D, we have $v >_\rho \max_{\pi_p}(L_{p+1})$. Therefore, by transitivity we have $v <_{\pi_{p+1}} u$. By Item C3, we have $\gamma_{p+1} = \mathfrak{c}(\pi_{p+1})$. □

Lemma 77. *Let $\rho \subseteq V \times V$ be a partial order on V, $\mathfrak{c} : V \times V \to \mathbb{N}$ be a cost function, and D be a nice ρ-consistent path decomposition of the graph G_ρ. Let π be a linear extension of ρ, and $\mu = (S_1, \tau_1, \gamma_1) \dots (S_l, \tau_l, \gamma_l)$ be a sequence such that for each position $p \in [l]$, $S_p = \{v \in B_p \mid v >_\pi \max_\pi(L_p)\}$, $\tau_p = \pi|_{S_p}$, and $\gamma_p = \mathfrak{c}(\pi|_{L_p \cup B_p})$. Then, μ is a compatible sequence for D.*

Proof. One can see that this construction satisfies conditions C1, C2 and C3 for each position in the path decomposition. □

Lemma 75 and Lemma 77 immediately yield an FPT dynamic programming algorithm for computing a linear extension of ρ. To define the algorithm more precisely, we first need to define the set of functions f_p that we will use to define a set of triples with $\mathcal{T}_p$. For each $p \in [l]$, we define $f_p : \mathcal{P}_p \to \mathbb{N}$ as follows. For each $(S, \tau) \in \mathcal{P}_p$, we let γ be the minimum cost of a partial solution π up to bag B_p such that $S = \{v \in B_p \mid v >_\pi \max_\pi(L_p)\}$ and $\tau = \pi|_{S_p}$; then we let $f_p(S, \tau) = \gamma$. Intuitively, f_p associates to each linear order τ the cost of an optimal partial solution "ending" by τ. Now, we will describe the algorithm. We process the path decomposition from left to right in l time steps, where at each time step p, we construct the value of f_p that we need and a subset $\mathcal{Q}_p \subseteq \mathcal{T}_p(f_p, 0)$ of *promising triples*, which are, intuitively, triples that have a potential to lead to an optimal solution. At time step 1, $B_1 = \{v\}$, we let $\mathcal{Q}_1 = \{(B_1, \emptyset, 0)\}$. At each time step $p \geq 2$, $\mathcal{Q}_p$ is the set of all triples in $\mathcal{T}_p(f_p, 0)$ that are compatible with some triple in $\mathcal{Q}_{p-1}$. At the end of the process, assuming that $\mathcal{Q}_p$ is non-empty for each $p \in [l]$, we can reconstruct a compatible sequence by backtracking. First, by selecting an arbitrary triple $\mathfrak{t}_l$ in $\mathcal{Q}_l$, then by selecting an arbitrary triple $\mathfrak{t}_{l-1}$ in $\mathcal{Q}_{l-1}$ compatible with $\mathfrak{t}_l$, and so on. Once we have constructed a compatible sequence $\mathfrak{t}_1 \dots \mathfrak{t}_l$, we can extract a linear extension π of cost γ_l by setting $\pi = \mathbf{tc}(\rho \cup \tau_1 \cup \dots \tau_l)$. This description gives rise to the following theorem.

Theorem 78. *Let $\rho \subseteq V \times V$, let w be the pathwidth of the cocomparability graph of ρ, and $\mathfrak{c} : V \times V \to [m]_0$ be a cost function. Then, one can compute an optimal solution in time $\mathcal{O}\left(w!^{\mathcal{O}(1)} \cdot |V| \cdot \log(|V| \cdot m)\right)$.*

Proof. By Lemma 36, one can construct a nice ρ-consistent path decomposition D of G_ρ of width $\mathcal{O}(w)$ in time $2^{\mathcal{O}(w)} \cdot |V|$.

Lemma 77 shows that, if a solution π exists, then there exists a compatible sequence associated with it. Now we will show that this sequence is actually built by the algorithm. Let μ be the sequence define in Lemma 77. We recall that the sequence is defined as follow $\mu = (S_1, \tau_1, \gamma_1) \dots (S_l, \tau_l, \gamma_l)$, where for each position $p \in [l]$, $S_p = \{v \in B_p \mid v >_\pi \max_\pi(L_p)\}$, $\tau_p = \pi|_{S_p}$ and $\gamma_p = \mathfrak{c}(\pi|_{L_p \cup B_p})$. First, because π is optimal, one can easily check that for each $p \in [l]$, $(S_p, \tau_p, \gamma_p) \in \mathcal{T}_p(f_p, 0)$. We will prove that this sequence is built by the algorithm by recurrence on the bags. By definition, $S_1 = B_1$ and $\gamma_1 = \mathfrak{c}(\tau_1)$, therefore the first triple is built by the algorithm. As $(S_{p+1}, \tau_{p+1}, \gamma_{p+1})$ is compatible with (S_p, τ_p, γ_p), then, by definition of compatibility and how the algorithm proceeds, if the algorithm builds (S_p, τ_p, γ_p) it will build $(S_{p+1}, \tau_{p+1}, \gamma_{p+1})$ in the next step. This proves the correctness of the algorithm.

Now we will prove the running time. In a nice path decomposition, there are $2|V|$ bags. For each position $p \in [2|V|]$, if B_p forgets a vertex v, then $\mathcal{Q}_p$ can be computed by removing v in each triple in $\mathcal{Q}_{p-1}$ and keeping triples with minimum cost for each fixed pair (S, τ). This can be done in time $\mathcal{O}(|\mathcal{Q}_{p-1}|^2)$. If B_p introduces a vertex v, then $\mathcal{Q}_p$ can be computed from $\mathcal{Q}_{p-1}$ by taking each triple $(S, \tau, \mathfrak{c})$ and adding v in τ at every possible position. Computing the new cost can be done in time $\mathcal{O}(w \cdot \log(n^2 \cdot m))$ and there are $|\mathcal{Q}_{p+1}|$ triples to compute. The log factor $\log(n^2 \cdot m)$ is the time of performing an addition on the costs, as the cost of a solution can be at most $n^2 \cdot m$. By Remark 74, we can bound the size of each $\mathcal{Q}_p$. □

Now, leveraging on Lemma 75 and Lemma 77, we will devise a fixed-parameter tractable algorithm for Diverse-CO parameterized by solution imperfection, number of solutions, scatteredness, and pathwidth of the cocomparability graph of the input partial order. Let ρ be a partial order and $D = (B_1, B_2, \ldots, B_l)$ be a nice ρ-consistent path decomposition of G_ρ of width w.

Definition 79. *Let $p \in [l]$, $r \in \mathbb{N}_{>0}$, $d \in \mathbb{N}$, $s \in \mathbb{N}$ and $f : \mathcal{P}_p \to \mathbb{N}$. Then, we let $\mathcal{I}_p^{r,d,s}(f, \delta)$ be the set of all tuples of the form*

$$((S^1, \tau^1, \gamma^1), \ldots, (S^r, \tau^r, \gamma^r), \partial, (\xi_{\{i,j\}})_{1 \le i < j \le r})$$

where $\partial \in [d]_0$, for each $1 \le i < j \le r$, $\xi_{\{i,j\}} \in [s]_0$, and for each $i \in [r]$, (S^i, τ^i, γ^i) is a triple in $\mathcal{T}_p(f, \delta)$.

Intuitively, $((S^i, \tau^i, \gamma^i))_{i \in [r]}$ are r partial linear extensions, ∂ will be the diversity of the r partial linear extensions and ξ will be the distance between all pairs of the r partial linear extensions.

Remark 80. For each $p \in [l]$, $r \in \mathbb{N}_{>0}$, $d \in \mathbb{N}$, $s \in \mathbb{N}$, $f : \mathcal{P}_p \to \mathbb{N}$ and $\delta \in \mathbb{N}$, the size of $\mathcal{I}_p^{r,d,s}(f, \delta)$ is bounded by $(e \cdot (\delta + 1) \cdot (w + 1)!)^r \cdot s^{r^2} \cdot d$.

For each $p \in [l - 1]$, each tuple

$$\mathfrak{u}_p = ((S_p^1, \tau_p^1, \gamma_p^1), \ldots, (S_p^r, \tau_p^r, \gamma_p^r), \partial_p, (\xi^p_{\{i,j\}})_{1 \le i < j \le r})$$

in $\mathcal{I}_p^{r,d,s}$ and each tuple

$$\mathfrak{u}_{p+1} = ((S_{p+1}^1, \tau_{p+1}^1, \gamma_{p+1}^1), \ldots, (S_{p+1}^r, \tau_{p+1}^r, \gamma_{p+1}^r), \partial_{p+1}, (\xi^{p+1}_{\{i,j\}})_{1 \le i < j \le r})$$

in $\mathcal{I}_{p+1}^{r,d,s}$, we define the *scatteredness increase table* of the pair $(\mathfrak{u}_p, \mathfrak{u}_{p+1})$, denoted by $\Delta\xi(\mathfrak{u}_p, \mathfrak{u}_{p+1})$, as the table holding the distance increase between each pair of partial linear extensions due to the potential newly introduced element in B_{p+1}. If B_{p+1} is a forget bag, then the scatteredness does not increase. Otherwise, if B_{p+1} introduces a vertex v, for each pair $\{i,j\}$ such that $1 \leq i < j \leq r$, we have

$$\begin{aligned}\Delta\xi_{\{i,j\}}(\mathfrak{u}_p, \mathfrak{u}_{p+1}) = {} & |\{u \in B_p \mid (u \notin S^i_{p+1} \vee u <_{\tau^i_{p+1}} v) \wedge v <_{\tau^j_{p+1}} u\}| + \\ & |\{u \in B_p \mid (u \notin S^j_{p+1} \vee u <_{\tau^j_{p+1}} v) \wedge v <_{\tau^i_{p+1}} u\}|\end{aligned} \tag{7.1}$$

where $(u \notin S^i_{p+1} \vee u <_{\tau^i_{p+1}} v)$ means that u is smaller than v in the i-th tuple, $(v <_{\tau^j_{p+1}} u)$ means that u is bigger than v in the j-th tuple, $(u \notin S^j_{p+1} \vee u <_{\tau^j_{p+1}} v)$ means that u is smaller than v in the j-th tuple and $(v <_{\tau^i_{p+1}} u)$ means that u is bigger than v in the i-th tuple. Intuitively, this measures the increase of the distance between each pair of the r partial solutions from the bag B_p to the bag B_{p+1}.

We define, in a similar way, the *diversity increase* of the pair $(\mathfrak{u}_p, \mathfrak{u}_{p+1})$, denoted by $\Delta\partial(\mathfrak{u}_p, \mathfrak{u}_{p+1})$, as the amount of diversity due to a potential newly introduced element in B_{p+1}. As the diversity is the sum of all pairwise distances between the r partial solutions, we will use the scatteredness increase table $(\mathfrak{u}_p, \mathfrak{u}_{p+1})$ to compute the diversity increase. We define the increase between two bags as

$$\Delta\partial(\mathfrak{u}_p, \mathfrak{u}_{p+1}) = \sum_{1 \leq i < j \leq r} \Delta\xi_{\{i,j\}}(\mathfrak{u}_p, \mathfrak{u}_{p+1}). \tag{7.2}$$

Intuitively, this measures the increase of the diversity between the r partial solutions up to the bag B_p and the r partial solutions up to the bag B_{p+1}.

We say that $\mathfrak{u}_p$ is compatible with $\mathfrak{u}_{p+1}$ if for all pair $\{i,j\}$, $\xi^{p+1}_{\{i,j\}} = \min(\xi^p_{\{i,j\}} + \Delta\xi_{\{i,j\}}(\mathfrak{u}_p, \mathfrak{u}_{p+1}), s)$, $\partial_{p+1} = \min(\partial_p + \Delta\partial(\mathfrak{u}_p, \mathfrak{u}_{p+1}), d)$, and for each $i \in [r]$, the triple $(S^i_p, \tau^i_p, \gamma^i_p)$ is compatible with the triple $(S^i_{p+1}, \tau^i_{p+1}, \gamma^i_{p+1})$.

A *diversity-compatible sequence* is a sequence of the form

$$\{((S^1_p, \tau^1_p, \gamma^1_p), \ldots, (S^r_p, \tau^r_p, \gamma^r_p), \partial_p, (\xi^p_{\{i,j\}})_{1 \leq i < j \leq r})\}_{p \in [l]},$$

where for each $p \in [l-1]$, the tuples at positions p and $p+1$ are compatible and for each $i \in [r]$, $S^i_1 = B_1$, $\gamma_i = 0$, $\partial_1 = 0$ and $\xi^1_{\{i,j\}} = 0$.

The next lemma is an analogue of Lemma 75 in the context of solution diversity.

Lemma 81. *Let $\rho \subseteq V \times V$ be a partial order on V, $\mathfrak{c} : V \times V \to \mathbb{N}$ be a cost function, and D be a nice ρ-consistent path decomposition of the graph G_ρ. Let*

$$\hat{\mu} = \{((S^1_p, \tau^1_p, \gamma^1_p), \ldots, (S^r_p, \tau^r_p, \gamma^r_p, \partial_p), (\xi^p_{\{i,j\}})_{1 \leq i < j \leq r})\}_{p \in [l]}$$

be a diversity-compatible sequence for D. Then, the following properties can be verified.

1. *For each $i \in [r]$, the order $\pi_i = \mathbf{tc}(\rho \cup \tau_1^i \cup \cdots \cup \tau_l^i)$ is a linear extension of ρ of cost γ_l^i.*
2. *For each i, j with $1 \le i < j \le r$, $\xi_{\{i,j\}}^l = \min(\text{KT-dist}(\pi_i, \pi_j), s)$.*
3. *$\partial_l = \min(\text{KT-Div}(\{\pi_1, \ldots, \pi_r\}), d)$.*

We note that Item 1 of Lemma 81 follows from Lemma 75, Item 2 follows from Equation 7.2, and Item 3 from Equation 7.1.

The next lemma is an analogue of Lemma 82 in the context of solution diversity.

Lemma 82. *Let $\rho \subseteq V \times V$ be a partial order on V, $\mathfrak{c} : V \times V \to \mathbb{N}$ be a cost function, and D be a nice ρ-consistent path decomposition of the graph G_ρ. Let $\pi_1, \ldots, \pi_r$ be r linear extensions of ρ, and*

$$\hat{\mu} = \{((S_p^1, \tau_p^1, \gamma_p^1), \ldots, (S_p^r, \tau_p^r, \gamma_p^r), \partial_p, (\xi_{\{i,j\}}^p)_{1 \le i < j \le r})\}_{p \in [l]}$$

be a sequence satisfying the following conditions.

1. *For each position $p \in [l]$, and each $i \in [r]$, $S_p^i = \{v \in B_p \mid v >_{\pi_i} \max_{\pi_i}(L_p)\}$, $\tau_p^i = \pi_i|_{S_p}$, $\gamma_p^i = \mathfrak{c}(\pi_i|_{L_p \cup B_p})$.*
2. *For each i, j with $1 \le i < j \le r$, $\xi_{\{i,j\}}^p = \min(\text{KT-dist}(\pi_i|_{L_p}, \pi_j|_{L_p}), s)$.*
3. *$\partial_p = \min(\text{KT-Div}(\{\pi_1|_{L_p \cup B_p}, \ldots, \pi_r|_{L_p \cup B_p}\}), d)$.*

Then, $\hat{\mu}$ is a diversity-compatible sequence for D.

Intuitively, what those lemmas say is that in order to construct a set of r solutions for an instance $(\rho, \mathfrak{c}, r, \delta, d, s)$ of Diverse-CO, all one needs to do is to construct r compatible sequences in parallel, by processing the given path decomposition from left to right, while using an additional register to keep track of the overall diversity at each time step and all the pairwise distances. In the same way that Lemma 75 and Lemma 77 yield an FPT dynamic programming algorithm parameterized by pathwidth for computing a single solution of an instance of CO (Theorem 78), Lemma 81 and Lemma 82 yield an FPT dynamic programming algorithm to compute a diverse set of solutions, in case it exists, parameterized by the cost of a solution, number of solutions and pathwidth (Theorem 83).

Theorem 83. *Let $\rho \subseteq V \times V$, w be the pathwidth of the cocomparability graph of ρ, and $\mathfrak{c} : V \times V \to [m]_0$ be a cost function. Then, one can determine whether ρ admits r linear extensions $\pi_1, \dots, \pi_r$ at distance at most δ from the optimum, of diversity at least d, and scatteredness at least s in time $\mathcal{O}\left((w! \cdot \delta)^{\mathcal{O}(r)} \cdot s^{r^2} \cdot d \cdot |V| \cdot \log(|V|^2 \cdot m)\right)$.*

Proof. This theorem is similar to Theorem 78. Here we build r linear extensions in parallel and we incrementally compute the diversity between the r solutions. The main difference is the computation of the diversity. Using Equation 7.1 and Equation 7.2, one can compute the increase of the pairwise distances and diversity in time $\mathcal{O}(r^2 \cdot w \cdot \log(n^2 \cdot m))$ for each tuple. The log factor $\log(n^2 \cdot m)$ again comes from performing an addition of costs, as the cost of a solution can be at most $n^2 \cdot m$. □

7.2.3 Applications to KPRA

By combining Theorem 78 with our reduction from KPRA to CO, we have an FPT algorithm for KPRA, parameterized by solution imperfection, number of solutions, scatteredness, and unanimity width (Corollary 84).

Corollary 84. *Let Π be a list of m partial votes over a set of n candidates C. Let w be the unanimity width of Π. Given Π and non-negative integers δ, r, s and d, one can determine in time*

$$\mathcal{O}\left((w! \cdot \delta)^{\mathcal{O}(r)} \cdot s^{r^2} \cdot d \cdot n \cdot \log(n^2 \cdot m)\right)$$

whether there is a set $R = \{\pi_1, \dots, \pi_r\}$ of r linear orders on C such that the Kemeny score for each order π_i is at distance at most δ of the optimum, and we find that $\text{KT-Div}(R) \geq d$ *and the scatteredness is at least s.*

Corollary 84 is our most general result that combines all the parameters, but not all applications need all parameters. Therefore, we will now derive some special cases.

Corollary 85. *Let Π be a list of m partial votes over a set of n candidates C. Let w be the unanimity width of Π. Given Π and non-negative integers δ, r and d, one can determine in time*

$$\mathcal{O}\left((w! \cdot \delta)^{\mathcal{O}(r)} \cdot d \cdot n \cdot \log(n^2 \cdot m)\right)$$

whether there is a set $R = \{\pi_1, \dots, \pi_r\}$ of r linear orders on C such that the Kemeny score for each order π_i is at distance at most δ of the optimum, and we find that $\text{KT-Div}(R) \geq d$.

Namely, by our formulation of R as a set, we implicitly have the requirement $s \geq 1$.

With our algorithm, it is possible to check if there exist r different optimal solutions. We can do this by setting $\delta = 0$, $s = 1$ and $d = 0$ and we get the following corollary.

Corollary 86. *Let* Π *be a list of* m *partial votes over a set of* n *candidates* C*. Let* w *be the unanimity width of* Π*. Given* Π *and non-negative integers* r *and* d*, one can determine in time*

$$\mathcal{O}\left((w!)^{\mathcal{O}(r)} \cdot 2^{r^2} \cdot n \cdot \log(n^2 \cdot m)\right)$$

whether there is a set $R = \{\pi_1, \ldots, \pi_r\}$ *of* r *different optimal linear orders on* C*.*

To get some insights of the structure of the solution space, one can ask for a set of r solutions of cost at most δ from the minimum cost with maximum diversity. This is not strictly a consequence of Corollary 84 but the same algorithm can be used to answer this question. In our algorithm, the value ∂ is used to compute the diversity of a partial solution up to d, but if the diversity is bigger than d, we just remember d. Then, if we set $d = r \cdot n^2$, which is an upper bound of the maximum diversity of r solutions, at the end of the algorithm, we will have a set of possible solutions with their exact diversities. From this, we can select the one with the biggest diversity. Therefore, we have the following result.

Corollary 87. *Let* Π *be a list of* m *partial votes over a set of* n *candidates* C*. Let* w *be the unanimity width of* Π*. Given* Π *and non-negative integers* δ *and* r*, one can compute in time*

$$\mathcal{O}\left((w! \cdot \delta)^{\mathcal{O}(r)} \cdot r \cdot n^3 \cdot \log(n^2 \cdot m)\right)$$

a set $R = \{\pi_1, \ldots, \pi_r\}$ *of* r *linear orders on* C *such that the Kemeny score for each order* π_i *is at distance at most* δ *of the optimum and such that* KT-Div(R) *is maximal.*

7.3 Discussions

In this chapter, we have addressed the Kemeny Partial Rank Aggregation problem, one of the most central problems in the theory of social choice, from the perspective of diversity of solutions and parameterized complexity theory. We have devised a fixed parameter tractable algorithm for the diverse version of KPRA with partially ordered votes where parameters are the solution imperfection, the number of solutions, the scatteredness and the unanimity width of the set of votes. As a by-product of our work, we have introduced new parameterized algorithms for problems in order theory that are of independent interest. In particular, we developed parameterized algorithms for the diverse version of the Completion of an Ordering problem (CO). The reduction given in Section 6.4 is solution preserving, therefore, our algorithm for Diverse CO directly gives an FPT algorithm for the diverse version of the One-Sided Crossing Minimization problem. The OSCM problem is a key building block for several heuristics such as heuristics to solve the Two-Sided Crossing Minimization problem and the Sugiyama approach [Sugiyama et al., 1981] for hierarchical graph drawing. In this context, diversity of solutions for OSCM could be used to improve current heuristics or design new ones. We believe that our algorithm for finding diverse solutions for CO has a very positive impact on the study of these and related computational problems in neighboring fields.

Chapter 8

Order Reconfiguration under Width Constraints

In this chapter, we look at reconfiguration problems. In a reconfiguration problem, instead of finding a solution for a computational problem, we want to study the connectivity of the solution space of the problem. Given some operations to modify solutions of a computational problem and two solutions $\mathcal{S}_1$ and $\mathcal{S}_2$, we want to know if there exists a sequence of modifications that reconfigure $\mathcal{S}_1$ into $\mathcal{S}_2$ where each step transforms a feasible solution into another. Reconfiguration problems can model real world applications and also give insight on the structure of solution spaces. Many problems have been studied under the reconfiguration framework such as Vertex Coloring [Johnson et al., 2016], List Edge-Coloring [Ito et al., 2012], Vertex Cover [Mouawad et al., 2014], Independent Set [Ito et al., 2014]. In this chapter, we consider the reconfiguration of linear orders of vertices of a graph under width constraints. The question is to know if under such constraints it is possible to reconfigure a solution into another. Our main result (Theorem 88) states that if τ and τ' are linear orders of cutwidth at most w, then τ can be reconfigured into τ' in width at most $2w$. Additionally, reconfiguration in width at most $2w$ can be done using at most $\mathcal{O}(n^2)$ swaps. Finally, a reconfiguration sequence can be found in polynomial time. In the same way, these results hold for the vertex separation number instead of the cutwidth (Theorem 101).

Using our results on the reconfiguration of linear orders, we establish an interesting connection between two apparently unrelated computational problems, the reachability for two-letter string rewriting and graph isomorphism, a famous and well-studied problem in graph theory (Theorem 100). Even if this connection does not lead to any direct advance on the graph isomorphism problem, it opens the way to study graph isomorphism from the rewriting and reconfiguration point of view. To do so, we introduce a notion of graph decomposition based on cutwidth named *unit decompositions* and a rewriting system, $R(2w)$, of unit decompositions. We show that one can rewrite a unit decomposition into another if and only if the two graphs associated with them are isomorphic (Theorem 98). This result, together with the fact that unit decompositions of minimum cutwidth can be approximated in FPT time, implies that the graph isomorphism problem for graphs of cutwidth at most w is FPT-equivalent to the reachability problem for $R(2w)$ (Theorem 100).

8.1 Linear Order Reconfiguration

We start by recalling some notations on linear order that we use in this chapter. Let V be a finite set with n elements. Even if a linear order is a special case of a partial order, in this chapter we use the definition and notation of linear orders as bijections between $[n]$ and V instead of the set based notation we used for partial orders in the previous chapters. More formally, we will use the following definition. A linear order τ on V is a bijection $\tau : [n] \to V$. Intuitively, for each $j \in [n]$ and $v \in V$, $\tau(j) = v$ indicates that v is the j-th element of τ. If $S \subseteq [n]$, then we let $\tau(S) = \{\tau(j) \ : \ j \in S\}$ be the image of S under τ.

Next, we define two operations on linear orders, namely to induce linear orders and the composition of two linear orders, that are useful to build the intermediate linear orders of a reconfiguration sequence.

Induce Linear Order. Let $\tau : [n] \to V$ be a linear order on a set V. Let $S \subseteq V$ be a subset of V. We let $\tau^S : [|S|] \to S$ be the linear order induced by τ on S. More precisely, if we write the elements of S in increasing order according to τ, then for each $i \in [|S|]$, $\tau^S(i)$ is the i-th element in this sequence.

Composition of Linear Orders. Let $i \in \{0, \dots, n\}$, and $\tau, \tau' : [n] \to V$. We let $\tau \oplus_i \tau' : [n] \to V$ be the linear order that orders the vertices in the subset $\tau([i]) \subseteq V$ according to τ, followed by the vertices in the subset $V \setminus \tau([i])$, ordered according to τ'. More precisely, $\tau \oplus_i \tau'$ is defined as follows for each $j \in [n]$.

$$\tau \oplus_i \tau'(j) = \begin{cases} \tau(j) & \text{if } j \leq i, \\ \tau'^{V \setminus \tau([i])}(j - i) & \text{if } j > i. \end{cases} \tag{8.1}$$

We note that in particular, $\tau \oplus_0 \tau' = \tau'$ and $\tau \oplus_n \tau' = \tau$.

8.2 Bounded Cutwidth Order Reconfiguration (BCOR)

We are now ready to focus on the main problem of this chapter, namely the Bounded Cutwidth Order Reconfiguration problem. We recall the statement of the problem.

Problem name: Bounded Cutwidth Order Reconfiguration
Given: An n-vertex graph G, two linear orders of the vertex set of G $\tau, \tau' : [n] \to V(G)$, and a non-negative integer $w \in \mathbb{N}$.
Output: Is it true that τ can be reconfigured into τ' in cutwidth at most w?

Given an instance of the Bounded Cutwidth Order Reconfiguration problem (G, τ, τ', w), it should be clear that if w is smaller than the cutwidth of the graph G, then the answer for BCOR is trivially no since in this case neither τ nor τ' are in $\mathrm{CW}(G, w)$. Recall that $\mathrm{CW}(G, w)$ is the set of linear orders of $v(G)$ of cutwidth at most w. On the other hand, we will show in Theorem 88 below that the answer is always yes if w is at least twice the cutwidth of the thickest input linear order.

Theorem 88. *Let G be an n-vertex graph and $\tau, \tau' : [n] \to V(G)$ be linear orders of $V(G)$ of cutwidth at most w. Then, τ can be reconfigured into τ' in cutwidth at most* $\mathrm{cw}(G, \tau) + \mathrm{cw}(G, \tau') \leq 2w$.

To prove this theorem, we need the following three lemmas. First, we will show that taking induced linear order does not increase the cutwidth (Lemma 89). Then, we will show that some specific intermediate linear orders have bounded cutwidth (Lemma 90). Finally, we will show that there is a reconfiguration sequence of bounded cutwidth between each consecutive pair of intermediate linear orders (Lemma 91). First, we recall some notations and properties that will be used in the proof. Let G be an n-vertex graph with vertex set $V(G)$ and edge set $E(G)$. Given sets $S, S' \subseteq V(G)$, we let $E(G, S, S') = \{\{u, v\} \in E(G) \ : \ u \in S, \ v \in S'\}$ be the set of edges with one endpoint in S and the other endpoint in S'. As a special case, we define $E(G, S) = E(G, S, V(G) \backslash S)$. We recall two properties that we use without explicit mention.

- **Monotonicity property:** If $T \subseteq S$ and $T' \subseteq S'$, then $E(G, T, T') \subseteq E(G, S, S')$.
- **Linearity property:** If $\{S_1, S_2\}$ is a partition of S, then $\{E(G, S_1, S'), E(G, S_2, S')\}$ is a partition of $E(G, S, S')$.

We start by the monotonicity of the cutwidth by taking induced linear orders.

Lemma 89. *Let G be an n-vertex graph, $S \subseteq V(G)$ and $\tau : [n] \to V(G)$ be a linear order on $V(G)$. Then, τ^S is a linear order on $V(G[S])$. Additionally,* $\mathrm{cw}(G[S], \tau^S) \leq \mathrm{cw}(G, \tau)$.

Proof. As $S = V(G[S])$, τ^S is a linear order on $V(G[S])$. Let $p \in [|S|]$ and let $p' \in [n]$ be the unique number such that $\tau^S(p) = \tau(p')$. Then,

$$\begin{aligned}
\mathrm{cw}(G[S], \tau^S, p) &= |E(G[S], \tau^S([p-1]))| \\
&= |E(G[S], \tau^S([p-1]), \{\tau^S(r) \mid r \geq p\})| \\
&= |E(G, \tau^S([p-1]), \{\tau^S(r) \mid r \geq p\})| \\
&\leq |E(G, \tau([p'-1]))| \\
&= \mathrm{cw}(G, \tau, p') \\
&\leq \mathrm{cw}(G, \tau)\,,
\end{aligned}$$

as $\tau^S([p-1]) \subseteq \tau([p'-1])$ and $\{\tau^S(r) \mid r \geq p\} \subseteq \{\tau(r') \mid r' \geq p'\} = V(G) \setminus \tau([p'-1])$. □

To prove Theorem 88, we will give a reconfiguration sequence that uses the composition of linear orders. At a high level, we will go from one linear order τ to the other τ' by using the composition of the two linear orders at each position $\tau' \oplus_i \tau$. Therefore, we show that the composition of two linear orders has bounded cutwidth.

Lemma 90. *Let G be an n-vertex graph and $\tau, \tau' : [n] \to V(G)$ be linear orders of $V(G)$ with cutwidth of at most w. Then, for each $i \in [n]$, $\tau \oplus_i \tau'$ has cutwidth at most* $\mathrm{cw}(G, \tau) + \mathrm{cw}(G, \tau') \leq 2w$.

Proof. Let $i, p \in [n]$. By definition of the cutwidth, we have that

$$\begin{aligned}
\mathrm{cw}(G, \tau \oplus_i \tau', p) &= |E(G, \tau \oplus_i \tau'([p-1]))| \\
&= |E(G, \tau \oplus_i \tau'([p-1]), V(G) \setminus \tau \oplus_i \tau'([p-1]))|\,.
\end{aligned}$$

There are two cases to be analyzed. First, if $p \leq i$, then, by definition of $\tau \oplus_i \tau'$ we have $\tau \oplus_i \tau'([p-1]) = \tau([p-1])$. Therefore,

$$\mathrm{cw}(G, \tau \oplus_i \tau', p) = |E(G, \tau([p-1]), V(G) \setminus \tau([p-1]))| = \mathrm{cw}(G, \tau, p) \leq \mathrm{cw}(G, \tau)\,.$$

Secondly, if $p > i$, then we have

$$\begin{aligned}
\mathrm{cw}(G, \tau \oplus_i \tau', p) &= |E(G, \tau \oplus_i \tau'([p-1]), V(G) \setminus \tau \oplus_i \tau'([p-1]))| \\
&\overset{(a)}{=} |E(G, \tau([i]), V(G) \setminus \tau \oplus_i \tau'([p-1]))| \\
&\quad + |E(G, (\tau \oplus_i \tau'([p-1])) \setminus \tau([i]), V(G) \setminus \tau \oplus_i \tau'([p-1]))| \\
&\overset{(b)}{\leq} \mathrm{cw}(G, \tau, i+1) + \mathrm{cw}(G[V(G) \setminus \tau([i])], \tau'^{V(G)\setminus\tau([i])}, p-i) \\
&\leq \mathrm{cw}(G, \tau) + \mathrm{cw}(G, \tau').
\end{aligned}$$

For Equality (a), observe that $\tau([i]) \subseteq \tau \oplus_i \tau'([p-1])$, therefore, $(\tau \oplus_i \tau'([p-1])) \cap \tau([i]) = \tau([i])$ and $\{\tau([i]), (\tau \oplus_i \tau'([p-1])) \setminus \tau([i])\}$ is a partition of $\tau \oplus_i \tau'([p-1])$. To understand Inequality (b), we need two arguments. As $\tau([i]) \subseteq \tau \oplus_i \tau'([p-1])$,

$$E(G, \tau([i]), V(G) \setminus (\tau \oplus_i \tau'([p-1]))) \subseteq E(G, \tau([i]), V(G) \setminus \tau([i]))\,,$$

which shows that the cardinality of the first set is upper-bounded by $\mathrm{cw}(G, \tau, i+1)$. As the edges in $E(G, (\tau \oplus_i \tau'([p-1])) \setminus \tau([i]), V(G) \setminus (\tau \oplus_i \tau'([p-1])))$ only connect vertices with positions beyond i within $\tau \oplus_i \tau'$, after an index shift, we see that only the linear order τ' really matters, which explains the inequality

$$\begin{aligned}
&|E(G, (\tau \oplus_i \tau'([p-1])) \setminus \tau([i]), V(G) \setminus (\tau \oplus_i \tau'([p-1])))| \\
&\qquad\qquad \leq \mathrm{cw}(G[V(G) \setminus \tau([i])], \tau'^{V(G)\setminus\tau([i])}, p-i)\,.
\end{aligned}$$

For the last inequality, apply Lemma 89 to derive $\mathrm{cw}(G[V(G) \setminus \tau([i])], \tau'^{V(G)\setminus\tau([i])}) \leq \mathrm{cw}(G, \tau')$. As p is arbitrary, we have that $\mathrm{cw}(G, \tau \oplus_i \tau') \leq \mathrm{cw}(G, \tau) + \mathrm{cw}(G, \tau') \leq 2w$ follows for each $i \in [n]$. □

Finally, we show that we can reconfigure the composition of two linear orders at position i, $\tau' \oplus_i \tau$, to the composition at position $i+1$, $\tau' \oplus_{i+1} \tau$, in bounded cutwidth.

Lemma 91. *Let G be an n-vertex graph, $\tau, \tau' : [n] \to V(G)$ be linear orders on $V(G)$ of cutwidth at most w, and $i \in \{0, \ldots, n-1\}$ be an integer. Then, $\tau \oplus_i \tau'$ can be reconfigured into $\tau \oplus_{i+1} \tau'$ in cutwidth at most* $\mathrm{cw}(G, \tau) + \mathrm{cw}(G, \tau') \leq 2w$.

Proof. By Lemma 90, $\tau \oplus_i \tau'$ and $\tau \oplus_{i+1} \tau'$ have cutwidth at most $\mathrm{cw}(G, \tau) + \mathrm{cw}(G, \tau') \leq 2w$. Let $j \in [n]$ such that $\tau \oplus_i \tau'(j) = \tau(i+1)$, i.e., j is the position of $\tau(i+1)$ in $\tau \oplus_i \tau'$. As for each $p \in [i]$, $\tau \oplus_i \tau'(p) = \tau \oplus_{i+1} \tau'(p) = \tau(p)$, we have $j > i$. Let us consider the following sequence of swaps:

$$\tau \oplus_i \tau' = \tau_0 \xrightarrow{j-1} \tau_1 \xrightarrow{j-2} \cdots \xrightarrow{i+1} \tau_{j-i-1} = \tau \oplus_{i+1} \tau'.$$

If $j = i+1$, this sequence is empty and $\tau \oplus_i \tau' = \tau \oplus_{i+1} \tau'$. At each step of this sequence, we swap $\tau(i+1)$ with its left neighbor. This brings $\tau(i+1)$ from position j to position $i+1$. By doing this, we transform $\tau \oplus_i \tau'$ into $\tau \oplus_{i+1} \tau'$.

Consider Figure 8.1 for an illustration of the key part of the following proof by induction. Inductively, we show that each element τ_t in the sequence has cutwidth upper-bounded by $\mathrm{cw}(G, \tau) + \mathrm{cw}(G, \tau') \leq 2w$. By Lemma 90, $\mathrm{cw}(G, \tau_0) = \mathrm{cw}(G, \tau \oplus_i \tau') \leq \mathrm{cw}(G, \tau) + \mathrm{cw}(G, \tau') \leq 2w$, which proves the induction basis. Let $t \in [j-i-1]$ and $p \in [n]$ be two integers. As induction hypothesis, we have $\mathrm{cw}(G, \tau_{t-1}) \leq \mathrm{cw}(G, \tau) + \mathrm{cw}(G, \tau') \leq 2w$. If $p \leq j-t$ or $p > j-t+1$, then we have $\tau_{t-1}([p-1]) = \tau_t([p-1])$, so $\mathrm{cw}(G, \tau_t, p) = \mathrm{cw}(G, \tau_{t-1}, p) \leq \mathrm{cw}(G, \tau) + \mathrm{cw}(G, \tau') \leq 2w$ by induction hypothesis. Otherwise, $p = j-t+1 \in \{i, \ldots, j\}$ (Figure 8.1) and we have

$$\begin{aligned}
\mathrm{cw}(G, \tau_t, p) &= |E(G, \tau_t([p-1]))| \\
&= |E(G, \tau_t([p-1]), \{\tau_t(r) \mid r \geq p\})| \\
&= |E(G, \tau_t([i] \cup \{p-1\}), \{\tau_t(r) \mid r \geq p\})| \\
&\quad + |E(G, \{\tau_t(l) \mid i < l < p-1\}, \{\tau_t(r) \mid r \geq p\})|.
\end{aligned}$$

By definition of τ_t and p, we have $\tau_t(p-1) = \tau_t(j-t) = \tau(i+1)$. Therefore, we have $|E(G, \tau_t([i] \cup \{p-1\}), \{\tau_t(r) \mid r \geq p\})| = |E(G, \tau([i+1]), \{\tau_t(r) \mid r \geq j-t+1\})|$. As we are swapping $\tau(i+1)$ leftwards, $\{\tau_t(r) \mid r \geq j-t+1\} \subseteq \{\tau(r) \mid r \geq i+2\} = V(G) \setminus \tau([i+1])$. Again by definition of τ_t and p, the elements in $\{\tau_t(l) \mid i < l < p-1\}$ are ordered according to τ', which is also true for $\{\tau_t(r) \mid r \geq p\}$. More formally, $\{\tau_t(l) \mid i < l < p-1\} = \{\tau \oplus_i \tau'(l) \mid i+1 \leq l \leq p-2\} = \{\tau'^{V(G)\setminus\tau([i+1])}(l') \mid l' \leq p-2-i\}$ and $\{\tau_t(r) \mid r \geq p\} = \{\tau \oplus_i \tau'(r) \mid r \geq p\} = \{\tau'^{V(G)\setminus\tau([i+1])}(r') \mid r' \geq p-i-1\}$. Therefore,

$$\begin{aligned}
\mathrm{cw}(G, \tau_t, p) &\leq \mathrm{cw}(G, \tau, i+2) + \mathrm{cw}(G[V(G) \setminus \tau([i+1])], \tau'^{V(G)\setminus\tau([i+1])}, p-i-1) \\
&\leq \mathrm{cw}(G, \tau) + \mathrm{cw}(G[V(G) \setminus \tau([i+1])], \tau'^{V(G)\setminus\tau([i+1])}) \\
&\leq \mathrm{cw}(G, \tau) + \mathrm{cw}(G, \tau') \\
&\leq 2w.
\end{aligned}$$

To achieve the penultimate inequality, we again apply Lemma 89. □

We are now ready to prove Theorem 88.

Proof of Theorem 88. Consider the following sequence: $\tau = \tau' \oplus_0 \tau \to^* \tau' \oplus_1 \tau \to^* \cdots \to^* \tau' \oplus_n \tau = \tau'$. By Lemma 91, one can realize each step in cutwidth at most $\mathrm{cw}(G, \tau) + \mathrm{cw}(G, \tau') \leq 2w$, which then also upper-bounds the whole reconfiguration sequence. □

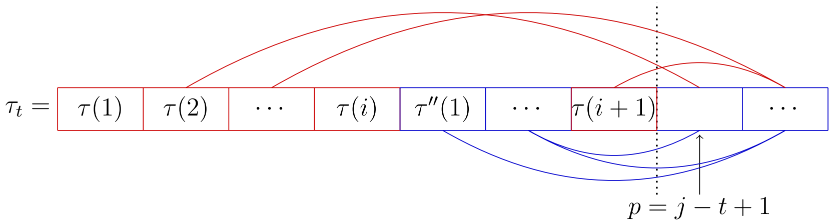


Figure 8.1: Illustration of the key part in Lemma 91. In this figure $\tau'' = \tau'^{V \setminus \tau([i+1])}$. The red part of the linear order follows the linear order τ for the first $i+1$ elements, and the blue part of the linear order follows τ' for the remaining elements. Then, the set of edges crossing the cut at position $p = j - t + 1$ can be split in two, the set of edges that start from the red part and the set of edges that start from the green part. The number of red edges is bounded by the cutwidth of τ and the number of green edges is bounded by the cutwidth of τ'.

We have proved, in Theorem 88, that one can always reconfigure a linear order τ, of cutwidth at most w, into a linear order τ', of cutwidth at most w, in cutwidth at most $2w$. Now, we show that for every $w \in \mathbb{N}_{>0}$, there exists a graph G and two linear orders τ and τ' of cutwidth w such that any reconfiguration sequence between τ and τ' needs linear orders of cutwidth at least $2w$. In other words, the bound given in Theorem 88 is sharp.

Proposition 92. *For each $w \in \mathbb{N}$, there exists a sequence of graphs G_n of size n and linear orders $\tau, \tau' : [n] \to V(G_n)$ such that $\mathrm{cw}(G_n, \tau) = \mathrm{cw}(G_n, \tau') = w$, but any reconfiguration sequence that transforms τ into τ' will have cutwidth of at least $2w$.*

Proof. For simplicity, we will start by giving a sequence of multigraphs with this property and we will see how to build simple graphs from them. Let $w, n \in \mathbb{N}$. Let $V(G_n) = [n]$ be the vertex set of a multigraph G_n with w edges between i and $i+1$ for each $i \in [n-1]$. Furthermore, let $\tau : [n] \to [n]$ be the identity and $\tau' : [n] \to [n]$ satisfy $\tau'(i) = n + 1 - i$ for $i \in [n]$. Then, $\mathrm{cw}(G_n, \tau) = \mathrm{cw}(G_n, \tau') = w$, but applying a swap to τ at any position will result in a linear order of cutwidth $2w$.

Now, to prove this result in the case of simple graphs, we will turn each graph G_n into a simple graph by replacing each edge in G with a path of length 2. In other words, we split each edge and add a new vertex in the middle. We call the vertices from G the main vertices and the added vertices the dummy vertices. To complete the construction, we have to extend τ and τ'. For that, we put the dummy vertices on the path from i to $i+1$ for each $i \in [n]$ after (respectively before) i and before (respectively after) $i+1$ in τ (respectively τ'). The order between the dummy vertices does not matter. Now it is easy to see that the cutwidth of τ and τ' is w. Any reconfiguration sequence that transforms τ into τ' needs to swap two main vertices at some point. Let i and $i+1$ be the two first main vertices to be swapped. If $i > 1$, then there exist w paths from i to $i+1$ and w paths from i to $i-1$. Those $2w$ paths are disjoint, therefore, the cut between i and $i+1$ has a size of at least $2w$. If $i = 1$ then $i+1 < n$ and the same reasoning works for $i+1$ instead of i. □

8.3 String Rewriting System and Graph Isomorphism

Now, we use our result on the Bounded Cutwidth Order Reconfiguration problem to make a connection between two apparently unrelated computational problems, namely, the Reachability problem for a two-letter rewriting system and the Graph Isomorphism problem. To do so, we introduce a notion of graph decomposition based on cutwidth named *unit decomposition*. In a unit decomposition, the graph is split into small pieces called *slices*. If a graph has bounded cutwidth, it can be decomposed into a unit decomposition of bounded width. Using the slices as an alphabet, we can see a graph as a word. Then, we introduce a rewriting system over this alphabet that preserve isomorphism. Our main result (Theorem 98) states that two graphs are isomorphic if and only if, their unit decompositions are reachable in this rewriting system.

8.3.1 Slice Rewriting System

We start by describing the notion of slices and unit decompositions. Intuitively, a slice is a piece of graphs. Given two compatible slices, we can combine them using an operation called the gluing operation to build a new bigger slice. After gluing enough slices, we end up with a graph. A unit decomposition is the representation of a graph as a sequence of compatible slices. Then, using slices as letters, we define an alphabet and a rewriting system over unit decompositions.

Slices. Let $\mathcal{I} = \{[a] \mid a \in \mathbb{N}\}$ denote the set of intervals of the form $[a] = \{1, \dots, a\}$ for $a \in \mathbb{N}$ (recall that $[0] = \emptyset$). We let $\mathcal{I}_0 = \{\{0\} \times [a] : [a] \in \mathcal{I}\}$, and $\mathcal{I}_1 = \{\{1\} \times [a] : [a] \in \mathcal{I}\}$. A *slice* $\mathbf{S} = (I, C, O, E)$ is a (multi-)graph where the vertex set $V = I \,\dot\cup\, C \,\dot\cup\, O$ is partitioned into an *in-frontier* $I \in \mathcal{I}_0$, a *center* $C \in \mathcal{I}$, and an *out-frontier* $O \in \mathcal{I}_1$, and E is a multiset of unordered pairs from $I \cup C \cup O$ in such a way that vertices of $I \cup O$ have degree exactly 1, there is no edge between any two vertices in I, and no edge between any two vertices in O. We depict slices as in Figure 8.2. We define slices using multigraphs, as the gluing operation, defined below, can take slices which are simple graphs, and create a slice that is a multigraph (see Figure 8.3). Given a slice $\mathbf{S}$, we

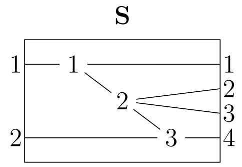


Figure 8.2: Slices are drawn as tiles. This figure depicts the slice $\mathbf{S} = (I, C, O, E)$ where $I = \{(0,1),(0,2)\}$, $C = \{1,2,3\}$, $O = \{(1,1),(1,2),(1,3),(1,4)\}$ and $E = \{\{(0,1),1\}, \{(0,2),3\},\{1,2\},\{2,3\},\{1,(1,1)\},\{2,(1,2)\},\{2,(1,3)\},\{3,(1,4)\}\}$. We omit the first element of the pair for frontier vertices and use the following convention. The in-frontier vertices are on the left of the tile and the out-frontier vertices are on the right of the tile. If the frontier vertices are not explicitly mentioned in the drawing, we assume that frontier vertices are ordered from top to bottom as in this drawing.

(a)

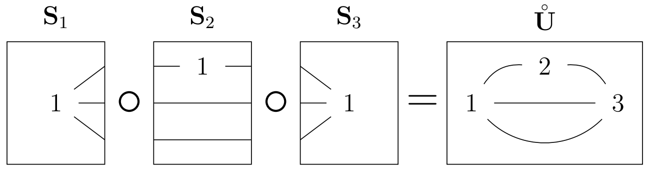


Figure 8.3: Slice associated with the unit decomposition $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2\mathbf{S}_3$. The resulting slice does not have any vertex in its frontier. It can therefore be seen as a multigraph on 3 vertices.

define $I(\mathbf{S})$ as the in-frontier of $\mathbf{S}$, $O(\mathbf{S})$ as the out-frontier of $\mathbf{S}$, and $C(\mathbf{S})$ as the center vertices of $\mathbf{S}$. The *width* of a slice $\mathbf{S}$ is defined as $\mathbf{w}(S) = \max(|I(\mathbf{S})|, |O(\mathbf{S})|)$.

Gluing Slices. A slice $\mathbf{S}_1 = (I_1, C_1, O_1, E_1)$ can be glued to $\mathbf{S}_2 = (I_2, C_2, O_2, E_2)$ if for some interval $[a] \in \mathcal{I}$, $O_1 = \{1\} \times [a]$ and $I_2 = \{0\} \times [a]$. In this case, the gluing gives rise to the slice $\mathbf{S}_1 \circ \mathbf{S}_2 = (I_1, C_1 \cup (|C_1| \oplus C_2), O_2, E)$ where $|C_1| \oplus C_2$ is a shift of the elements in C_2 by $|C_1|$, more formally $|C_1| \oplus C_2 = [|C_1| + |C_2|] \setminus [|C_1|] = \{|C_1|+1, |C_1|+2, \dots, |C_1|+|C_2|\}$,

$$\begin{aligned}
E =&\{\{x,y\} \in E_1 \mid x,y \in I_1 \cup C_1\} \\
&\cup \{\{x, y+|C_1|\} \mid \{x,y\} \in E_2 \wedge x \in O_2 \wedge y \in C_2\} \\
&\cup \{\{x+|C_1|, y+|C_1|\} \mid \{x,y\} \in E_2 \wedge x,y \in C_2\} \\
&\cup \{\{x,y\} \mid \exists i, \{x,(1,i)\} \in E_1 \wedge y \in O_2 \wedge \{(0,i),y\} \in E_2\} \\
&\cup \{\{x,y\} \mid \exists i, \{x,(1,i)\} \in E_1 \wedge y \in |C_1| \oplus C_2 \wedge \{(0,i), y-|C_1|\} \in E_2\}.
\end{aligned}$$

Note that the gluing operation is associative. Therefore, we will not write parentheses for the gluing of more than two slices. Figure 8.4 illustrates the gluing of two slices.

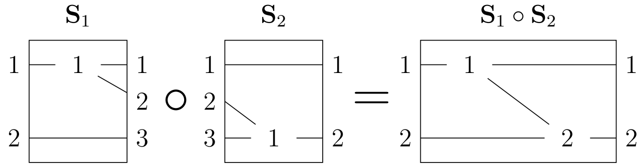


Figure 8.4: Gluing of two slices $\mathbf{S}_1$ and $\mathbf{S}_2$. The gluing operation is a way to merge two slices into one. In this example, the edge from the center vertex 1 from $\mathbf{S}_1$ to the out-frontier vertex $(1, 2)$ is stitched to the edge from the in-frontier vertex $(0, 2)$ to the center vertex 1 from $\mathbf{S}_2$ to form the edge between the center vertices 1 and 2 in $\mathbf{S}_1 \circ \mathbf{S}_2$. The stitching of edges is done following the order of the frontier vertices.

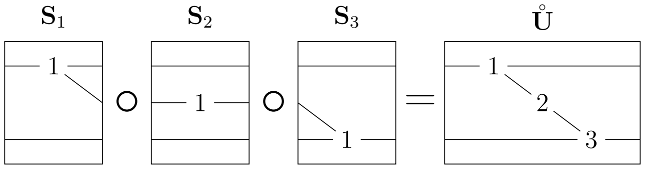


Figure 8.5: Slice associated with the unit decomposition $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2\mathbf{S}_3$. The gluing operation is associative, therefore parentheses are not needed.

Unit Slices and Unit Decompositions. We say that a slice is a *unit slice* if it has a unique vertex in its center. A *unit decomposition* is a sequence $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2 \dots \mathbf{S}_n$, where $\mathbf{S}_i$ are unit slices and $\mathbf{S}_i \circ \mathbf{S}_{i+1}$ is well defined for each $i \in [n-1]$. The *slice associated with a unit decomposition* $\mathbf{U}$ is defined as $\mathring{\mathbf{U}} = \mathbf{S}_1 \circ \mathbf{S}_2 \circ \dots \circ \mathbf{S}_n$ (see Figure 8.5 for an example). Note that if the in-frontier of $\mathbf{S}_1$ and the out-frontier of $\mathbf{S}_n$ are empty, then $\mathring{\mathbf{U}}$ is just a multigraph with vertex set $[n]$ (see Figure 8.3). For each $w \in \mathbb{N}$, we define the alphabet $\mathbf{\Sigma}(w)$ as the set of all unit slices of width at most w.

Proposition 93. *$\mathbf{\Sigma}(w)$ is finite, $|\mathbf{\Sigma}(w)| \in \mathcal{O}(w^4)$.*

Proof. In a unit slice of width at most w, the in-frontier and out-frontier can have between 0 and w vertices and there is only one center vertex. Therefore there are $(w+1)^2$ configurations for the vertices of a slice. By definition, vertices of the in-frontier and the out-frontier have degrees exactly 1, vertices in the in-frontier can be connected to the center vertex or an out-frontier vertex and there is no self-loop. Therefore, fixing the connectivity of the in-frontier vertices defines the full unit slice, because, if an out-frontier vertex is not connected to an in-frontier vertex, then it must be connected to the center vertex. Once the number of vertices in the frontier is chosen, there are at most $(w+1) \cdot (w+2)$ ways to connect the vertices in the in-frontier. □

We let $\mathbf{\Sigma}(w)^{\circledast}$ denote the set of all unit decompositions over $\mathbf{\Sigma}(w)$.

The order of the unit slices in a unit decomposition $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2 \dots \mathbf{S}_n$ induces a linear

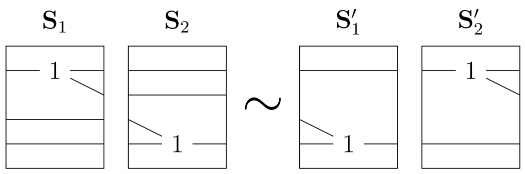


Figure 8.6: Local equivalence. $\mathbf{S}_1\mathbf{S}_2$ is (locally) $\mathcal{R}(4)$-equivalent to $\mathbf{S}'_1\mathbf{S}'_2$.

order $\tau_{\mathbf{U}}$ on the center vertices of the slice $\mathring{\mathbf{U}}$. We extend this linear order to all the vertices of $\mathring{\mathbf{U}}$ by putting the vertices in the in-frontier first, then the center vertices and, finally, the vertices in the out-frontier. More formally, the linear order defined by $\mathbf{U}$ sets $\tau_{\mathbf{U}}(i) = (0, i)$ for each $(0, i) \in I(\mathbf{S}_1)$, $\tau_{\mathbf{U}}(|I(\mathbf{S}_1)| + i) = i$ for each $i \in \{1, \ldots, n\}$ and $\tau_{\mathbf{U}}(|I(\mathbf{S}_1)| + n + i) = (1, i)$ for each $(1, i) \in O(\mathbf{S}_n)$.

Given a unit decomposition $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2 \ldots \mathbf{S}_n$ in $\mathbf{\Sigma}(w)^{\circledast}$, we let $\mathbf{w}(\mathbf{U}) = \max_{i\in[n]} \mathbf{w}(\mathbf{S}_i)$ be the *width* of $\mathbf{U}$. Recall that $\mathbf{w}(\mathbf{S}_i) = \max(|I(\mathbf{S}_i)|, |O(\mathbf{S}_i)|)$.

Equivalence of Slices. Let $\mathbf{S}_1 = (I_1, C_1, O_1, E_1)$ and $\mathbf{S}_2 = (I_2, C_2, O_2, E_2)$ be two slices. We say that $\mathbf{S}_1$ is *equivalent* to $\mathbf{S}_2$, denoted by $\mathbf{S}_1 \sim \mathbf{S}_2$, if and only if $I_1 = I_2$, $O_1 = O_2$, $C_1 = C_2$, and there is an isomorphism ϕ from $\mathbf{S}_1$ to $\mathbf{S}_2$ such that the restriction of ϕ to I_1 and O_1 is the identity function. In other words, $\mathbf{S}_1$ and $\mathbf{S}_2$ are equivalent if they are equal up to the renaming of the center vertices.

We let $\mathcal{R}(w) \subseteq \mathbf{\Sigma}(w)^2 \times \mathbf{\Sigma}(w)^2$ be the set of all rewriting rules of the form $\mathbf{S}_1\mathbf{S}_2 \to \mathbf{S}'_1\mathbf{S}'_2$ such that $\mathbf{S}_1 \circ \mathbf{S}_2 \sim \mathbf{S}'_1 \circ \mathbf{S}'_2$. By Proposition 93, $\mathcal{R}(w)$ is finite and $\mathcal{R}(w) \in \mathcal{O}(w^8)$. We call two unit decompositions $\mathbf{U}, \mathbf{U}' \in \mathbf{\Sigma}(w)^{\circledast}$ *locally* $\mathcal{R}(w)$*-equivalent*, and denote this fact by $\mathbf{U} \overset{w}{\sim} \mathbf{U}'$, if there exist $\mathbf{W}, \mathbf{W}' \in \mathbf{\Sigma}(w)^{\circledast}$ and $\mathbf{S}_1, \mathbf{S}'_1, \mathbf{S}_2, \mathbf{S}'_2 \in \mathbf{\Sigma}(w)$ with $\mathbf{S}_1 \circ \mathbf{S}_2 \sim \mathbf{S}'_1 \circ \mathbf{S}'_2$ such that $\mathbf{U} = \mathbf{W}\mathbf{S}_1\mathbf{S}_2\mathbf{W}'$ and $\mathbf{U}' = \mathbf{W}\mathbf{S}'_1\mathbf{S}'_2\mathbf{W}'$ (see Figure 8.6). In other words, $\mathbf{U}$ is locally $\mathcal{R}(w)$-equivalent to $\mathbf{U}'$ if $\mathbf{U}$ can be rewritten in one step into $\mathbf{U}'$ using a rule from $\mathcal{R}(w)$.

We let $\overset{w}{\equiv} \subseteq \mathbf{\Sigma}(w)^{\circledast} \times \mathbf{\Sigma}(w)^{\circledast}$ be the equivalence relation defined on unit decompositions by taking the reflexive, symmetric and transitive closure of $\overset{w}{\sim}$. If $\mathbf{U} \overset{w}{\equiv} \mathbf{U}'$, then we say that $\mathbf{U}'$ is $\mathcal{R}(w)$*-equivalent* to $\mathbf{U}$. We note that if $\mathbf{U}$ is a unit decomposition in $\mathbf{\Sigma}(w)^{\circledast}$ then any unit decomposition $\mathbf{U}'$ that is $\mathcal{R}(w)$-equivalent to $\mathbf{U}$ is also a unit decomposition in $\mathbf{\Sigma}(w)^{\circledast}$. We also note that there may exist unit decompositions in $\mathbf{\Sigma}(w)^{\circledast}$ that are not $\mathcal{R}(w)$-equivalent but that are $\mathcal{R}(w')$-equivalent for some $w' > w$.

Twisting. Let $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_n$ and $\mathbf{U}' = \mathbf{S}'_1\mathbf{S}'_2 \cdots \mathbf{S}'_n$ be two unit decompositions. We say that $\mathbf{U}$ is a *twisting* of $\mathbf{U}'$ if $\mathring{\mathbf{U}} = \mathring{\mathbf{U}}'$. Note that we are not equating slices up

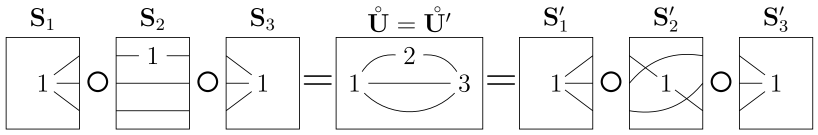


Figure 8.7: The unit decomposition $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2\mathbf{S}_3$ is a twisting of the unit decomposition $\mathbf{U}' = \mathbf{S}'_1\mathbf{S}'_2\mathbf{S}'_3$. Note that $\mathbf{S}_2 \circ \mathbf{S}_3 = \mathbf{S}'_2 \circ \mathbf{S}'_3$. Note that if we let π be the permutation that sets $\pi(1) = 2$, $\pi(2) = 3$ and $\pi(3) = 1$, then $\mathbf{S}'_2$ is obtained by permuting the out-frontier of $\mathbf{S}_2$ according to π and $\mathbf{S}'_3$ is obtained by permuting the in-frontier of $\mathbf{S}_3$ according to π.

to isomorphism. In other words, we are really requiring that the slices $\mathring{\mathbf{U}}$ and $\mathring{\mathbf{U}}'$ are syntactically identical.

Let $\mathbf{S}_1$ and $\mathbf{S}_2$ be unit slices in $\mathbf{\Sigma}(w)$ such that the out-frontier of $\mathbf{S}_1$ and the in-frontier of $\mathbf{S}_2$ have w' vertices for some $w' \leq w$. Given a permutation $\pi : [w'] \to [w']$, let $\mathbf{S}_1^\pi$ be the slice obtained by renaming each vertex $(1, i)$ in the out-frontier of $\mathbf{S}_1$ to $(1, \pi(i))$, and let ${}^\pi\mathbf{S}_2$ be the slice obtained by renaming each vertex $(0, i)$ in the in-fronter of $\mathbf{S}_2$ to $(0, \pi(i))$. Then, it should be clear that $\mathbf{S}_1 \circ \mathbf{S}_2 = \mathbf{S}_1^\pi \circ {}^\pi\mathbf{S}_2$. In other words, $\mathbf{S}_1^\pi {}^\pi\mathbf{S}_2$ is a twisting of $\mathbf{S}_1\mathbf{S}_2$. Additionally, for each two unit slices $\mathbf{S}'_1$ and $\mathbf{S}'_2$ such that $\mathbf{S}'_1\mathbf{S}'_2$ is a twisting of $\mathbf{S}_1\mathbf{S}_2$ ($\mathbf{S}_1 \circ \mathbf{S}_2 = \mathbf{S}'_1 \circ \mathbf{S}'_2$), it should be clear that there is some permutation π such that $\mathbf{S}'_1 = \mathbf{S}_1^\pi$ and $\mathbf{S}'_2 = {}^\pi\mathbf{S}_2$. Note also that for every two such slices $\mathbf{S}'_1$ and $\mathbf{S}'_2$, the rewriting rule $\mathbf{S}_1\mathbf{S}_2 \to \mathbf{S}'_1\mathbf{S}'_2$ belongs to $\mathcal{R}(w)$. This implies that if a unit decomposition $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2 \ldots \mathbf{S}_n$ is a twisting of a unit decomposition $\mathbf{U}' = \mathbf{S}'_1\mathbf{S}'_2 \ldots \mathbf{S}'_n$, then $\mathbf{U}$ and $\mathbf{U}'$ are $\mathcal{R}(w)$-equivalent and can be transformed into each other by applying a sequence of rewriting rules that "twists" for each $i \in [n-1]$ the out-frontier of $\mathbf{S}_i$ and the in-frontier of $\mathbf{S}_{i+1}$ according to some permutation π_i. This process is illustrated in Figure 8.7.

Proposition 94 (Twisting)**.** *Let* $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_n$ *and* $\mathbf{U}' = \mathbf{S}'_1\mathbf{S}'_2 \cdots \mathbf{S}'_n$ *be two unit decompositions in* $\mathbf{\Sigma}(w)^{\circledast}$ *such that* $\mathbf{U}$ *is a twisting of* $\mathbf{U}'$*. Then,* $\mathbf{U}$ *can be transformed into* $\mathbf{U}'$ *by the application of* $n-1$ *rewriting rules from* $\mathcal{R}(w)$*.*

8.3.2 Graph Isomorphism as a Rewriting System

Now, we are ready to see the connection between the Graph Isomorphism problem and the Reachability problem in $\mathcal{R}(w)$. First, we show that a graph G has cutwidth at most w if and only if it has a unit decompositions of width at most w (Proposition 95 and Proposition 96). Then, we show that the rewriting system $\mathcal{R}(w)$ preserves isomorphism (Lemma 97). Building on those results, we show the connection between the Reachability problem in $\mathcal{R}(2w)$ and the Graph Isomorphism for graphs of cutwidth at most w.

Intuitively, a unit decomposition $\mathbf{U}$ is a decomposition of the graph $\mathring{\mathbf{U}}$. This decomposition induces an ordering of the vertices of $\mathring{\mathbf{U}}$. The size of the common frontier between two neighbouring slices in $\mathbf{U}$ corresponds to the size of the cut at the same position in $\mathring{\mathbf{U}}$ with respect to $\tau_{\mathbf{U}}$. Therefore, $\mathbf{U}$ induces an ordering of $\mathring{\mathbf{U}}$ of cutwidth $\mathbf{w}(\mathbf{U})$. This idea is formalized by the following proposition.

Proposition 95. *Let $w \in \mathbb{N}$, and $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2 \dots \mathbf{S}_n$ be a unit decomposition in $\mathbf{\Sigma}(w)^{\circledast}$, and $\tau_{\mathbf{U}}$ be the linear order induced by $\mathbf{U}$ on $\mathring{\mathbf{U}}$. Then,* $\mathrm{cw}(\mathring{\mathbf{U}}, \tau_{\mathbf{U}}) = \mathbf{w}(\mathbf{U})$.

Proof. This follows by noticing that each vertex in $I(\mathbf{S}_1)$ has degree 1 and there is no edge between vertices in $I(\mathbf{S}_1)$, therefore for each position p in $\{1, \dots, |I(\mathbf{S}_1)| + 1\}$, $\mathrm{cw}(\mathring{\mathbf{U}}, \tau_{\mathbf{U}}, p) \leq |I(\mathbf{S}_1)|$ and $\mathrm{cw}(\mathring{\mathbf{U}}, \tau_{\mathbf{U}}, |I(\mathbf{S}_1)| + 1) = |I(\mathbf{S}_1)|$, in the same way, we have for each p in $\{|I(\mathbf{S}_1)| + n + 1, \dots, |I(\mathbf{S}_1)| + n + |O(\mathbf{S}_n)|\}$, $\mathrm{cw}(\mathring{\mathbf{U}}, \tau_{\mathbf{U}}, p) \leq |O(\mathbf{S}_n)|$ and $\mathrm{cw}(\mathring{\mathbf{U}}, \tau_{\mathbf{U}}, |I(\mathbf{S}_1)| + n + 1) = |O(\mathbf{S}_n)|$, and for each $p \in \{|I(\mathbf{S}_1)| + 2, \dots, |I(\mathbf{S}_1)| + n\}$, $\mathrm{cw}(\mathring{\mathbf{U}}, \tau_{\mathbf{U}}, p) = |O(\mathbf{S}_{p-|I(\mathbf{S}_1)|-1})| = |I(\mathbf{S}_{p-|I(\mathbf{S}_1)|})|$. □

The relation between cutwidth and unit decomposition is valid in both directions. The following proposition states the reverse direction compared to Proposition 95.

Proposition 96. *Let G be an n-vertex graph and τ be a linear order on the vertices of G of cutwidth w. Then, we can construct in time $\mathcal{O}(wn)$ a unit decomposition $\mathbf{U}$ such that $\tau = \tau_{\mathbf{U}}$.*

Proof. We will do this construction by first drawing the graph G in the plane. G does not need to be planar for this construction to work. First, we will place the vertices of G on a straight line L isomorphic to $\mathbb{R}$. The i-th vertex of G with respect to the linear order τ is placed at the coordinate i on the line. Then, edges are drawn as curves between their endpoints. Now, we will draw $n+1$ lines perpendicular to L at coordinates $\{-0.5, 0.5, 1.5, \dots, n-0.5, n+0.5\}$. We call these lines *cut-lines*. The cutwidth of τ is w, therefore each cut-line intersects at most w edges in the drawing of G. We put a vertex at the intersection of a cut-line and an edge. The graph between two consecutive cut-lines defines a unit slice of width at most w. Taking all those slices in the order induced by τ on the line L gives a unit decomposition $\mathbf{U}$ of width w such that $\tau = \tau_{\mathbf{U}}$. Figure 8.8 illustrates this construction. □

Now that we have defined the rewriting system, we are ready to show the connection between the rewriting system $\mathcal{R}(w)$ and the graph isomorphism problem. This connection is formalized in Theorem 98. The next lemma shows one direction in this connection.

Lemma 97. *Let $w \in \mathbb{N}$ and $\mathbf{U}$ and $\mathbf{U}'$ be unit decompositions in $\mathbf{\Sigma}(w)^{\circledast}$. If $\mathbf{U}$ is $\mathcal{R}(w)$-equivalent to $\mathbf{U}'$, then $\mathring{\mathbf{U}}$ is isomorphic to $\mathring{\mathbf{U}}'$.*

(b)

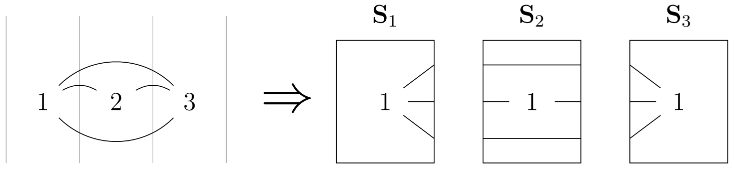


Figure 8.8: Slicing of the graph G on the left into a unit decomposition $\mathbf{U}$ on the right.

Proof. It is enough to show that if $\mathbf{U}$ can be transformed into $\mathbf{U}'$ in one $\mathcal{R}(w)$-rewriting step then $\mathring{\mathbf{U}}$ is isomorphic to $\mathring{\mathbf{U}}'$. Therefore, assume that $\mathbf{U} \to \mathbf{U}'$. Then, there exist unit decompositions $\mathbf{W}, \mathbf{W}' \in \mathbf{\Sigma}(w)^{\circledast}$ and a rewriting rule $\mathbf{S}_1\mathbf{S}_2 \to \mathbf{S}_1'\mathbf{S}_2'$ in $\mathcal{R}(w)$ such that $\mathbf{U} = \mathbf{W}\mathbf{S}_1\mathbf{S}_2\mathbf{W}'$ and $\mathbf{U}' = \mathbf{W}\mathbf{S}_1'\mathbf{S}_2'\mathbf{W}'$. Since $\mathbf{S}_1 \circ \mathbf{S}_2 \sim \mathbf{S}_1' \circ \mathbf{S}_2'$, we have an isomorphism φ from $\mathbf{S}_1 \circ \mathbf{S}_2$ to $\mathbf{S}_1' \circ \mathbf{S}_2'$ that acts as the identity map on frontier vertices. This implies that $\mathring{\mathbf{U}} = \mathring{\mathbf{W}} \circ \mathbf{S}_1 \circ \mathbf{S}_2 \circ \mathring{\mathbf{W}}'$ is isomorphic to $\mathring{\mathbf{U}}' = \mathring{\mathbf{W}} \circ \mathbf{S}_1' \circ \mathbf{S}_2' \circ \mathring{\mathbf{W}}'$. □

An interesting question is whether, for each $w \in \mathbb{N}$, there is some $w' \in \mathbb{N}$ such that any two unit decompositions $\mathbf{U}$ and $\mathbf{U}'$ in $\mathbf{\Sigma}(w)$ are $\mathcal{R}(w')$-equivalent if and only if $\mathring{\mathbf{U}}$ is isomorphic to $\mathring{\mathbf{U}}'$. The answer turns out to be yes, as shown in Theorem 98 below.

Theorem 98. *Let $\mathbf{U}$ and $\mathbf{U}'$ be unit decompositions in $\mathbf{\Sigma}(w)^{\circledast}$. Then, $\mathring{\mathbf{U}}$ is isomorphic to $\mathring{\mathbf{U}}'$ if and only if $\mathbf{U}$ and $\mathbf{U}'$ are $\mathcal{R}(2w)$-equivalent.*

Proof. Let $\mathbf{U} = \mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_n$ and $\mathbf{U}' = \mathbf{S}_1'\mathbf{S}_2' \cdots \mathbf{S}_n'$. Suppose that $\mathbf{U}$ and $\mathbf{U}'$ are $\mathcal{R}(2w)$-equivalent. Then, by Lemma 97, $\mathring{\mathbf{U}}$ is isomorphic to $\mathring{\mathbf{U}}'$.

For the converse, suppose that $\mathring{\mathbf{U}}$ is isomorphic to $\mathring{\mathbf{U}}'$ and let φ be an isomorphism from $\mathring{\mathbf{U}}$ to $\mathring{\mathbf{U}}'$. We show that $\mathbf{U}$ and $\mathbf{U}'$ are $\mathcal{R}(2w)$-equivalent.

Given a position $i \in [n-1]$ in the unit decomposition $\mathbf{U}$, a *swap* between $\mathbf{S}_i$ and $\mathbf{S}_{i+1}$ is a rewriting rule in $\mathcal{R}(w')$ for some w' that rewrites $\mathbf{U}$ into the unit decomposition

$$\mathbf{U}_i = \mathbf{S}_1\mathbf{S}_2 \cdots \mathbf{S}_{i-1}\mathbf{S}_i''\mathbf{S}_{i+1}''\mathbf{S}_{i+2} \cdots \mathbf{S}_n$$

such that, the function $\psi : [n] \to [n]$ that sets $\psi(p) = p$ for all $p \notin \{i, i+1\}$, $\psi(i) = i+1$ and $\psi(i+1) = i$ is an isomorphism from $\mathring{\mathbf{U}}$ to $\mathring{\mathbf{U}}_i$.

Intuitively, we swap the center vertex of $\mathbf{S}_i$ with the center vertex of $\mathbf{S}_{i+1}$. Note that, because of the twisting of the frontier, there may be several rewriting rules corresponding to such a swap. Now, a swap in the unit decomposition $\mathbf{U}$ corresponds to a swap in $\tau_{\mathbf{U}}$ as defined for linear orders in Section 8.1. The isomorphism φ defines a transformation of $\tau_{\mathbf{U}}$ into $\tau_{\mathbf{U}'}$.

By Proposition 95, $\mathrm{cw}(\mathring{\mathbf{U}}, \tau_{\mathbf{U}}) \leq w$ and $\mathrm{cw}(\mathring{\mathbf{U}}', \tau_{\mathbf{U}'}) \leq w$. Now, our result in Section 8.2 can be used for the slice rewriting system $\mathcal{R}(2w)$. More precisely, it follows from Theorem 88 that we can transform $\tau_{\mathbf{U}}$ into $\tau_{\mathbf{U}'}$ by a sequence of $\mathcal{O}(n^2)$ swaps and at each step, the cutwidth is at most $2w$. By using the rewriting rules from $\mathcal{R}(2w)$, we can replicate these swaps into the unit decomposition $\mathbf{U}$, obtaining in this way a unit decomposition $\mathbf{U}''$ such that $\tau_{\mathbf{U}''} = \tau_{\mathbf{U}'}$. Since $\mathring{\mathbf{U}}'' = \mathring{\mathbf{U}}'$, we have that $\mathbf{U}''$ is a twisting of $\mathbf{U}'$. Therefore, it follows from Proposition 94 that $\mathbf{U}''$ can be further transformed into $\mathbf{U}'$ by applying a sequence of $n-1$ rewriting rules from $\mathcal{R}(w) \subseteq \mathcal{R}(2w)$.

Hence, $\mathbf{U}$ can be rewritten into $\mathbf{U}''$ by applying $\mathcal{O}(n^2)$ rewriting rules from $\mathcal{R}(2w)$. □

Theorem 98 allows us to establish connections between the graph isomorphism problem for graphs of cutwidth at most w and the reachability problem in $\mathcal{R}(2w)$.

Theorem 99 ([Giannopoulou et al., 2019]). *Let G be an n-vertex graph of cutwidth w. We can compute a linear order τ of the vertices of G of width w in time $2^{\mathcal{O}(w^2 \log w)} \cdot n$.*

Theorem 100. *Graph isomorphism for n-vertex graphs of cutwidth at most w can be reduced in time $2^{\mathcal{O}(w^2 \log w)} \cdot n$ to $\mathcal{R}(2w)$-reachability.*

Proof. Given n-vertex graphs G and G' of cutwidth at most w, we first compute in time $2^{\mathcal{O}(w^2 \log w)} \cdot n$ linear orders τ and τ' of the vertex sets of G and G', respectively, of cutwidth at most w. Then, from Proposition 96, we construct unit decompositions $\mathbf{U}$ and $\mathbf{U}'$ such that $\tau_{\mathbf{U}} = \tau$, $\tau_{\mathbf{U}'} = \tau'$, G is isomorphic to $\mathring{\mathbf{U}}$ and G' is isomorphic to $\mathring{\mathbf{U}}'$. By Proposition 96, those decompositions belong to $\mathbf{\Sigma}(w)^{\circledast}$. By Theorem 98, we have that $\mathring{\mathbf{U}}$ and $\mathring{\mathbf{U}}'$ are isomorphic if and only if $\mathbf{U}$ and $\mathbf{U}'$ are $\mathcal{R}(2w)$-equivalent. □

8.4 Bounded Vertex Separation Number Order Reconfiguration

In this section, we show that the techniques employed in Section 8.2 for total orders of bounded cutwidth can be generalized to the context of orders of bounded vertex-separation number (Theorem 101). We consider that this generalization may be of independent interest in the theory of reconfiguration since vertex separation number is a width measure for graphs that is strictly more expressive than cutwidth. We note that it is not clear to us how to establish a direct connection between this generalized order reconfiguration problem and string rewriting as in the case of cutwidth.

For each $w \in \mathbb{N}$ and each n-vertex graph G, let $\mathrm{VSN}(G,w) = \{\tau : [n] \to V(G) : \mathrm{vsn}(G,\tau) \leq w\}$ be the set of linear orders of $V(G)$ of vertex separation number at most w. We say that τ can be *reconfigured* into τ' in vertex separation number at most w if there is a *reconfiguration sequence* $\tau = \tau_0 \xrightarrow{i_1} \tau_1 \xrightarrow{i_2} \cdots \xrightarrow{i_r} \tau_r = \tau'$ such that for each $j \in [r]$, $\tau_j \in \mathrm{VSN}(G,w)$.

Problem name: Bounded Vertex Separation Number Reconfiguration
Given: An n-vertex graph G, two linear order $\tau, \tau' : [n] \to V(G)$ on the vertex set of G, and a non-negative integer $w \in \mathbb{N}$.
Output: Is it true that τ can be reconfigured into τ' in vertex separation number at most w?

The proof of Theorem 101 below is analogous to the proof of Theorem 88. More precisely, this proof follows by restating Lemma 89, Lemma 90 and Lemma 91 in terms of the vertex separation number of a graph instead of cutwidth, and then by using this last restated lemma to conclude the proof, as done in Theorem 88.

Theorem 101. *Let G be an n-vertex graph and $\tau, \tau' : [n] \to V(G)$ be linear orders of $V(G)$ of vertex separation number at most w. Then, τ can be reconfigured into τ' in vertex separation number at most* $\mathrm{vsn}(G,\tau) + \mathrm{vsn}(G,\tau') \leq 2w$.

8.5 Discussions

In this chapter, we have studied the order reconfiguration problem under the framework of the theory of fixed-parameter tractability. In particular, in our main technical result, we have shown that the order reconfiguration problem for linear orders of cutwidth at most w can always be achieved in cutwidth at most $2w$ (Theorem 88). Using this result, we have established new connections between the graph isomorphism problem and the reachability problem for a special two-letter string rewriting system operating on unit slices. In particular, we have proven that unit decompositions $\mathbf{U}$ and $\mathbf{U}'$ of width w are $R(2w)$-equivalent if and only if the graphs $\mathring{\mathbf{U}}$ and $\mathring{\mathbf{U}}'$ are isomorphic (Theorem 98).

Theorem 98 opens up the possibility of studying the graph isomorphism problem under the perspective of term rewriting theory. The most immediate question in this direction is the complexity of deciding $R(2w)$-reachability for unit decompositions of width w. By a reduction to isomorphism of graphs of cutwidth w, this problem can be solved in time $2^{\mathcal{O}(w \cdot \mathsf{polylog} w)} n^{\mathcal{O}(1)}$ using the results from [Grohe et al., 2018b]. Can techniques that are intrinsic to string/term rewriting theory be used to improve this running time? Can such techniques be used to obtain algorithms running in time $f(w) \cdot n^{\mathcal{O}(1)}$ for some computable function $f : \mathbb{N} \to \mathbb{N}$? Note that a positive answer to this question would be conceptually relevant even if the function $f(w)$ is substantially worse than the current $2^{\mathcal{O}(w \cdot \mathsf{polylog}(w))}$, since techniques based on rewriting may carry over to contexts where group theoretic techniques do not. One interesting line of attack for this question would be to study connections between $R(2w)$ and techniques related to Knuth-Bendix completion and their generalizations [Sternagel and Thiemann, 2013, Wehrman et al., 2006, Kapur and Narendran, 1985, Hirokawa et al., 2019].

A natural question that arises in the context of reconfiguration of linear orders is the following: given two linear orders τ and τ', what is the minimum cutwidth of a linear order τ'' occurring in a reconfiguration sequence from τ to τ'? Is this problem NP-hard, or hard to approximate? Is it solvable in FPT-time for certain parameters? We thank one of the reviewers of [Arrighi et al., 2021a] for bringing these interesting questions to our attention.

Bibliography

[Abadir and Ferguson, 1990] Abadir, M. S. and Ferguson, J. (1990). An improved layout verification algorithm (LAVA). In Adshead, G. and Jess, J. A. G., editors, *European Design Automation Conference, EURO-DAC 1990, Glasgow, Scotland, UK, March 12-15, 1990*, pages 391–395. IEEE Computer Society.

[Abbassi et al., 2013] Abbassi, Z., Mirrokni, V. S., and Thakur, M. (2013). Diversity maximization under matroid constraints. In Dhillon, I. S., Koren, Y., Ghani, R., Senator, T. E., Bradley, P., Parekh, R., He, J., Grossman, R. L., and Uthurusamy, R., editors, *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 32–40. ACM.

[Abdulrahim and Misra, 1998] Abdulrahim, M. A. and Misra, M. (1998). A graph isomorphism algorithm for object recognition. *Pattern Anal. Appl.*, 1(3):189–201.

[Adolphson and Hu, 1973] Adolphson, D. and Hu, T. C. (1973). Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3):403–423.

[Adomavicius and Kwon, 2014] Adomavicius, G. and Kwon, Y. (2014). Optimization-based approaches for maximizing aggregate recommendation diversity. *INFORMS Journal on Computing*, 26(2):351–369.

[Alon et al., 2009] Alon, N., Lokshtanov, D., and Saurabh, S. (2009). Fast FAST. In Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S. E., and Thomas, W., editors, *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 49–58. Springer.

[Arora and Barak, 2009] Arora, S. and Barak, B. (2009). *Computational Complexity - A Modern Approach.* Cambridge University Press.

[Arrighi and de Oliveira Oliveira, 2021] Arrighi, E. and de Oliveira Oliveira, M. (2021). Three is enough for steiner trees. In Coudert, D. and Natale, E., editors, *19th Inter-*

national Symposium on Experimental Algorithms, SEA 2021, June 7-9, 2021, Nice, France, volume 190 of *LIPIcs*, pages 5:1–5:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Arrighi et al., 2020] Arrighi, E., Fernau, H., de Oliveira Oliveira, M., and Wolf, P. (2020). Width notions for ordering-related problems. In Saxena, N. and Simon, S., editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 182 of *LIPIcs*, pages 9:1–9:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Arrighi et al., 2021a] Arrighi, E., Fernau, H., de Oliveira Oliveira, M., and Wolf, P. (2021a). Order reconfiguration under width constraints. In Bonchi, F. and Puglisi, S. J., editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 8:1–8:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Arrighi et al., 2021b] Arrighi, E., Fernau, H., Hoffmann, S., Holzer, M., Jecker, I., de Oliveira Oliveira, M., and Wolf, P. (2021b). On the complexity of intersection non-emptiness for star-free language classes. In Bojanczyk, M. and Chekuri, C., editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*, volume 213 of *LIPIcs*, pages 34:1–34:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Arrighi et al., 2021c] Arrighi, E., Fernau, H., Lokshtanov, D., de Oliveira Oliveira, M., and Wolf, P. (2021c). Diversity in kemeny rank aggregation: A parameterized approach. In Zhou, Z., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 10–16. ijcai.org.

[Baader and Nipkow, 1999] Baader, F. and Nipkow, T. (1999). *Term Rewriting and All That*. Cambridge University Press.

[Babai, 2016] Babai, L. (2016). Graph isomorphism in quasipolynomial time [extended abstract]. In Wichs, D. and Mansour, Y., editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 684–697. ACM.

[Babai et al., 1982] Babai, L., Grigoryev, D. Y., and Mount, D. M. (1982). Isomorphism of graphs with bounded eigenvalue multiplicity. In Lewis, H. R., Simons, B. B., Burkhard, W. A., and Landweber, L. H., editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, STOC '82*, pages 310–324. ACM.

[Babai et al., 1983] Babai, L., Kantor, W. M., and Luks, E. M. (1983). Computational complexity and the classification of finite simple groups. In *24th Annual Symposium on Foundations of Computer Science, FOCS '83*, pages 162–171. IEEE.

[Barendsen, 2003] Barendsen, E. (2003). *Term Rewriting Systems*. Cambridge University Press.

[Bartholdi et al., 1989] Bartholdi, III, J., Tovey, C. A., and Trick, M. A. (1989). Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6:157–165.

[Baste et al., 2020] Baste, J., Fellows, M. R., Jaffke, L., Masarík, T., de Oliveira Oliveira, M., Philip, G., and Rosamond, F. A. (2020). Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. In Bessiere, C., editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1119–1125. ijcai.org.

[Baste et al., 2019] Baste, J., Jaffke, L., Masařík, T., Philip, G., and Rote, G. (2019). FPT algorithms for diverse collections of hitting sets. *Algorithms*, 12:254.

[Bastert and Matuszewski, 1999] Bastert, O. and Matuszewski, C. (1999). Layered drawings of digraphs. In Kaufmann, M. and Wagner, D., editors, *Drawing Graphs, Methods and Models (the book grow out of a Dagstuhl Seminar, April 1999)*, volume 2025 of *Lecture Notes in Computer Science*, pages 87–120. Springer.

[Battista et al., 1999] Battista, G. D., Eades, P., Tamassia, R., and Tollis, I. G. (1999). *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall.

[Berztiss, 1973] Berztiss, A. T. (1973). A backtrack procedure for isomorphism of directed graphs. *J. ACM*, 20(3):365–377.

[Betzler et al., 2009] Betzler, N., Fellows, M. R., Guo, J., Niedermeier, R., and Rosamond, F. A. (2009). Fixed-parameter algorithms for kemeny rankings. *Theor. Comput. Sci.*, 410(45):4554–4570.

[Bodlaender, 2012] Bodlaender, H. L. (2012). Fixed-parameter tractability of treewidth and pathwidth. In Bodlaender, H. L., Downey, R., Fomin, F. V., and Marx, D., editors, *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 196–227. Springer.

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

[Book and Otto, 1993] Book, R. V. and Otto, F. (1993). String-rewriting systems. In *String-Rewriting Systems*, pages 35–64. Springer.

[Booth and Lueker, 1976] Booth, K. S. and Lueker, G. S. (1976). Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379.

[Botafogo, 1993] Botafogo, R. A. (1993). Cluster analysis for hypertext systems. In Korfhage, R., Rasmussen, E. M., and Willett, P., editors, *Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Pittsburgh, PA, USA, June 27 - July 1, 1993*, pages 116–125. ACM.

[Bouchitté and Habib, 1987] Bouchitté, V. and Habib, M. (1987). NP-completeness properties about linear extensions. *Order*, 4(2):143–154.

[Brandenburg et al., 2013] Brandenburg, F., Gleißner, A., and Hofmeier, A. (2013). Comparing and aggregating partial orders with kendall tau distances. *Discrete Math., Alg. and Appl.*, 5(2).

[Brandenburg and Gleißner, 2016] Brandenburg, F. J. and Gleißner, A. (2016). Ranking chain sum orders. *Theor. Comput. Sci.*, 636:66–76.

[Bredereck et al., 2020] Bredereck, R., Kaczmarczyk, A., and Niedermeier, R. (2020). Electing successive committees: Complexity and algorithms. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020*, pages 1846–1853. AAAI Press.

[Brzozowski, 1976] Brzozowski, J. A. (1976). Hierarchies of aperiodic languages. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 10(2):33–49.

[Buchheim et al., 2010] Buchheim, C., Wiegele, A., and Zheng, L. (2010). Exact algorithms for the quadratic linear ordering problem. *INFORMS J. Comput.*, 22(1):168–177.

[Çakiroglu et al., 2009] Çakiroglu, O. A., Erten, C., Karatas, Ö., and Sözdinler, M. (2009). Crossing minimization in weighted bipartite graphs. *J. Discrete Algorithms*, 7(4):439–452.

[Casel et al., 2019] Casel, K., Day, J. D., Fleischmann, P., Kociumaka, T., Manea, F., and Schmid, M. L. (2019). Graph and string parameters: Connections between pathwidth, cutwidth and the locality number. In Baier, C., Chatzigiannakis, I., Flocchini, P., and Leonardi, S., editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 109:1–109:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Chan and Patrascu, 2010] Chan, T. M. and Patrascu, M. (2010). Counting inversions, offline orthogonal range counting, and related problems. In Charikar, M., editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 161–173. SIAM.

[Charbit et al., 2007] Charbit, P., Thomassé, S., and Yeo, A. (2007). The minimum feedback arc set problem is np-hard for tournaments. *Comb. Probab. Comput.*, 16(1):1–4.

[Charon and Hudry, 2007] Charon, I. and Hudry, O. (2007). A survey on the linear ordering problem for weighted or unweighted tournaments. *4OR*, 5(1):5–60.

[Cheng et al., 2004] Cheng, X., Li, Y., Du, D.-Z., and Ngo, H. Q. (2004). Steiner trees in industry. In *Handbook of combinatorial optimization*, pages 193–216. Springer.

[Chiang et al., 2013] Chiang, M., Lam, H., Liu, Z., and Poor, H. V. (2013). Why steiner-tree type algorithms work for community detection. In *Proc. of (AISTATS 2013)*, volume 31, pages 187–195.

[Chudnovsky et al., 2020] Chudnovsky, M., Pilipczuk, M., Pilipczuk, M., and Thomassé, S. (2020). Quasi-polynomial time approximation schemes for the maximum weight independent set problem in H-free graphs. In Chawla, S., editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2260–2278. SIAM.

[Cohen and Brzozowski, 1971] Cohen, R. S. and Brzozowski, J. A. (1971). Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5(1):1–16.

[Conte et al., 2003] Conte, D., Foggia, P., Sansone, C., and Vento, M. (2003). Graph matching applications in pattern recognition and image processing. In *Proceedings of the 2003 International Conference on Image Processing, ICIP 2003, Barcelona, Catalonia, Spain, September 14-18, 2003*, pages 21–24. IEEE.

[Cook, 1971] Cook, S. A. (1971). The complexity of theorem-proving procedures. In Harrison, M. A., Banerji, R. B., and Ullman, J. D., editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM.

[Cormen et al., 2001] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms, Second Edition.* The MIT Press and McGraw-Hill Book Company.

[Corneil et al., 2013] Corneil, D. G., Dalton, B., and Habib, M. (2013). LDFS-based certifying algorithm for the minimum path cover problem on cocomparability graphs. *SIAM J. Comput.*, 42(3):792–807.

[Corneil et al., 2009] Corneil, D. G., Olariu, S., and Stewart, L. (2009). The LBFS structure and recognition of interval graphs. *SIAM J. Discret. Math.*, 23(4):1905–1953.

[Cygan et al., 2015] Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. (2015). *Parameterized Algorithms.* Springer.

[Danna and Woodruff, 2009] Danna, E. and Woodruff, D. L. (2009). How to select a small set of diverse solutions to mixed integer programming problems. *Operations Research Letters*, 37(4):255–260.

[Darwiche and Marquis, 2002] Darwiche, A. and Marquis, P. (2002). A knowledge compilation map. *Journal of Artificial Intelligence Research (JAIR)*, 17:229–264.

[Díaz et al., 2002] Díaz, J., Petit, J., and Serna, M. (2002). A survey of graph layout problems. *ACM Computing Surveys (CSUR)*, 34(3):313–356.

[Downey and Fellows, 1999] Downey, R. G. and Fellows, M. R. (1999). *Parameterized Complexity.* Springer.

[Downey and Fellows, 2013] Downey, R. G. and Fellows, M. R. (2013). *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer.

[Dujmovic et al., 2006] Dujmovic, V., Fellows, M. R., Hallett, M. T., Kitching, M., Liotta, G., McCartin, C., Nishimura, N., Ragde, P., Rosamond, F. A., Suderman, M., Whitesides, S., and Wood, D. R. (2006). A fixed-parameter approach to 2-layer planarization. *Algorithmica*, 45(2):159–182.

[Dujmovic et al., 2003] Dujmovic, V., Fernau, H., and Kaufmann, M. (2003). Fixed parameter algorithms for one-sided crossing minimization revisited. In Liotta, G., editor, *Graph Drawing, 11th International Symposium, GD 2003, Perugia, Italy, September 21-24, 2003, Revised Papers*, volume 2912 of *Lecture Notes in Computer Science*, pages 332–344. Springer.

[Dujmovic et al., 2008] Dujmovic, V., Fernau, H., and Kaufmann, M. (2008). Fixed parameter algorithms for one-sided crossing minimization revisited. *J. Discrete Algorithms*, 6(2):313–323.

[Dujmovic and Whitesides, 2004] Dujmovic, V. and Whitesides, S. (2004). An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica*, 40(1):15–31.

[Dwork et al., 2001] Dwork, C., Kumar, R., Naor, M., and Sivakumar, D. (2001). Rank aggregation methods for the web. In Shen, V. Y., Saito, N., Lyu, M. R., and Zurko, M. E., editors, *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 613–622. ACM.

[Eades and Kelly, 1986] Eades, P. and Kelly, D. (1986). Heuristics for reducing crossings in 2-layered networks. *Ars Combinatoria*, 21A:89–98.

[Eades and Wormald, 1994] Eades, P. and Wormald, N. C. (1994). Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403.

[Eiben et al., 2019] Eiben, E., Ganian, R., Kangas, K., and Ordyniak, S. (2019). Counting linear extensions: Parameterizations by treewidth. *Algorithmica*, 81(4):1657–1683.

[Eiter et al., 2013] Eiter, T., Erdem, E., Erdogan, H., and Fink, M. (2013). Finding similar/diverse solutions in answer set programming. *Theory and Practice of Logic Programming*, 13(3):303–359.

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

[Fagin et al., 2004] Fagin, R., Kumar, R., Mahdian, M., Sivakumar, D., and Vee, E. (2004). Comparing and aggregating rankings with ties. In Beeri, C. and Deutsch, A., editors, *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 14-16, 2004, Paris, France*, pages 47–58. ACM.

[Fargier and Marquis, 2014] Fargier, H. and Marquis, P. (2014). Disjunctive closures for knowledge compilation. *Artificial Intelligence*, 216:129–162.

[Farkas and Timotity, 2019] Farkas, M. and Timotity, D. (2019). Voting issues: A brief history of preference aggregation. *WorldQuant*, pages 26:1–7.

[Faulon, 1998] Faulon, J. (1998). Isomorphism, automorphism partitioning, and canonical labeling can be solved in polynomial-time for molecular graphs. *J. Chem. Inf. Comput. Sci.*, 38(3):432–444.

[Fernau, 2005] Fernau, H. (2005). *Parameterized Algorithmics: A Graph-Theoretic Approach.* Habilitationsschrift, Universität Tübingen, Germany.

[Fernau et al., 2014] Fernau, H., Fomin, F. V., Lokshtanov, D., Mnich, M., Philip, G., and Saurabh, S. (2014). Social choice meets graph drawing: How to get subexponential time algorithms for ranking and drawing problems. *TSINGHUA SCIENCE AND TECHNOLOGY*, 19(4):374–386.

[Fernau and Krebs, 2017] Fernau, H. and Krebs, A. (2017). Problems on finite automata and the exponential time hypothesis. *Algorithms*, 10(1):24.

[Fishburn, 1985] Fishburn, P. C. (1985). *Interval Orders and Interval Graphs.* John Wiley.

[Flum and Grohe, 2006] Flum, J. and Grohe, M. (2006). *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer.

[Fomin et al., 2020] Fomin, F. V., Golovach, P. A., Jaffke, L., Philip, G., and Sagunov, D. (2020). Diverse pairs of matchings. In Cao, Y., Cheng, S., and Li, M., editors, *31st International Symposium on Algorithms and Computation (ISAAC 2020).* Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[Fomin et al., 2014] Fomin, F. V., Heggernes, P., Kratsch, D., Papadopoulos, C., and Villanger, Y. (2014). Enumerating minimal subset feedback vertex sets. *Algorithmica*, 69(1):216–231.

[Forster, 2004] Forster, M. (2004). A fast and simple heuristic for constrained two-level crossing reduction. In Pach, J., editor, *Graph Drawing, 12th International Symposium, GD 2004, New York, NY, USA, September 29 - October 2, 2004, Revised Selected Papers*, volume 3383 of *Lecture Notes in Computer Science*, pages 206–216. Springer.

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

[Furst et al., 1980] Furst, M. L., Hopcroft, J. E., and Luks, E. M. (1980). Polynomial-time algorithms for permutation groups. In *21st Annual Symposium on Foundations of Computer Science, FOCS '80*, pages 36–41. IEEE Computer Society.

[Gabor et al., 2018] Gabor, T., Belzner, L., Phan, T., and Schmid, K. (2018). Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *2018 IEEE International Conference on Autonomic Computing, ICAC 2018, Trento, Italy, September 3-7, 2018*, pages 131–140. IEEE Computer Society.

[Gallai, 1967] Gallai, T. (1967). Transitiv orientierbare Graphen. *Acta Mathematica Academiae Scientarum Hungaricae*, 18:25–66.

[Galle, 1989] Galle, P. (1989). Branch & sample: A simple strategy for constraint satisfaction. *BIT Numerical Mathematics*, 29(3):395–408.

[Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

[Ghouila-Houri, 1964] Ghouila-Houri, A. (1964). Flots et tensions dans un graphe. *Annales scientifiques de l'École Normale Supérieure*, 3e série, 81(3):267–339.

[Giannopoulou et al., 2019] Giannopoulou, A. C., Pilipczuk, M., Raymond, J.-F., Thilikos, D. M., and Wrochna, M. (2019). Cutwidth: obstructions and algorithmic aspects. *Algorithmica*, 81(2):557–588.

[Gilmore and Hoffman, 1964a] Gilmore, P. C. and Hoffman, A. J. (1964a). A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics*, 16:539–548.

[Gilmore and Hoffman, 1964b] Gilmore, P. C. and Hoffman, A. J. (1964b). A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics*, 16:539–548.

[Glover et al., 2000] Glover, F., Løkketangen, A., and Woodruff, D. L. (2000). Scatter search to generate diverse MIP solutions. In *Computing Tools for Modeling, Optimization and Simulation*, pages 299–317. Springer.

[Gollapudi and Sharma, 2009] Gollapudi, S. and Sharma, A. (2009). An axiomatic approach for result diversification. In Quemada, J., León, G., Maarek, Y. S., and Nejdl, W., editors, *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 381–390. ACM.

[Golovach et al., 2018] Golovach, P. A., Heggernes, P., Kanté, M. M., Kratsch, D., Sæther, S. H., and Villanger, Y. (2018). Output-polynomial enumeration on graphs of bounded (local) linear mim-width. *Algorithmica*, 80(2):714–741.

[Gori et al., 2004] Gori, M., Maggini, M., and Sarti, L. (2004). Graph matching using random walks. In *17th International Conference on Pattern Recognition, ICPR 2004, Cambridge, UK, August 23-26, 2004*, pages 394–397. IEEE Computer Society.

[Greistorfer et al., 2008] Greistorfer, P., Løkketangen, A., Voß, S., and Woodruff, D. L. (2008). Experiments concerning sequential versus simultaneous maximization of objective function and distance. *Journal of Heuristics*, 14(6):613–625.

[Grohe, 2012] Grohe, M. (2012). Fixed-point definability and polynomial time on graphs with excluded minors. *Journal of the ACM*, 59(5):27.

[Grohe and Marx, 2015] Grohe, M. and Marx, D. (2015). Structure theorem and isomorphism test for graphs with excluded topological subgraphs. *SIAM Journal on Computing*, 44(1):114–159.

[Grohe and Neuen, 2021] Grohe, M. and Neuen, D. (2021). Recent advances on the graph isomorphism problem. In Dabrowski, K. K., Gadouleau, M., Georgiou, N., Johnson, M., Mertzios, G. B., and Paulusma, D., editors, *Surveys in Combinatorics, 2021: Invited lectures from the 28th British Combinatorial Conference, Durham, UK, July 5-9, 2021*, pages 187–234. Cambridge University Press.

[Grohe et al., 2018a] Grohe, M., Neuen, D., and Schweitzer, P. (2018a). A faster isomorphism test for graphs of small degree. In Thorup, M., editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 89–100. IEEE.

[Grohe et al., 2018b] Grohe, M., Neuen, D., Schweitzer, P., and Wiebking, D. (2018b). An improved isomorphism test for bounded-tree-width graphs. In Chatzigiannakis, I., Kaklamanis, C., Marx, D., and Sannella, D., editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, pages 67:1–67:14.

[Grohe and Schweitzer, 2015] Grohe, M. and Schweitzer, P. (2015). Isomorphism testing for graphs of bounded rank width. In Guruswami, V., editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 1010–1029. IEEE.

[Grohe and Schweitzer, 2020] Grohe, M. and Schweitzer, P. (2020). The graph isomorphism problem. *Commun. ACM*, 63(11):128–134.

[Habib et al., 2000] Habib, M., McConnell, R. M., Paul, C., and Viennot, L. (2000). Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theor. Comput. Sci.*, 234(1-2):59–84.

[Habib and Möhring, 1994] Habib, M. and Möhring, R. H. (1994). Treewidth of cocomparability graphs and a new order-theoretic parameter. *Order*, 11(1):47–60.

[Hadžić et al., 2009] Hadžić, T., Holland, A., and O'Sullivan, B. (2009). Reasoning about optimal collections of solutions. In Gent, I. P., editor, *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, volume 5732 of *Lecture Notes in Computer Science*, pages 409–423. Springer.

[Hakimi, 1971] Hakimi, S. L. (1971). Steiner's problem in graphs and its implications. *Networks*, 1(2):113–133.

[Healy and Nikolov, 2013] Healy, P. and Nikolov, N. S. (2013). Hierarchical drawing algorithms. In Tamassia, R., editor, *Handbook on Graph Drawing and Visualization*, pages 409–453. Chapman and Hall/CRC.

[Hebrard et al., 2005] Hebrard, E., Hnich, B., O'Sullivan, B., and Walsh, T. (2005). Finding diverse and similar solutions in constraint programming. In Veloso, M. M. and Kambhampati, S., editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 372–377. AAAI Press / The MIT Press.

[Hebrard et al., 2007] Hebrard, E., O'Sullivan, B., and Walsh, T. (2007). Distance constraints in constraint satisfaction. In Veloso, M. M., editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 106–111.

[Hemaspaandra et al., 2005] Hemaspaandra, E., Spakowski, H., and Vogel, J. (2005). The complexity of Kemeny elections. *Theoretical Computer Science*, 349(3):382–391.

[Hirokawa et al., 2019] Hirokawa, N., Middeldorp, A., Sternagel, C., and Winkler, S. (2019). Abstract completion, formalized. *Logical Methods in Computer Science*, 15(3).

[Hsu and Ma, 1999] Hsu, W. and Ma, T. (1999). Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs. *SIAM J. Comput.*, 28(3):1004–1020.

[Hudry, 2008] Hudry, O. (2008). NP-hardness results for the aggregation of linear orders into median orders. *Annals of Operations Research*, 163:63–88.

[Impagliazzo and Paturi, 2001] Impagliazzo, R. and Paturi, R. (2001). On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375.

[Ingmar et al., 2020] Ingmar, L., de la Banda, M. G., Stuckey, P. J., and Tack, G. (2020). Modelling diversity of solutions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1528–1535.

[Ito et al., 2011] Ito, T., Demaine, E. D., Harvey, N. J. A., Papadimitriou, C. H., Sideri, M., Uehara, R., and Uno, Y. (2011). On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12-14):1054–1065.

[Ito et al., 2012] Ito, T., Kaminski, M., and Demaine, E. D. (2012). Reconfiguration of list edge-colorings in a graph. *Discret. Appl. Math.*, 160(15):2199–2207.

[Ito et al., 2014] Ito, T., Kaminski, M., Ono, H., Suzuki, A., Uehara, R., and Yamanaka, K. (2014). On the parameterized complexity for token jumping on graphs. In Gopal, T. V., Agrawal, M., Li, A., and Cooper, S. B., editors, *Theory and Applications of Models of Computation - 11th Annual Conference, TAMC 2014, Chennai, India, April 11-13, 2014. Proceedings*, volume 8402 of *Lecture Notes in Computer Science*, pages 341–351. Springer.

[Johnson et al., 2016] Johnson, M., Kratsch, D., Kratsch, S., Patel, V., and Paulusma, D. (2016). Finding shortest paths between graph colourings. *Algorithmica*, 75(2):295–321.

[Joobbani, 2012] Joobbani, R. (2012). *An artificial intelligence approach to VLSI routing*, volume 9. Springer Science & Business Media.

[Kanté et al., 2014] Kanté, M. M., Limouzy, V., Mary, A., and Nourine, L. (2014). On the enumeration of minimal dominating sets and related notions. *SIAM Journal on Discrete Mathematics*, 28(4):1916–1929.

[Kanté and Nourine, 2016] Kanté, M. M. and Nourine, L. (2016). Minimal dominating set enumeration. In *Encyclopedia of Algorithms*, pages 1287–1291.

[Kapur and Narendran, 1985] Kapur, D. and Narendran, P. (1985). The Knuth-Bendix completion procedure and Thue systems. *SIAM Journal on Computing*, 14(4):1052–1072.

[Karakostas et al., 2003] Karakostas, G., Lipton, R. J., and Viglas, A. (2003). On the complexity of intersecting finite state automata and NL versus NP. *Theoretical Computer Science*, 302(1):257–274.

[Karger, 2001] Karger, D. R. (2001). A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Rev.*, 43(3):499–522.

[Karp, 1972] Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York.

[Karpinski and Schudy, 2010] Karpinski, M. and Schudy, W. (2010). Faster algorithms for feedback arc set tournament, Kemeny rank aggregation and betweenness tournament. In Cheong, O., Chwa, K., and Park, K., editors, *Algorithms and Computation - 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I*, volume 6506 of *Lecture Notes in Computer Science*, pages 3–14. Springer.

[Kasai and Iwata, 1985] Kasai, T. and Iwata, S. (1985). Gradually intractable problems and nondeterministic log-space lower bounds. *Mathematical Systems Theory*, 18(1):153–170.

[Kemeny, 1959] Kemeny, J. G. (1959). Mathematics without numbers. *Daedalus*, 88:571–591.

[Kendall, 1938] Kendall, M. G. (1938). A new measure of rank correlation. *Biometrika*, 30(1/2):81–93.

[Kenyon-Mathieu and Schudy, 2007] Kenyon-Mathieu, C. and Schudy, W. (2007). How to rank with few errors. In Johnson, D. S. and Feige, U., editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 95–103. ACM.

[Keyder and Geffner, 2009] Keyder, E. and Geffner, H. (2009). Trees of shortest paths vs. steiner trees: Understanding and improving delete relaxation heuristics. In *Proc. of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1734–1739.

[Kinnersley, 1992] Kinnersley, N. G. (1992). The vertex separation number of a graph equals its path-width. *Inf. Process. Lett.*, 42(6):345–350.

[Klop, 1990] Klop, J. W. (1990). *Term Rewriting Systems.* Centrum voor Wiskunde en Informatica.

[Kobayashi et al., 2014] Kobayashi, Y., Maruta, H., Nakae, Y., and Tamaki, H. (2014). A linear edge kernel for two-layer crossing minimization. *Theor. Comput. Sci.*, 554:74–81.

[Kobayashi and Tamaki, 2015] Kobayashi, Y. and Tamaki, H. (2015). A fast and simple subexponential fixed parameter algorithm for one-sided crossing minimization. *Algorithmica*, 72(3):778–790.

[Köbler et al., 2011] Köbler, J., Kuhnert, S., Laubner, B., and Verbitsky, O. (2011). Interval graphs: Canonical representations in logspace. *SIAM J. Comput.*, 40(5):1292–1315.

[Köbler et al., 2015] Köbler, J., Kuhnert, S., and Watanabe, O. (2015). Interval graph representation with given interval and intersection lengths. *J. Discrete Algorithms*, 34:108–117.

[Kozen, 1977] Kozen, D. (1977). Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science, FOCS*, pages 254–266. IEEE Computer Society.

[Kratsch and Schweitzer, 2010] Kratsch, S. and Schweitzer, P. (2010). Isomorphism for graphs of bounded feedback vertex set number. In Kaplan, H., editor, *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory*, volume 6139 of *Lecture Notes in Computer Science*, pages 81–92. Springer.

[Lange and Rossmanith, 1992] Lange, K. and Rossmanith, P. (1992). The emptiness problem for intersections of regular languages. In Havel, I. M. and Koubek, V., editors, *Mathematical Foundations of Computer Science 1992, 17th International Symposium, MFCS*, volume 629 of *Lecture Notes in Computer Science*, pages 346–354. Springer.

[Lappas et al., 2009] Lappas, T., Liu, K., and Terzi, E. (2009). Finding a team of experts in social networks. In *Proc. of 15th International Conference on Knowledge Discovery and Data Mining (KDD 2009)*, pages 467–476.

[Lee and Younis, 2010] Lee, S. and Younis, M. F. (2010). Recovery from multiple simultaneous failures in wireless sensor networks using minimum steiner tree. *J. Parallel Distrib. Comput.*, 70(5):525–536.

[Lengauer, 1981] Lengauer, T. (1981). Black-white pebbles and graph separation. *Acta Informatica*, 16:465–475.

[List, 2013] List, C. (2013). Social Choice Theory. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2013 edition.

[Liu et al., 2011] Liu, L., Wu, B., and Yao, E. (2011). Minimizing the sum cost in linear extensions of a poset. *J. Comb. Optim.*, 21(2):247–253.

[Løkketangen and Woodruff, 2005] Løkketangen, A. and Woodruff, D. L. (2005). A distance function to support optimized selection decisions. *Decision Support Systems*, 39(3):345–354.

[Lokshtanov et al., 2011] Lokshtanov, D., Marx, D., and Saurabh, S. (2011). Lower bounds based on the exponential time hypothesis. *Bull. EATCS*, 105:41–72.

[Lokshtanov et al., 2014] Lokshtanov, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. (2014). Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 186–195. IEEE.

[Lowrance and Wagner, 1975] Lowrance, R. and Wagner, R. A. (1975). An extension of the string-to-string correction problem. *J. ACM*, 22(2):177–183.

[Luks, 1982] Luks, E. M. (1982). Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65.

[Makedon and Sudborough, 1989] Makedon, F. and Sudborough, I. H. (1989). On minimizing width in linear layouts. *Discrete Applied Mathematics*, 23(3):243–265.

[Marquis, 2011] Marquis, P. (2011). Existential closures for knowledge compilation. In Walsh, T., editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 996–1001. IJCAI/AAAI.

[Martí and Laguna, 2003] Martí, R. and Laguna, M. (2003). Heuristics and metaheuristics for 2-layer straight line crossing minimization. *Discrete Applied Mathematics*, 127(3):665–678.

[Messmer and Bunke, 1999] Messmer, B. T. and Bunke, H. (1999). A decision tree approach to graph and subgraph isomorphism detection. *Pattern Recognit.*, 32(12):1979–1998.

[Miller, 1980] Miller, G. (1980). Isomorphism testing for graphs of bounded genus. In Miller, R. E., Ginsburg, S., Burkhard, W. A., and Lipton, R. J., editors, *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, STOC '80*, pages 225–235. ACM.

[Monjardet, 1973] Monjardet, B. (1973). Tournois et ordres médians pour une opinion. *Mathématiques et Sciences humaines*, 43:55–73.

[Mouawad et al., 2014] Mouawad, A. E., Nishimura, N., Raman, V., and Wrochna, M. (2014). Reconfiguration over tree decompositions. In Cygan, M. and Heggernes, P., editors, *Parameterized and Exact Computation - 9th International Symposium, IPEC 2014, Wroclaw, Poland, September 10-12, 2014. Revised Selected Papers*, volume 8894 of *Lecture Notes in Computer Science*, pages 246–257. Springer.

[Muñoz et al., 2001] Muñoz, X., Unger, W., and Vrt'o, I. (2001). One sided crossing minimization is NP-hard for sparse graphs. In Mutzel, P., Jünger, M., and Leipert, S., editors, *Graph Drawing, 9th International Symposium, GD 2001 Vienna, Austria, September 23-26, 2001, Revised Papers*, volume 2265 of *Lecture Notes in Computer Science*, pages 115–123. Springer.

[Mutzel, 1995] Mutzel, P. (1995). A polyhedral approach to planar augmentation and related problems. In Spirakis, P. G., editor, *Algorithms - ESA '95, Third Annual European Symposium, Corfu, Greece, September 25-27, 1995, Proceedings*, volume 979 of *Lecture Notes in Computer Science*, pages 494–507. Springer.

[Mutzel, 2009] Mutzel, P. (2009). Optimization in leveled graphs. In Floudas, C. A. and Pardalos, P. M., editors, *Encyclopedia of Optimization, Second Edition*, pages 2813–2820. Springer.

[Nadel, 2011] Nadel, A. (2011). Generating diverse solutions in SAT. In Sakallah, K. A. and Simon, L., editors, *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, volume 6695 of *Lecture Notes in Computer Science*, pages 287–301. Springer.

[Nagamochi, 2005] Nagamochi, H. (2005). An improved bound on the one-sided minimum crossing number in two-layered drawings. *Discret. Comput. Geom.*, 33(4):569–591.

[Nishimura, 2018] Nishimura, N. (2018). Introduction to reconfiguration. *Algorithms*, 11(4):52.

[Ohlrich et al., 1993] Ohlrich, M., Ebeling, C., Ginting, E., and Sather, L. (1993). Subgemini: Identifying subcircuits using a fast subgraph isomorphism algorithm. In Dunlop, A. E., editor, *Proceedings of the 30th Design Automation Conference. Dallas, Texas, USA, June 14-18, 1993*, pages 31–37. ACM Press.

[Papadimitriou, 2007] Papadimitriou, C. H. (2007). *Computational complexity.* Academic Internet Publ.

[Pe'er and Shamir, 1997a] Pe'er, I. and Shamir, R. (1997a). Realizing interval graphs with size and distance constraints. *SIAM J. Discret. Math.*, 10(4):662–687.

[Pe'er and Shamir, 1997b] Pe'er, I. and Shamir, R. (1997b). Satisfiability problems on intervals and unit intervals. *Theor. Comput. Sci.*, 175(2):349–372.

[Pelz and Roettcher, 1991] Pelz, G. and Roettcher, U. (1991). Circuit comparison by hierarchical pattern matching. In *1991 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1993, Santa Clara, CA, USA, November 11-14, 1991. Digest of Technical Papers*, pages 290–293. IEEE Computer Society.

[Petit, 2011] Petit, J. (2011). Addenda to the survey of layout problems. *Bull. EATCS*, 105:177–201.

[Petit and Trapp, 2015] Petit, T. and Trapp, A. C. (2015). Finding diverse solutions of high quality to constraint optimization problems. In Yang, Q. and Wooldridge, M. J., editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 260–267. AAAI Press.

[Petit and Trapp, 2019] Petit, T. and Trapp, A. C. (2019). Enriching solutions to combinatorial problems via solution engineering. *INFORMS Journal on Computing*, 31(3):429–444.

[Place and Zeitoun, 2019] Place, T. and Zeitoun, M. (2019). Generic results for concatenation hierarchies. *Theory of Computing Systems*, 63(4):849–901.

[Rampersad and Shallit, 2010] Rampersad, N. and Shallit, J. (2010). Detecting patterns in finite regular and context-free languages. *Information Processing Letters*, 110(3):108–112.

[Reidenbach and Schmid, 2013] Reidenbach, D. and Schmid, M. L. (2013). Finding shuffle words that represent optimal scheduling of shared memory access. *Int. J. Comput. Math.*, 90(6):1292–1309.

[Reidenbach and Schmid, 2014] Reidenbach, D. and Schmid, M. L. (2014). Patterns with bounded treewidth. *Inf. Comput.*, 239:87–99.

[Schaefer, 2012] Schaefer, M. (2012). The graph crossing number and its variants: A survey. *The Electronic Journal of Combinatorics*, pages DS21–Dec.

[Schittekat and Sörensen, 2009] Schittekat, P. and Sörensen, K. (2009). OR practice — supporting 3PL decisions in the automotive industry by generating diverse solutions to a large-scale location-routing problem. *Operations Research*, 57(5):1058–1067.

[Schwentick et al., 2001] Schwentick, T., Thérien, D., and Vollmer, H. (2001). Partially-ordered two-way automata: A new characterization of DA. In Kuich, W., Rozenberg, G., and Salomaa, A., editors, *Developments in Language Theory, 5th International Conference, DLT*, volume 2295 of *Lecture Notes in Computer Science*, pages 239–250. Springer.

[Sechen, 2012] Sechen, C. (2012). *VLSI placement and global routing using simulated annealing*, volume 54. Springer Science & Business Media.

[Simjour, 2009] Simjour, N. (2009). Improved parameterized algorithms for the Kemeny aggregation problem. In Chen, J. and Fomin, F. V., editors, *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, volume 5917 of *Lecture Notes in Computer Science*, pages 312–323. Springer.

[Sipser, 1997] Sipser, M. (1997). *Introduction to the theory of computation.* PWS Publishing Company.

[Stallmann et al., 2001] Stallmann, M. F. M., Brglez, F., and Ghosh, D. (2001). Heuristics, experimental subjects, and treatment evaluation in bigraph crossing minimization. *ACM J. Exp. Algorithmics*, 6:8.

[Sternagel and Thiemann, 2013] Sternagel, C. and Thiemann, R. (2013). Formalizing Knuth-Bendix orders and Knuth-Bendix completion. In van Raamsdonk, F., editor, *24th International Conference on Rewriting Techniques and Applications, RTA 2013*, volume 21 of *LIPIcs*. Schloss Dagstuhl, Leibniz-Zentrum für Informatik.

[Straubing, 1981] Straubing, H. (1981). A generalization of the Schützenberger product of finite monoids. *Theoretical Computer Science*, 13:137–150.

[Straubing, 1985] Straubing, H. (1985). Finite semigroup varieties of the form V^*D. *Journal of Pure and Applied Algebra*, 36:53–94.

[Sugiyama et al., 1981] Sugiyama, K., Tagawa, S., and Toda, M. (1981). Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man Cybern.*, 11(2):109–125.

[Thérien, 1981] Thérien, D. (1981). Classification of finite monoids: the language approach. *Theoretical Computer Science*, 14(2):195–208.

[Thilikos et al., 2005] Thilikos, D. M., Serna, M. J., and Bodlaender, H. L. (2005). Cutwidth I: A linear time fixed parameter algorithm. *J. Algorithms*, 56(1):1–24.

[Trapp and Konrad, 2015] Trapp, A. C. and Konrad, R. A. (2015). Finding diverse optima and near-optima to binary integer programs. *IIE Transactions*, 47(11):1300–1312.

[Trotter, 1992] Trotter, W. T. (1992). *Combinatorics and Partially Ordered Sets: Dimension Theory.* Johns Hopkins University Press, Baltimore, Maryland.

[Trotter, 1997] Trotter, W. T. (1997). New perspectives on interval orders and interval graphs. In Bailey, R. A., editor, *Surveys in Combinatorics*, volume 241 of *London Mathematical Society Lecture Note Series*, pages 237–286.

[Vrt'o, 2008] Vrt'o, I. (2008). Crossing numbers of graphs: A bibliography. *Available electronically at ftp://ifi. savba. sk/pub/imrich/crobib. ps. gz.*

[Wareham, 2000] Wareham, T. (2000). The parameterized complexity of intersection and composition operations on sets of finite-state automata. In Yu, S. and Paun, A., editors, *Implementation and Application of Automata, 5th International Conference, CIAA*, volume 2088 of *Lecture Notes in Computer Science*, pages 302–310. Springer.

[Washio and Motoda, 2003] Washio, T. and Motoda, H. (2003). State of the art of graph-based data mining. *SIGKDD Explor.*, 5(1):59–68.

[Wehar, 2014] Wehar, M. (2014). Hardness results for intersection non-emptiness. In Esparza, J., Fraigniaud, P., Husfeldt, T., and Koutsoupias, E., editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 354–362. Springer.

[Wehar, 2016] Wehar, M. (2016). *On the Complexity of Intersection Non-Emptiness Problems.* PhD thesis, University at Buffalo.

[Wehrman et al., 2006] Wehrman, I., Stump, A., and Westbrook, E. (2006). Slothrop: Knuth-Bendix completion with a modern termination checker. In Pfenning, F., editor, *Term Rewriting and Applications, 17th International Conference, RTA 2006*, volume 4098 of *Lecture Notes in Computer Science*, pages 287–296. Springer.

[Wineberg and Oppacher, 2003] Wineberg, M. and Oppacher, F. (2003). The underlying similarity of diversity measures used in evolutionary computation. In Cantú-Paz, E., Foster, J. A., Deb, K., Davis, L., Roy, R., O'Reilly, U., Beyer, H., Standish, R. K., Kendall, G., Wilson, S. W., Harman, M., Wegener, J., Dasgupta, D., Potter, M. A., Schultz, A. C., Dowsland, K. A., Jonoska, N., and Miller, J. F., editors, *Genetic and Evolutionary Computation - GECCO 2003, Genetic and Evolutionary Computation Conference, Chicago, IL, USA, July 12-16, 2003. Proceedings, Part II*, volume 2724 of *Lecture Notes in Computer Science*, pages 1493–1504. Springer.

[Wong and Reingold, 1991] Wong, D. F. and Reingold, E. M. (1991). Probabilistic analysis of a grouping algorithm. *Algorithmica*, 6(2):192–206.

[Wrochna, 2018] Wrochna, M. (2018). Reconfiguration in bounded bandwidth and treedepth. *Journal of Computer and System Sciences*, 93:1–10.

[Young and Levenglick, 1978] Young, H. P. and Levenglick, A. (1978). A consistent extension of Condercet's election principle. *SIAM J. Appl. Math.*, 35(2):285–300.

[Zuylen and Williamson, 2009] Zuylen, A. v. and Williamson, D. P. (2009). Deterministic pivoting algorithms for constrained ranking and clustering problems. *Math. Oper. Res.*, 34(3):594–620.

Part III

Other Publications

Chapter 9

Three is Enough for Steiner Tree

Emmanuel Arrighi and Mateus de Oliveira Oliveira.

An extended abstract appeared in the proceedings of 19th International Symposium on Experimental Algorithms, SEA 2021, June 7–9, 2021, Nice, France:
Leibniz International Proceedings in Informatics (LIPIcs) 190 (2021) pp. 5:1—5:15.
DOI: 10.4230/LIPIcs.SEA.2021.5

Three is Enough for Steiner Trees

Emmanuel Arrighi[*1] and Mateus de Oliveira Oliveira[†1]

[1]University of Bergen, Norway

Abstract

In the Steiner tree in graph problem, the input consists of an edge-weighted graph G together with a set S of terminal vertices. The goal is to find a minimum weight tree in G that spans all terminals. This fundamental NP-hard problem has direct applications in many subfields of combinatorial optimization, such as planning, scheduling, etc. In this work we introduce a new heuristic for the Steiner tree problem, based on a simple routine for improving the cost of sub-optimal Steiner trees: first, the sub-optimal tree is split into three connected components, and then these components are reconnected by using an algorithm that computes an optimal Steiner tree with 3-terminals (the roots of the three components). We have implemented our heuristic into a solver and compared it with several state-of-the-art solvers on well-known data sets. Our solver performs very well across all the data sets, and outperforms most of the other benchmarked solvers on very large graphs, which have been either obtained from real-world applications or from randomly generated data sets.

1 Introduction

In the *Steiner tree* problem in graph, we are given an undirected graph G whose edges are weighted with non-negative values, and a subset of vertices S, whose elements are called *terminals.* The goal is to find a minimum-weight tree in G whose nodes span all terminal in S. This is a fundamental NP hard problem [Karp, 1972], which has been studied since the seventies [Hakimi, 1971] and which has found applications in several fields of research such as planning [Keyder and Geffner, 2009], social networks [Lappas et al., 2009], sensor

[*]The author was supported by research Council of Norway (no. 274526)
[†]The author was supported by research Council of Norway (no. 288761)

networks [Lee and Younis, 2010], community detection [Chiang et al., 2013], VLSI circuit design [Joobbani, 2012], as well as in numerous applications in industry [Cheng et al., 2004].

Since Steiner tree is an NP-hard problem, most research surrounding this problem has been devoted both to the task of developing heuristics that work reasonably well in practice, and to the task of developing approximation algorithms that provide approximation guarantees within polynomial time. In particular, a short list of heuristic paradigms that have been used to attack the Steiner-tree problem include simulated annealing [Lundy, 1985], genetic algorithms [Chakraborty, 2001], logic programming [Menai, 2009] and constraint solving [de Uña et al., 2016]. On the other hand, when it comes to approximation algorithms, the approximation ratio guarantee achievable by algorithms running in polynomial time was gradually improved from 2 [Takahashi, 1990] to 1.39 [Byrka et al., 2013] in a span of two and a half decades [Takahashi, 1990, Zelikovsky, 1993, Berman and Ramaiyer, 1994, Zelikovsky, 1996, Prömel and Steger, 1997, Karpinski and Zelikovsky, 1997, Hougardy and Prömel, 1999, Robins and Zelikovsky, 2000, Robins and Zelikovsky, 2005, Byrka et al., 2013]. It is worth noting that unless $P = NP$, the Steiner tree problem in general graphs cannot be approximated within a factor of $1 + \epsilon$ for sufficiently small $\epsilon > 0$ [Bern and Plassmann, 1989].

In this work, we introduce a new heuristic for the Steiner tree problem and show that on large graphs, it outperforms several state of the art algorithms. Our heuristic has two main components. First, we devise a method that can be used to quickly compute a good initial Steiner tree. The second component is based on an improvement procedure that takes a Steiner tree as input and tries to output a lighter Steiner tree. Essentially, this procedure is executed until it stabilizes, or until a specific time limit is up. It is worth noting that our improvement strategy is similar in spirit to an improvement procedure used in a celebrated approximation algorithm due to Robins and Zelikovsky [Robins and Zelikovsky, 2005].

This improvement procedure can be explained in two high-level steps. First, given a (potentially suboptimal) Steiner tree T_0, one appropriately split it into three subtrees T_1, T_2 and T_3 such that, all terminals are contained in $T_1 \cup T_2 \cup T_3$. Then those three subtrees are reconnected together by solving an instance of the Steiner tree problem with three terminals. This gives a new Steiner tree T_0'. As the Steiner tree problem with three terminals can be solve exactly and efficiently, the weight of T_0' is at most the weight of T_0.

We observe that there are two crucial differences between the optimization procedure used in our heuristic and the one used in the algorithm of [Robins and Zelikovsky, 2005].

The first is that their algorithm runs in a complete graph where for each two vertices v and u, the weight of the edge $\{v, u\}$ is the weight of the shortest path between v and u in the original graph, while our algorithm runs without the need to compute shortest paths between all possible pairs of vertices. The second difference is that in Robins and Zelikovsky's algorithm, the split is chosen to be the optimal one, while in our algorithm we replace optimality by a greedy selection strategy. Building the complete graph and looking for the optimum splitting is costly and cannot be done on large graphs. Therefore, the algorithm of [Robins and Zelikovsky, 2005] cannot handle large real world instances. By doing something that does not need such large structures our approach can handle large instances. As a consequence, our optimization procedure performs especially well on large graphs. We also note that this optimization procedure can also be used to improve the weight of sub-optimal Steiner trees output by other solvers.

To validate our new heuristic, we implement a solver in C++ and benchmark it against several state of the art solvers for the Steiner tree problem on well known data sets. These solvers implement several paradigms, such as genetic algorithms, linear programming algorithms, local search algorithms as well as algorithms with approximation guarantees. The data sets were obtained from a variety of sources, such as established real-world benchmarks for the Steiner tree problem, data sets of common use in the field of road networks, and a synthetic data set where instances are generated at random. Our solver obtained very good results in most data sets. In particular our solver was able to obtain solutions that are on par with those obtained by solvers that employed large scale mixed-linear programming suites such as SCIP [Gleixner et al., 2017]. Our solver was also able to handle very large instances, with millions of vertices and edges, while most of the solvers failed in these instances. A detailed exposition of these results can be found in section 4.

2 Preliminaries

In this section, we set notation for basic graph-theoretic concepts used in the description of our algorithm. We let $\mathbb{N}$ denote the set of *natural numbers*. For a finite set V, we let $\mathcal{P}(V, 2) = \{\{u, v\} \ : \ u, v \in V, u \neq v\}$ be the set of unordered pairs of elements from V.

An *undirected graph* is a pair $G = (V, E)$ where V is a set of *vertices* and $E \subseteq \mathcal{P}(V, 2)$ is a set of *undirected edges*. We may write $V(G)$ to denote the vertex-set of G and $E(G)$ to denote the edge-set of G. An *edge-weighted* graph is a graph $G = (V, E)$ together with a *cost function* $\text{cost} : E \to \mathbb{N}$. We let $\text{cost}(G)$ be the sum of the costs of all edges in G.

We say that a graph H is a *subgraph* of a graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For each subset $X \subseteq V(G)$, the subgraph of G induced by X is the graph $G[X]$ with vertex set X and edge set $E(G) \cap \mathcal{P}(X, 2)$.

A *walk* in a graph G is a sequence of vertices $v_1, \ldots, v_k$ such that for each i in $\{1, \ldots, k-1\}$, $\{v_i, v_{i+1}\} \in E(G)$. A *path* in G is a walk in which all vertices are distinct. We let $\text{dist}(v, v')$ be the minimum number of edges in a path between v and v'. We say that G is *connected* if for each two vertices v_1 and v_2 there is a path between v_1 and v_2. A *cycle* is a walk $v_1, \ldots, v_k$ such that $v_1 = v_k$ and $v_i \neq v_j$ for $i, j \leq k$ and $i \neq j$. A graph is *acyclic* if it contains no cycle.

A *tree* is a connected acyclic graph T. A *rooted tree* is a tree T together with a distinguished vertex r. If T is a rooted tree with root r, and $v \in V(T)$ is such that $r \neq v$, then the parent of v is the unique neighbour v' of v such that $\text{dist}(r, v') < \text{dist}(r, v)$. Note that the root r does not have a parent. Each neighbour v' of v with $\text{dist}(r, v') > \text{dist}(r, v)$ is called a *child* of v. A *leaf* of T is a vertex with no child. A *descendant* of a vertex v is a vertex v' such that the unique path between r and v' contains v. We consider v to be a descendant of itself. The *subtree of* T *rooted at* v is the subgraph of T induced by the set of descendants of v.

Given a graph G, a spanning tree of G is a tree T such that T is a subgraph of G and $V(T) = V(G)$. Given a connected edge-weighted graph G, and a vertex $v \in v(G)$, a *shortest-path tree* for G rooted at v is an edge-weighted spanning tree T of G rooted at v such that for each vertex $u \in V(G)$, the distance between v and u in G is equal to the distance between v and u in T.

Let G be an undirected edge-weighted graph and let $S \subseteq V(G)$ be a subset of vertices of G whose elements are called *terminals.* A *Steiner tree* in G is a subgraph T of G such that T is a tree and $S \subseteq V(T)$. We note that T may contain non-terminal vertices. We call the vertices in $V(T) \backslash S$, *Steiner points.* The cost of a tree, $\text{cost}(T)$, is the sum of the costs of its edges.

Let G be a graph and H be a connected subgraph of G. The *contraction* of H in G, written G/H, is the graph obtained from G by first deleting all vertices of H, then by adding a new vertex v_H, and finally by connecting v_H to a vertex $u \in V(G) \backslash V(H)$ in G/H if and only if there is an edge between u and some vertex from $V(H)$ in G. The weight of an edge between v_H and u is the minimum weight of an edge connecting a vertex of H to u. In our algorithm, we will need to temporarily contract a subgraph H multiple times. Doing the actual contraction of a subgraph and restoring the graph afterwards is a costly operation. To avoid such costly operations, we use several tricks to simulate the contraction of a subgraph in our implementation. Since our graphs are weighted, the contraction of a subgraph H can be simulated by simply setting the weights of the edges of H to 0, and therefore, the topology of the original graph remains unchanged. When we need to build the shortest path tree rooted in the vertex v_H, vertex obtained by the contraction of H, we simulate the contraction of H by using a custom initialisation of the Dijkstra's algorithm. In this case, the graph remains unchanged.

3 Our Heuristics

In this section, we describe the main components of the heuristic used in our Steiner tree solver 3TST. There are three main components. A pre-processing component, which simplifies the input graph, a greedy procedure that constructs an initial Steiner tree, an optimization function that takes a given Steiner tree as input and outputs another Steiner tree that is at least as light as the original one. This optimization procedure is then repeated until it stabilizes, or until the time is up. Once the solution can not be improved, our solver starts again with a new starting Steiner tree. It repeats this procedure until it receives a timeout signal. These three components are described in more details below.

3.1 Preprocessing

During the preprocessing step, we modify the input graph by applying two standard rules [Uchoa et al., 2002, Rehfeldt et al., 2019a] with the goal of eliminating redundancies. Once a solution is obtained in the modified graph, this solution can be easily converted into a solution to the original graph. The two preprocessing rules we apply are the following.

1. The first rule removes non-terminal vertices of degree 1 from the graph. These vertices are redundant because if a Steiner tree contains such a vertex, then one can safely delete it from the tree and still obtain a valid Steiner tree.
2. The second rule eliminates non-terminal vertices of degree 2. More precisely, let u be a non-terminal vertex of degree 2 connected to vertices v_1 and v_2 by edges $e_1 = \{u, v_1\}$ and $e_2 = \{u, v_2\}$ respectively. Then we delete the vertex u and the edges e_1 and e_2 from the graph. If the graph has an edge of cost c connecting v_1 and v_2, then we update the cost of this edge to $\min(c, \text{cost}(e_1) + \text{cost}(e_2))$. Otherwise, we just add a new edge $e = \{v_1, v_2\}$ of cost $\text{cost}(e_1) + \text{cost}(e_2)$ to the graph. This rule is repeated until no vertex of degree 2 is left.

This preprocessing step can be done in time $O(n)$ where n is the number of vertices. Note that if a solution to the modified graph contains an edge $e = \{u, v\}$ that is not present in the original graph, then one can obtain a solution to the original graph by replacing each such edge e by a path between u and v in which all internal vertices have degree 2.

3.2 Minimum Steiner trees with 2 or 3 terminals

A fact that we will use often both in the construction of an initial Steiner tree and in our optimization procedure is the fact that Steiner trees with two or three terminals can be computed very quickly by using elementary algorithms. Indeed, a Steiner tree with two terminal vertices t_0 and t_1 is simply a shortest path between these two vertices. On the other hand, it can be shown that if T is a Steiner tree with 3 terminals $\{t_0, t_1, t_2\}$ then there is a *center vertex* c such that T is obtained by taking the union of the shortest paths p_0, p_1 and p_2 between c and the terminals t_0, t_1 and t_2 respectively. We call c the center of T (Figure 1). We observe that c can be one of the terminals. Therefore, to construct such Steiner tree, we need to find the center c. Then, the Steiner tree will be the union of the shortest paths between each terminal t_0, t_1 and t_2 and c (p_0, p_1 and p_2). To find the center we proceeds as follow. We start by computing a shortest path tree rooted in t_0, t_1 and t_2 using the Dijkstra's algorithm. Then, we can iterate through the vertices of G and set as the center c the vertex that minimizes the sum of the lengths of the shortest paths p_0, p_1 and p_2. In this paper, we will call this procedure $\mathsf{3Steiner}(G, t_0, t_1, t_2)$. We note that $\mathsf{3Steiner}(G, t_0, t_1, t_2)$ is a deterministic procedure that produces an optimal Steiner tree with three terminals, and runs in time $O(m + n\log(n))$ where n the number of vertices and m the number of edges.

3Steiner is used in construction of an initial solution (subsection 3.3) and in the optimization procedure (subsection 3.4). In the later one, 3Steiner is often called with three terminals that are close to each other in G. Intuitively, in such case, we do not need to compute the full shortest path tree from each terminals, we just need to compute it up to some distance. More precisely, one can see that the center c cannot be further away from t_0 than t_1 or t_2, otherwise t_1 or t_2 would be a better center. Therefore, in 3Steiner, when computing the shortest path tree from t_0, we run the Dijkstra algorithm until it reaches one of the other terminals. The same argument and early stop criteria applies for computation of the shortest path trees rooted in t_1 and t_2.

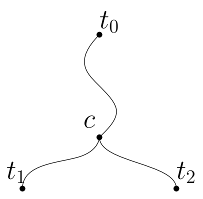


Figure 1: A Steiner tree with three terminals t_0, t_1 and t_2 is a union of shortest paths between a center vertex c and t_0, t_1 and t_2 respectively.

3.3 Constructing an Initial Solution

Once the preprocessing procedure has been applied, our algorithm proceeds to construct a suitable initial solution. We actually implement two initialization functions. Both functions take as input a triple (G, S, r) consisting of a graph G, a set of terminals S and a root vertex r, and return a Steiner tree covering the terminals in S rooted at r. We note that the root can be an arbitrary vertex in the graph, but in our implementation we always choose a terminal to be the root.

The first function, $\mathsf{DetInitialST}(G, S, r)$, is used to construct a reasonable first-solution. We use this procedure first, one time for each terminal in S. This function is completely deterministic. At each step, the function $\mathsf{DetInitialST}(G, S, r)$ maintains the following data:

1. a partial Steiner tree T spanning some of the terminals;
2. a graph G/T obtained from G by contracting T to its root r; and
3. a shortest-path tree D for G/T rooted at r.

In the beginning, T contains only the root r, $G/T = G$, and D is simply the shortest-path tree for G rooted at r. After this initialization has taken place, the algorithm enters in a loop, where at each iteration, at least two new terminals t_1 and t_2 are incorporated to the tree. Each iteration consists of three steps.

1. First, one applies a function $\mathsf{SelectTerminals}$ that selects the next two terminals t_1, t_2 that will be added to the tree. This function proceeds as follows. First, it sets t_1 as the terminal with *greatest* distance to the root vertex r in the graph G/T. Note that the contraction of T is simulated by setting the costs of its edges to 0 in the graph G. Subsequently, a shortest path from r to t_1 is temporarily contracted for the selection of t_2, and t_2 is selected as the terminal with the *greatest* distance to r.

Algorithm 1: DetInitialST(G, S, r)

```
Input: An edge weighted graph $G$, a set of terminals $S$, a vertex $r$
Output: A Steiner tree in $G$ connecting all terminals in $S$ rooted in $r$
$T \leftarrow r$
while there are two terminals in $S$ not spanned by $T$ do
    $G' \leftarrow G/T$
    $D \leftarrow$ ShortestPathTree$(G', r)$
    $t_1, t_2 \leftarrow$ SelectTerminals$(G', D)$
    $T' \leftarrow$ 3Steiner$(G', r, t_1, t_2)$
    $T \leftarrow T \cup T'$
end
if some terminal $t \in S$ is not spanned by $T$ then
    Set $T \leftarrow T \cup p$ where $p$ is a shortest path between $r$ and $t$ in $G/T$
end
return $T$
```

2. Once the terminals t_1 and t_2 have been determined, one calls the function 3Steiner to compute the minimum Steiner tree T' in G/T with respect to the terminal set $\{r, t_1, t_2\}$.

3. Finally, the two trees T and T' are *merged*. This merging process consists in taking the union $T \cup T'$. Note that the union $T \cup T'$ is a tree.

The three steps above are repeated until a Steiner tree spanning at least $|S|-1$ terminals in S has been obtained. If, at the end, a terminal $t \in S$ is not spanned by T, then the shortest path between r and t in G/T is added to T. The algorithm described above is specified more formally in algorithm 1. The procedure DetInitialST(G, S, r) has time complexity $O(|S| \cdot (m + n\log(n)))$, where n the number of vertices, m the number of edges of G.

Once we have a starting Steiner tree, we will improve it by applying the optimization procedure described in subsection 3.4. Since this optimization procedure may converge to a local minimum, we will repeat the optimization process with respect to several initial Steiner trees. To get several initial Steiner trees, we first use DetInitialST, using each of the terminal in S as root. DetInitialST is deterministic therefore, the only way to get different tree is by using a different starting vertex as root. From this point on, each initial Steiner tree will be selected using a much cheaper procedure, which we call RandomInitialST(G, S, r). This procedure simply selects random path between some terminal t_1 in S and the root vertex r. Subsequently, it selects a random path between some terminal t_2 and some vertex in the first path, then a random path between some terminal t_3 and some vertex in the previous paths and so on, until all terminals have been selected. Each random path is selected by performing a random walk in the graph starting at the terminal to be added.

3.4 Optimization Procedure

Once the preprocessing stage has been completed, and an initial Steiner tree has been computed using the procedure described in the previous subsection, our algorithm applies an optimization procedure that takes a Steiner tree T rooted at a *terminal vertex* r as input, and outputs a Steiner tree T', also rooted at r, with equal or smaller weight than T. This optimization procedure is repeated until the time is up or until it has stabilized. Alternatively, the procedure can be halted by an external algorithm even if it has not stabilized. In this case the best Steiner tree computed so far is given as the result.

Intuitively, this optimization procedure tries to improve T be replacing part of it by a locally optimal one, using the fact that we can compute efficiently an optimal solution when there is only 3 terminals. This optimization procedure works in two stages. In the first stage, we split the Steiner tree T into three subtrees T_1, T_2 and T_r, where T_1 and T_2 are rooted at vertices v_1 and v_2 respectively, and T_r is rooted at r. Subsequently, we reconnect the three subtrees by finding an optimal Steiner tree with respect to $\{T_1, T_2, T_r\}$.

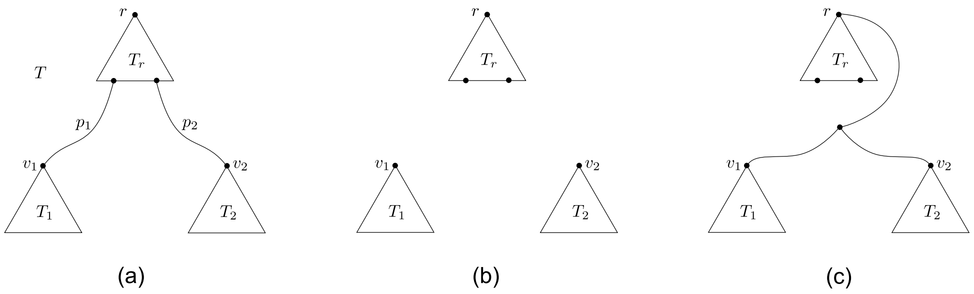


Figure 2: (a) A Steiner tree T, a pair $\{v_1, v_2\}$ of vertices in $\mathsf{SelectCut}(T)$, and Steiner paths p_1 and p_2. (b) The internal vertices of p_1 and p_2 are removed. This results into three trees T_r, T_1 and T_2. (c) $\{r, v_1, v_2\}$ are connected using an optimal 3-terminal Steiner tree obtained using $\mathsf{3Steiner}(G, r, v_1, v_2)$ function.

Before describing the details of the procedure, we need to define the concept of a *relevant vertex*. Let T be Steiner tree of G a rooted at a vertex r. We say that a vertex $v \in V(T)$ is *relevant* for T if v is a terminal or if v has at least 2 children in T. A path p in T is a *Steiner path* if the two endpoints of p are relevant for T and if the remaining vertices of p are not relevant, which means they are Steiner points of degree 2 in T. Note that each middle vertex of a Steiner path has a unique child. Let v and v' be relevant vertices. We say that v' is a *relevant child* of v if these two vertices are the endpoints of a relevant path in T and if $\text{dist}(r, v) < \text{dist}(r, v')$.

The algorithm starts by applying a simple routine that prunes the input Steiner tree. More precisely, this routine processes the input tree by removing every Steiner point that does not have a terminal as descendant. Such Steiner points do not connect the root to any terminal, and therefore can be safely removed. The resulting tree is still a Steiner tree and every leaf is a terminal.

Subsequently, the algorithm executes a procedure $\mathsf{Improve}(G, S, T)$ that takes a graph G, a set of terminals S and a tree T as input, and tries to modify T with the goal of reducing its cost by proceeding as follows.

1. First, we construct a list $\mathsf{SelectCut}(T)$ containing a selection of pairs of the form $\{v_1, v_2\}$ where both v_1 and v_2 are relevant vertices, $v_1 \neq r$, and $v_2 \neq r$. The list is constructed as follow. The vertices of T are traversed in a reverse depth-first search order. For each relevant vertex v in T, we add to the list $\mathsf{SelectCut}(T)$ all pairs of relevant children of v and if v is a terminal, all pairs containing v and a relevant child of v.

2. Now, for each pair of vertices (v_1, v_2) in the list $\mathsf{SelectCut}(T)$ built in the previous step, we call a function $\mathsf{Cut}(T, v_1, v_2)$ that cuts the tree above each of the vertices v_1 and v_2. More precisely, for each $i \in \{1, 2\}$, one deletes from T the internal vertices of the *unique* Steiner path p_i that starts at v_i that is contained in the unique path between v_i and r in T (Figure 2.(a)). Such Steiner paths p_1 and p_2 are always guaranteed to exist because the root is a relevant vertex. This process splits the original tree into three disjoint subtrees T_r, T_1, T_2 (Figure 2.(b)).

3. Subsequently, the algorithm contracts each of the three subtrees into a single vertex. More precisely, T_r is contracted to r, T_1 is contracted to v_1, and T_2 is contracted to v_2. We let G' be the contracted graph. We note that in practice, the contraction of a subtree is simulated by using a custom initialization in the Dijkstra algorithm.

4. Finally we apply the subroutine $\mathsf{3Steiner}(G', r, v_1, v_2)$ to computes an optimal 3-terminal Steiner tree with terminal set $\{r, v_1, v_2\}$ (Figure 2.(c)). This tree, together with the three subtrees T_r, T_1 and T_2 give rise to a tree $T_r \cup T_1 \cup T_2 \cup \mathsf{3Steiner}(G', r, v_1, v_2)$ whose weight is at most the weight of the input tree T. The algorithm then returns a spanning-tree of this graph.

A summary of the algorithm is provided in algorithm 2. The procedure $\mathsf{Improve}$ does not cut in all possible ways the tree T in three parts but tries to cut it at promising locations. These locations are selected by the function $\mathsf{SelectCut}$. This is done to speed up the convergence of the algorithm. The procedure Improve is receptively applied until it stabilizes. After that, we apply a variation of this procedure called ImproveFull. The only difference is that instead of using the function SelectCut in the first step, ImproveFull build a list of all pairs of relevant vertices using a function called FullCut.

Let t be the number of terminals in the graph and Δ be the maximum degree in T. Every leaf of T are terminal, therefore we have $\Delta \leq t$. We note that relevant vertices are either terminals or have at least 2 children in T. As the leaf of T are terminals. There is at most $O(t)$ Steiner points that are relevant vertices. Therefore, there are at most $O(t)$ relevant vertices. For each relevant vertices, the algorithm generates at most Δ^2 pair of vertices. Therefore, the time complexity of the function Improve(G, S, T) is $O(t \cdot \Delta^2 \cdot (m + n \log(n)))$, where n is the number of vertices and m is the number of edges in G. The complexity of the function ImproveFull(G, S, T) is $O(t^2 \cdot (m + n \log(n)))$.

Algorithm 2: Improve(G, S, T)

```
Input: An edge weighted graph $G$, a set of terminals $S$ and a Steiner tree $T$ in $G$
 spanning $S$
Output: A Steiner tree $T'$ of cost at most cost$(T)$ spanning $S$.
for $\{v_1, v_2\} \in$ SelectCut$(T)$ do
    $G' \leftarrow G$
    if $v_1 \in T$ and $v_2 \in T$ then
        $(T_r, T_1, T_2) \leftarrow$ Cut$(T, v_1, v_2)$
        $G' \leftarrow G'/T_r/T_1/T_2$      ($G'$ is obtained by contracting $T_r, T_1$ and $T_2$)
        $T \leftarrow T_r \cup T_1 \cup T_2 \cup$ 3Steiner$(G', r, v_1, v_2)$
    end
end
return $T$
```

4 Experimental results

We have implemented our heuristic algorithm in C++ and compared it with six state-of-the art solvers for the Steiner tree problem, including solvers that competed at the PACE challenge 2018 [Bonnet and Sikora, 2019]. We refer to our solver as 3TST, an acronym for *3-Terminal Steiner Tree*. We will also benchmark a variant of our algorithm called FAST-3TST. In this variation, the algorithm generate a single initial solution using DetInitialST applied to the first terminal, improve it until it stabilizes, and outputs the result. The idea behind FAST-3TST is to see the potential of the greedy algorithm and the optimization procedure. We will see that the results of FAST-3TST are closed to 3TST. The remaining solvers in our benchmarks are named according to the surnames, or initials of their respective authors. These solvers are listed below.

1. Grandcola's Solver[1] implements a local search algorithm.
2. HTKME Solver[2] combines a star contraction algorithm from [Dvorák et al., 2018] with several auxiliary heuristics.
3. HGSSB Solver[3] performs a shortest path heuristic followed by a local optimization step.
4. RCLG Solver[4] implements an evolutionary algorithm.
5. KR Solver[5] reduces the Steiner tree problem to a linear programming problem.
6. AO solver[6] is based on a local optimization heuristic.

We used the original implementation of each of these solvers in our benchmark, without any modification in the code. We benchmarked all the solvers on different data sets, some of which are well established datasets for the Steiner tree problem (PACE2018 dataset [Bonnet and Sikora, 2019], Vienna dataset), and some of which are well known datasets in the field of networks (Urban Road Networks set [Road Networks, 2016], Network repository [Rossi and Ahmed, 2015]). Finally, we also compared the solvers on synthetic data sets obtained by generating random d-regular graphs for distinct values of d.

To compare the different solver we need a *baseline value* for each graph in our benchmark. To get it, we use the exact solver SCIP-Jack [Rehfeldt et al., 2019b] to get an optimal solution if possible or a lower bound on the optimal solution. The SCIP-Jack solver uses the mixed integer programming framework SCIP to solve the Steiner tree problem in graph. Solutions obtain by SCIP-Jack to the dual problem allows us to get a lower bound on the insance. We run SCIP-Jack for 4 hours on a computer equipped with an Ryzen$^{\text{TM}}$ 1800X with at least 4Go of ram per instance running Ubuntu$^{\text{TM}}$ 21.04. Some of the graphs considered are too big to even get a lower bound using SCIP-Jack. In such case, we use the best value found by any solver during our experiments as the baseline value.

For each graph considered in our benchmark, we run each solver with a time limit of 30 minutes. When the time limit is reached, each solver received a Unix signal SIGTERM, and had 30 seconds to output a solution before being killed. This is a similar experimental setting as the one used in the PACE challenge 2018, whose theme was the Steiner tree

[1]`http://www.dil.univ-mrs.fr/~gcolas/sgls.c`
[2]`https://github.com/goderik01/PACE2018`
[3]`https://github.com/maxhort/Pacechallenge-TrackC/`
[4]`https://github.com/HeathcliffAC/SteinerTreeProblem`
[5]`https://github.com/dRehfeldt/scipjack/`
[6]`https://github.com/SteinerGardeners/TrackC-Version1`

problem in graph. Each solution is associated with a score, which is defined as the relative distance of the solution to the baseline value. If the value of the solution is v and the baseline value is b then the score is the ratio $\frac{v-b}{b}$. The score of a solver on a data set is the sum of the scores over all graphs in the data set. With this measure, the lower the score the better is the performance of the solver. In particular, a solver that gets a score of 0 in a given instance have either found an optimal solution or is the best solver on that instance.

For some instances of some data sets, some solvers did not output a feasible solution. In these cases, we assigned a default value for the instance. To avoid penalizing excessively a solver on such instances, we have defined the default value as the weight of a Steiner tree obtained by computing a minimum spanning tree of the input graph and subsequently by pruning this tree in such a way that each leaf is a terminal.

Each solver that uses a random procedure has an option to choose a particular seed with the goal of making a computation deterministic, and therefore reproducible. We run each solver 5 times with 5 different seeds and took the average cost of solutions. All the 5 seeds were chosen before the experiments were run. All our experiments were executed on computers equipped with a Intel Xeon-Gold$^{\text{TM}}$ 6138 with a memory limit set at 4Gb of RAM.

Set/Solvers	AO	Grandcolas	HGSSB	HTKME	KR	RCLG	3TST	FAST-3TST
PACE 2018	2.0091	1.9618	4.3455	1.0320	1.2737*	**0.8366**	1.4328^4	1.7787^5
Geo Original	0.3817	0.8114	1.1406	0.1504	0.3703*	0.1014	$\mathbf{0.0652}^1$	0.0704^2
Geo Prepro.	0.1321	0.0947	1.0366	0.1176	**0.0298**	0.0906	0.0778^2	0.0846^3
I Simple	0.0138	0.0447*	0.2565	0.0065	**0.0001**	0.0125	0.0060^2	0.0061^3
I Advanced	0.0301	0.0431	0.4576	0.0180	**0.0006**	0.0183	0.0224^4	0.0236^5
PUC	2.6413	2.3005	4.4214	1.3161	1.2249	**1.0044**	2.2534^4	2.9537^7
I640	1.1444	0.4658	4.8736	0.3898	**0.1638**	0.1793	0.9324^5	2.4845^7
3-regular	165.6978	163.1698	180.0352	165.0622	**160.4443***	160.9944	161.7175^3	161.7269^4
4-regular	65.8835	65.4648	73.4311	66.7534	70.9786*	**65.0612**	65.5426^4	65.5409^3
5-regular	222.8103	220.4505	243.6847	227.3415	536.6191*	**218.6553**	220.9386^3	220.9934^4
6-regular	0.4947	0.2588	1.2041	0.4716	9.0585*	**0.0923**	0.2701^3	0.2728^4
7-regular	0.6996	0.4149	1.1212	0.9136*	11.3531*	**0.0631**	0.3385^3	0.3207^2
8-regular	29.9533	29.6135	33.5098	30.6292	39.5680*	**29.2551**	29.7326^3	29.7371^4
9-regular	98.3775	95.6141	105.5675	100.2748	236.7506*	**94.0635**	96.5225^4	96.1116^3
10-regular	108.9216	106.5536	117.4406	111.6649	342.7558*	**104.4088**	107.4134^3	107.6474^4
20-regular	33.8401	32.9470	36.9325	34.7577*	42.4205*	**32.4760**	33.6287^3	33.8804^5
City road	5.3505*	2.4517*	6.5266*	0.7154	**0.0667**	0.3478	1.4210^4	1.4329^5
Big road	NC	NC	5.5116*	2.2711*	5.2166*	**0.0104***	1.5616^{*3}	0.9776^{*2}

Table 1: Summary: ratios for all algorithms and all data sets. The smallest the value the better is the solver on a given data set. A value of 0 means that the solver was the best in all instances of the data set. Values in bold are the smallest values on the dataset among all the solvers. The superscript number in the column of our solvers give the rank of our solvers on that data set. For example 0.0397^2 means that our solver is the second best solver on the data set. (*) means that for some instances, the solver did not output a feasible solution. NC means that the solver could not find a solution for any of the instances of the data set.

In all figures and tables, our implementation is called 3TST and the fast variant is called FAST-3TST. Table 1 summarises the comparison between all the solvers. This table gathers the sum of ratios obtained by each algorithm on each data set. The symbol "*" following an entry in the table is used to indicate that for some graphs in the data set, the solver did not output a feasible solution for at least one run. *NC* means that the solver did not output a feasible solution for any of the graph in the data set. We can see that our main implementation (3TST) obtains good results in most data sets, and on one dataset, our main implementation obtains the best score (Geo Original). Additionally, it is worth noting that the variation of our implementation is, with RCLG, the one that could find feasible solutions more often in the Big Road Networks data set, which contains graphs with millions of nodes. The difference between 3TST and FAST-3TST is that FAST-3TST need less memory, all the failure for our implementation comes from reaching the memory limit.

PACE-Challenge Graphs of the PACE Challenge 2018 dataset were selected by the organizers of the competition from the hard instances of the well known Steinlib and Vienna data sets. The average number of vertices is $27K$, the average number of edges is $48K$, and the average number of terminals is 1114, with a median at 360.5. Finally, most of these instances have treewidth above 40.

Figure 3 shows the score of each solver on each instance. Instances are sorted by increasing number of vertices. On the first half of the instances, both variants of our implementation provide decent solution but not as good as RCLG, KR or HTKME which are among the four first in the PACE Challenge 2018. And Figure 4 show a focus on the second half of the instances. Those are the larger instances of the dataset. On those larger instances with smaller average degree, both variant of our implementation is very good and on par with KR which is the best solver on this part of the data. We note that the implementation of KR is based on the SCIP Optimization Suite, a state-of-the-art tool for mixed integer programming [Gleixner et al., 2017]. We note that 3TST and FAST-3TST results are really closed. The main difference comes from the small instances on which 3TST have time to try more than one initial solution. On big instance, 3TST does not have time to finish stabilising the first initial solution and therefore is identical to FAST-3TST.

Vienna set Graphs in the Vienna set were generated from real-world telecommunication networks at the University of Vienna. This dataset is split into several types of instances. We realized our benchmark on the so called *I*-Instances sub-dataset and the Geo-Instances sub-dataset. *I*-Instances datases contains 85 instances representing deployment areas from various Austrian cities, but they also include rural areas with smaller population density and very sparse infrastructure. The underlying graphs contain between 7K and 178K nodes, 9K and 239K edges, and between 38 and 4991 terminals. I-instances are available after simple preprocessing that eliminates non-terminal nodes of degrees 1 and that contracts non-terminal nodes of degree 2. GEO-Instances contains 23 instances originating from an Austrian city, with different deployment areas and different density concerning the number of terminals. The graphs contain between $42K$ and $235K$ nodes, $52K$ and $366K$ edges, and between 88 and 6313 terminals. GEO-instances are available both in their original form, or after advanced preprocessing as proposed by Ribeiro et al. [Ribeiro et al., 2002].

Figure 5 shows the score of each solver on each instance of the I simple preprocessed instances dataset. The graph is zoom in to be able to see the variation between most of the solvers. As HGSSB has quite large ratios compared to the other solvers, its graph is outside of the figure for most of the instances. Instances are sorted by increasing number of vertices. We can see a similar behaviour as for the PACE Challenge instances. On small instances, both variations of our implementation give decent solutions and show their strength on larger instance where they give very good solutions. The instances of this data set are small enough so that the KR solver, which is base on an exact solver, manage to give the best solution in all case. KR solver use the same framework SCIP as the SCIP-Jack exact solver. On such data set solutions given by KR Solver are optimal or really close to optimal.

Figure 6 (resp. Figure 7) show the score of each solver on each instance of the Geo Original (resp. Geo Advanced Preprocessed) dataset. Our implementation performs well across the dataset. Both variants are just behind KR on most instances. Those two versions of the dataset, with and without preprocessing, show the impact of preprocessing on the different solvers. We can see that preprocessing have little impact for RCLG solver and our solvers. In opposition to our previous solver AO or Grandcolas solver which perform poorly on the big instance without preprocessing.

Steinlib [Koch et al., 2000] The SteinLib [Koch et al., 2000] is a collection of Steiner tree problems in graphs and variants. This library is focus on hard instances for the Steiner Tree problem on Graph. From this library, we use two datasets, namely PUC [Rosseti et al., 2001] and I640 [Duin, 1993]. In PUC, graphs are hypercubes chosen for their large integrality gaps and instances in I640 are random generated sparse graphs with so called incidence edge weights, which are chosen to defy preprocessing.

Figure 8 (resp. Figure 9) shows the score of each solver on each instance of the PUC (resp. I640) instances dataset. I640 is an old datasets, we note that most solvers solve every instances optimally or close to optimally. The PUC dataset is more interesting, instances are still hard and only few of them are solved optimally. On this dataset, our solvers perform inline with the other solvers.

d-regular graphs We generated random d-regular graphs using the random generator from the python package Networkx. The number of vertices were chosen uniformly at random from the range $[10000; 200000]$. The weights on the edge follow a normal distribution with mean uniformly chosen from the range $[2000; 10000]$ and standard deviation uniformly chosen from the range $[200; 2000]$. Negative weights were set to 0. The number of terminals was chosen uniformly at random between 2% and 10% of the number of vertices. Terminals were chosen uniformly at random from the vertices. We generated 10 graphs for each d in the set $\{3, 4, 5, 6, 7, 8, 9, 10, 20\}$.

Figure 10 and Table 1 show the evolution of the ratio for each solver with respect to the degree d of the vertices of the graphs. The gap between the costs obtained by the solver and the lower bound is really big, therefore, in Figure 10 the baseline value used is the best solution found during the experiment. The KR solver perform quite poorly on those datasets. The ratio obtained by the KR solver for $d \geq 4$ are arround 10, therefore they cannot be seen on Figure 10. The best solver in these datasets is the solver RCLG, which implements a genetic algorithm. The ratios obtained by our solvers (3TST) alternated between the third best and forth best. We note that, the solver KR, which reduces the Steiner tree problem to mixed integer-programming, started failing to give feasible solutions for some instances. We also note that for most of value of d, the ratio obtains by all the solvers are big. This seems to indicate that on those dataset, the exact solver SCIP-Jack does not manage to get good lower bound. This is coherent with the failures of KR on some of the instances.

City Road Networks This well known data set contains graphs associated with road networks for 80 of the most populated urban areas in the world. As the original graphs were not connected we filtered each instance by taking only the largest connected component of each graph. Since these graphs do not come originally with information about terminal nodes, we selected these terminals at random. First, we selected a number r uniformly at random in the range between 2% and 10% of the number of vertices. Subsequently, we selected r distinct vertices uniformly at random among the vertices of the graph. The graphs contain between $2K$ and $685K$ nodes, $3K$ and $924K$ edges and between 246 and 53275 terminals

Figure 11 shows the score of each solver on each instance of the City Road Networks set.

Solvers	AO	Grandcolas	HGSSB	HTKME	KR	RCLG	3TST	FAST-3TST
Instance 1	NC	NC	0.3803*	0.0226	NC	0.0010	0.1241^{4}	0.1241^{3}
Instance 2	NC	NC	NC	0.0062	NC	0.0001	0.0997^{4}	0.0996^{3}
Instance 3	NC	NC	NC	1.0189*	NC	0.0000	0.1359^{3}	0.1356^{2}
Instance 4	NC	NC	NC	0.0223	0.0002	0.0073	0.1246^{4}	0.1247^{5}
Instance 5	NC	NC	NC	0.0027	0.0000	0.0019	0.1046^{4}	0.1047^{5}
Instance 6	NC	NC	NC	NC	NC	0.0000	0.1331^{3}	0.1329^{2}
Instance 7	NC	NC	NC	NC	NC	0.0000	NC	0.1702^{2}
Instance 8	NC	NC	NC	NC	NC	0.0000	NC	0.0859^{2}
Instance 9	NC	NC	NC	NC	NC	NC	NC	NC

Table 2: Big road networks: ratios for all algorithms on the big road networks data set. The smallest the value the better is the solver on a given data set. A value of 0 means that the solver was the best the instance. The superscript number in the column of our solver give the rank of our solver on that instance. For example 0.0891^{2} means that our solver is the second best solver on the instance. NC means that the solver could not find a solution for the instance.

Instances are sorted by increasing number of vertices. We note that on the first half of the instances, almost all solvers manage to give really good solutions. As the size of the instances grow, the solver KR, which reduces Steiner tree to mixed-integer programming, starts to be the dominant best solver. Nevertheless, our solver still outputs solutions with a very good ratio (of at most 0.2). Unlike with the other datasets, our approach does not scale smoothly as the size of the graphs increase. In the second half of the dataset, the ratio obtained by our solvers oscillate between less than 0.02 and more than 0.1.

Big Road Networks In this data set was used to push the solvers to their limits. We selected 9 unweighted road networks with more than 1 million nodes. As in the previous dataset, the biggest connected component was kept, the number of terminals was chosen uniformly at random between 2% and 10% of the number of vertices. These graphs contain between $1087K$ and $23947K$ nodes, $1541K$ and $28854K$ edges, and between $52K$ and $2074K$ terminals

Table 2 shows the ratio of each solver on each of these five instances. On this dataset HGSSB, HTKME, KR, RCLG and our algorithm (3TST and FAST-3TST) managed to output some solution for some of the instances. AO and Grandcolas did not output any valid solution. On the 9 graphs, HGSSB outputs a solution on some of it run on the first instance, HTKME outputs some solution for the first 5 instances, KR outputs solutions for 2 instances, RCLG output solution for all except the biggest instance, and 3TST output solution for the first 6 instance and 3TST for all instance except the last one. This dataset highlights one of the strengths of our solver, which is the ability to handle very large instances and still give good solutions, when compared with other solvers. On this dataset, the failures of our implementation are due to the memory limit. The same applies for most the failures of the other solvers as well.

Running time of FAST-3TST FAST-3TST is the only solver that terminates by itself. All other tested solvers run until the timeout signal is received. Therefore, measuring the running time is only interesting for FAST-3TST. We have seen in our experiment, even if FAST-3TST is more restricted than 3TST, the results obtained are quite close. Figure 12, Figure 13 and Figure 14 show the running time of FAST-3TST on each instance of the PACE Challenge, PUC and I640 dataset. Let first note that on those 3 datasets, FAST-3TST runs significantly less than 30 min on many instances. This shows that the simple combination of DetInitialST and our optimization procedure outputs fast good solutions.

5 Conclusion

In this work, we introduced a simple combinatorial heuristic algorithm for the Steiner tree problem. Our heuristic is similar in spirit to the classic approximation algorithm of Robin and Zelikovsky [Robins and Zelikovsky, 2005], that works by replacing sub-trees of a prospective solution with Steiner trees on a small set of terminals. In our case, we use a routine that splits a prospective solution Steiner tree into three disjoint subtrees, and that reconnects these subtrees by taking the union with a 3-terminal Steiner tree, where the terminals are the roots of the subtrees. We note that one distinguishing feature of our algorithm is that it is well suited for large graphs, since it does not require the bookkeeping of the distances between all pairs of vertices in the graph. Indeed we almost only need to keep track of the edges of a slightly pre-processed version of the input graph, where non-terminal vertices of degree 1 are removed, and edges containing non-terminal vertices of degree 2 are contracted.

Our experimental results have shown that our algorithm fits well the category of a general purpose Steiner tree heuristic, since it was able to obtain good solutions in all benchmarked datasets when compared with other solvers. We note that the best solver in some datasets was built upon a state-of-the art mixed-integer programming package. In some other datasets, the best solver was based on genetic algorithms. On the other hand, our algorithm essentially consists in the application of a single simple replacement routine that is applied multiple times until the time limit is reached. Still the solutions obtained by our solvers were very competitive, often being the second best in the benchmarks and with a very small ratio $(v - b)/b$ where v is the weight of our solution and b a known lower bound or the weight of the best solver. It is also worth noting that our algorithm was able to handle graphs with millions of vertices, while most of the other solvers failed in all these big instances. Finally, it is worth noting that one possible application of our Steiner-tree improvement sub-routine is as a black-box that can be used to improve the solution output by other solvers.

References

[Berman and Ramaiyer, 1994] Berman, P. and Ramaiyer, V. (1994). Improved approximations for the steiner tree problem. *J. Algorithms*, 17(3):381–408.

[Bern and Plassmann, 1989] Bern, M. W. and Plassmann, P. E. (1989). The steiner problem with edge lengths 1 and 2. *Inf. Process. Lett.*, 32(4):171–176.

[Bonnet and Sikora, 2019] Bonnet, É. and Sikora, F. (2019). The PACE 2018 Parameterized Algorithms and Computational Experiments Challenge: The Third Iteration. In *Proc. of IPEC 2018*, volume 115, pages 26:1–26:15.

[Byrka et al., 2013] Byrka, J., Grandoni, F., Rothvoß, T., and Sanità, L. (2013). Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33.

[Chakraborty, 2001] Chakraborty, G. (2001). Genetic algorithm approaches to solve various steiner tree problems. In *Steiner Trees in Industry*, pages 29–69. Springer.

[Cheng et al., 2004] Cheng, X., Li, Y., Du, D.-Z., and Ngo, H. Q. (2004). Steiner trees in industry. In *Handbook of combinatorial optimization*, pages 193–216. Springer.

[Chiang et al., 2013] Chiang, M., Lam, H., Liu, Z., and Poor, H. V. (2013). Why steiner-tree type algorithms work for community detection. In *Proc. of (AISTATS 2013)*, volume 31, pages 187–195.

[de Uña et al., 2016] de Uña, D., Gange, G., Schachte, P., and Stuckey, P. J. (2016). Steiner tree problems with side constraints using constraint programming. In *Proc. of the 30th AAAI Conference on Artificial Intelligence.*

[Duin, 1993] Duin, C. (1993). *Steiner Problems in Graphs.* PhD thesis, University of Amsterdam.

[Dvorák et al., 2018] Dvorák, P., Feldmann, A. E., Knop, D., Masarík, T., Toufar, T., and Veselý, P. (2018). Parameterized approximation schemes for steiner trees with small number of steiner vertices. In *Proc. of (STACS 2018)*, volume 96, pages 26:1–26:15.

[Gleixner et al., 2017] Gleixner, A., Eifler, L., Gally, T., Gamrath, G., Gemander, P., Gottwald, R. L., Hendel, G., Hojny, C., Koch, T., Miltenberger, M., Müller, B., Pfetsch, M. E., Puchert, C., Rehfeldt, D., Schlösser, F., Serrano, F., Shinano, Y., Viernickel, J. M., Vigerske, S., Weninger, D., Witt, J. T., and Witzig, J. (2017). The SCIP Optimization Suite 5.0. ZIB-Report 17-61, Zuse Institute Berlin.

[Hakimi, 1971] Hakimi, S. L. (1971). Steiner's problem in graphs and its implications. *Networks*, 1(2):113–133.

[Hougardy and Prömel, 1999] Hougardy, S. and Prömel, H. J. (1999). A 1.598 approximation algorithm for the steiner problem in graphs. In *Proc. of the 10th Symposium on Discrete Algorithms (SODA 1999)*, pages 448–453.

[Joobbani, 2012] Joobbani, R. (2012). *An artificial intelligence approach to VLSI routing*, volume 9. Springer Science & Business Media.

[Karp, 1972] Karp, R. M. (1972). Reducibility among combinatorial problems. In *Proc. of Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103.

[Karpinski and Zelikovsky, 1997] Karpinski, M. and Zelikovsky, A. (1997). New approximation algorithms for the steiner tree problems. *J. Comb. Optim.*, 1(1):47–65.

[Keyder and Geffner, 2009] Keyder, E. and Geffner, H. (2009). Trees of shortest paths vs. steiner trees: Understanding and improving delete relaxation heuristics. In *Proc. of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1734–1739.

[Koch et al., 2000] Koch, T., Martin, A., and Voß, S. (2000). SteinLib: An updated library on steiner tree problems in graphs. Technical Report ZIB-Report 00-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, Berlin.

[Lappas et al., 2009] Lappas, T., Liu, K., and Terzi, E. (2009). Finding a team of experts in social networks. In *Proc. of 15th International Conference on Knowledge Discovery and Data Mining (KDD 2009)*, pages 467–476.

[Lee and Younis, 2010] Lee, S. and Younis, M. F. (2010). Recovery from multiple simultaneous failures in wireless sensor networks using minimum steiner tree. *J. Parallel Distrib. Comput.*, 70(5):525–536.

[Lundy, 1985] Lundy, M. (1985). Applications of the annealing algorithm to combinatorial problems in statistics. *Biometrika*, 72(1):191–198.

[Menai, 2009] Menai, M. E. B. (2009). A logic-based approach to solve the steiner tree problem. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 73–79. Springer.

[Prömel and Steger, 1997] Prömel, H. J. and Steger, A. (1997). Rnc-approximation algorithms for the steiner problem. In *Proc. of the 14th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1997), Proceedings*, volume 1200 of *LNCS*, pages 559–570.

[Rehfeldt et al., 2019a] Rehfeldt, D., Koch, T., and Maher, S. J. (2019a). Reduction techniques for the prize collecting steiner tree problem and the maximum-weight connected subgraph problem. *Networks*, 73(2):206–233.

[Rehfeldt et al., 2019b] Rehfeldt, D., Shinano, Y., and Koch, T. (2019b). Scip-jack: An exact high performance solver for steiner tree problems in graphs and related problems. In *Modeling, Simulation and Optimization of Complex Processes HPSC 2018*, Proceedings of the 7th International Conference on High Performance Scientific Computing. accepted for publication.

[Ribeiro et al., 2002] Ribeiro, C. C., Uchoa, E., and Werneck, R. F. F. (2002). A hybrid GRASP with perturbations for the steiner problem in graphs. *INFORMS Journal on Computing*, 14(3):228–246.

[Road Networks, 2016] Road Networks, U. (2016). Urban road network data.

[Robins and Zelikovsky, 2000] Robins, G. and Zelikovsky, A. (2000). Improved steiner tree approximation in graphs. In *Proc. of the 11th Symposium on Discrete Algorithms (SODA 2000)*, pages 770–779.

[Robins and Zelikovsky, 2005] Robins, G. and Zelikovsky, A. (2005). Tighter bounds for graph steiner tree approximation. *SIAM J. Discrete Math.*, 19(1):122–134.

[Rosseti et al., 2001] Rosseti, I., de Aragão, M., Ribeiro, C., Uchoa, E., and Werneck, R. (2001). New benchmark instances for the Steiner problem in graphs. In *Extended Abstracts of the 4th Metaheuristics International Conference (MIC'2001)*, pages 557–561, Porto.

[Rossi and Ahmed, 2015] Rossi, R. A. and Ahmed, N. K. (2015). The network data repository with interactive graph analytics and visualization. In *AAAI*.

[Takahashi, 1990] Takahashi, H. (1990). An approximate solution for the steiner problem in graphs. *Math. Japonica.*, 6:573–577.

[Uchoa et al., 2002] Uchoa, E., de Aragão, M. P., and Ribeiro, C. C. (2002). Preprocessing steiner problems from VLSI layout. *Networks*, 40(1):38–50.

[Zelikovsky, 1993] Zelikovsky, A. (1993). An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9(5):463–470.

[Zelikovsky, 1996] Zelikovsky, A. (1996). Better approximation bounds for the network and euclidean steiner tree problems. *University of Virginia, Charlottesville, VA*.

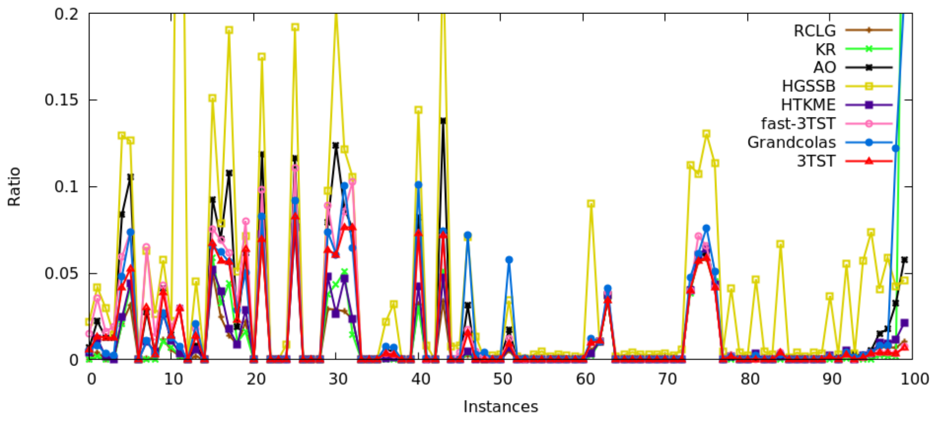


Figure 3: PACE Challenge: Show the ratio obtained by each solver on each instance of the PACE Challenge data set. Instances are sorted by increasing number of vertices.

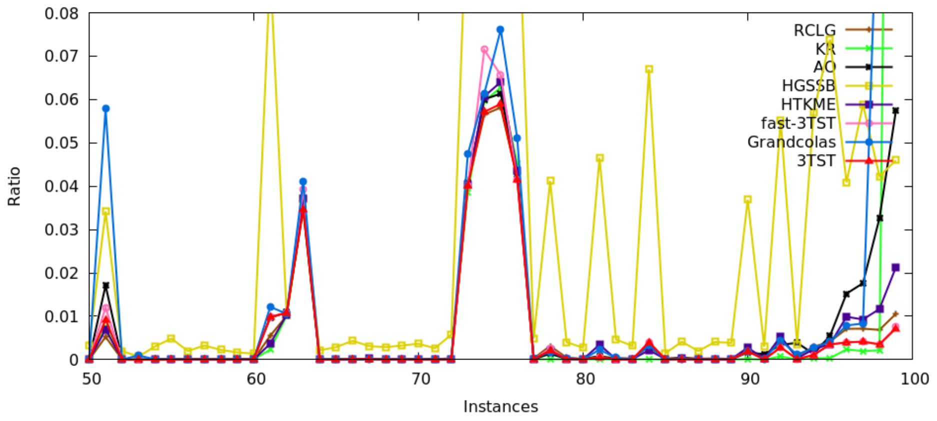


Figure 4: PACE Challenge: Show the ratio obtained by each solver on the 50 largest instances of the PACE Challenge data set. Instances are sorted by increasing number of vertices.

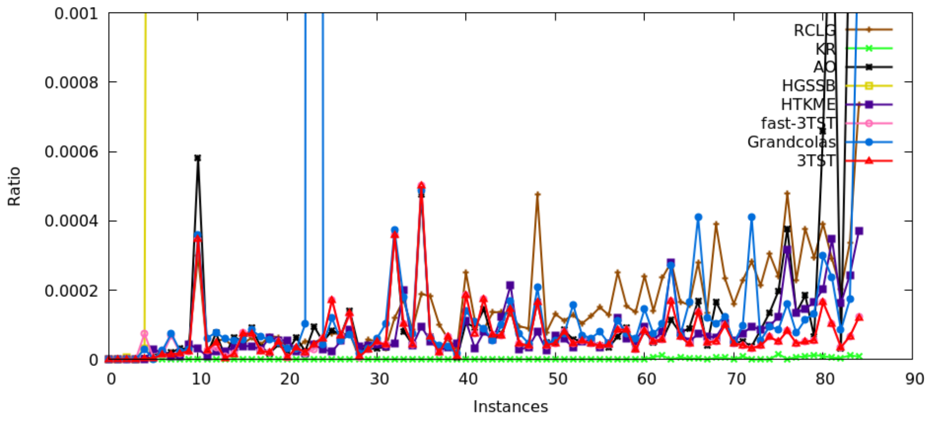


Figure 5: I simple: Show the ratio obtained by each solver on each instance of the I simple preprocessed instances data set. Instances are sorted by increasing number of vertices.

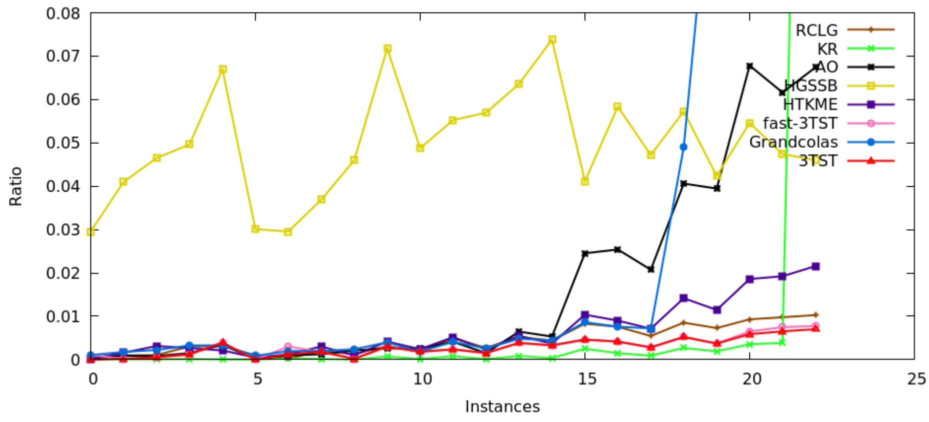


Figure 6: Geo Original: Show the ratio obtained by each solver on each instance of the Geo Original instances data set. Instances are sorted by increasing number of vertices.

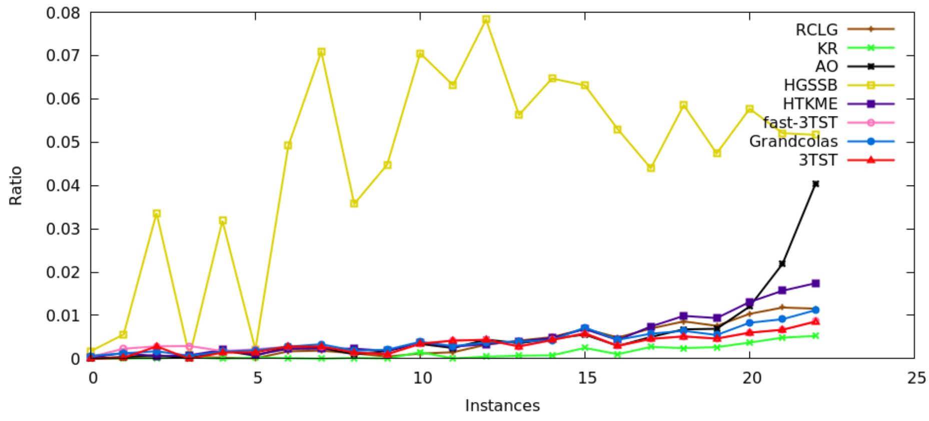


Figure 7: I simple: Show the ratio obtained by each solver on each instance of the Geo Advanced preprocessed instances data set. Instances are sorted by increasing number of vertices.

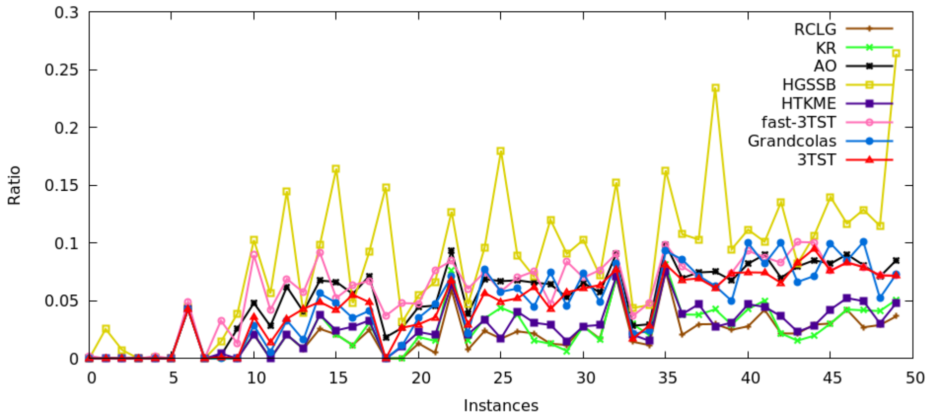


Figure 8: PUC: Show the ratio obtained by each solver on each instance of the PUC instances data set. Instances are sorted by increasing number of vertices.

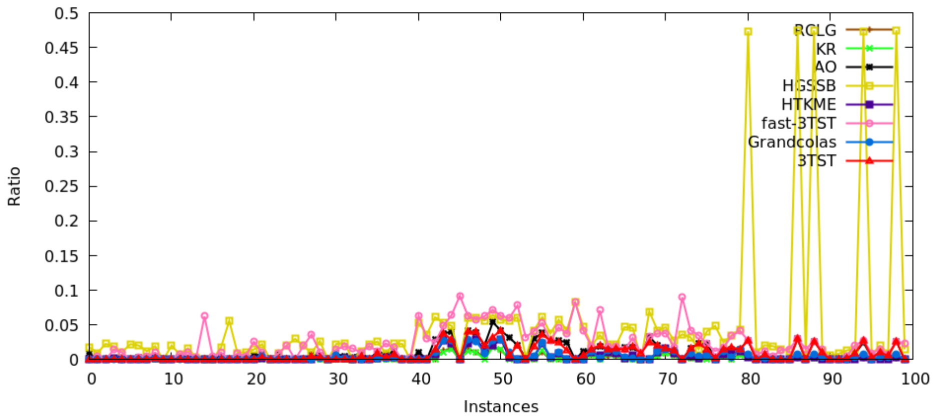


Figure 9: I640: Show the ratio obtained by each solver on each instance of the I640 instances data set. Instances are sorted by increasing number of vertices.

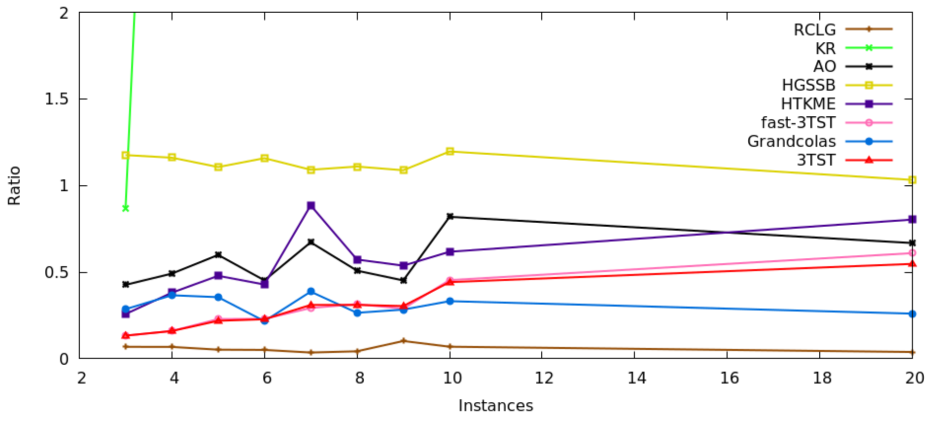


Figure 10: d-regular: Show the ratio obtained by each solver on d-regular random graph. Show the evolution of the ratio with respect to increasing values of d. Starting from $d = 4$, the ratio of the KR solver is around 10.

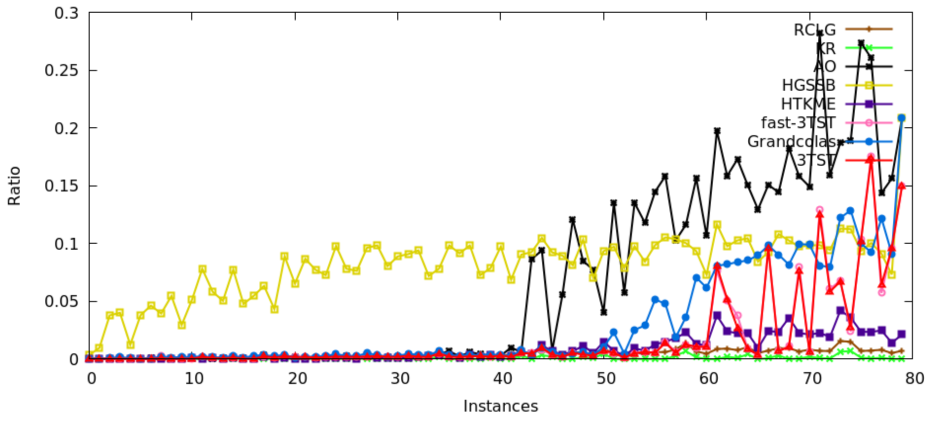


Figure 11: City road networks: Show the ratio obtained by each solver on each instance of the City road networks data set. Instances are sorted by increasing number of vertices.

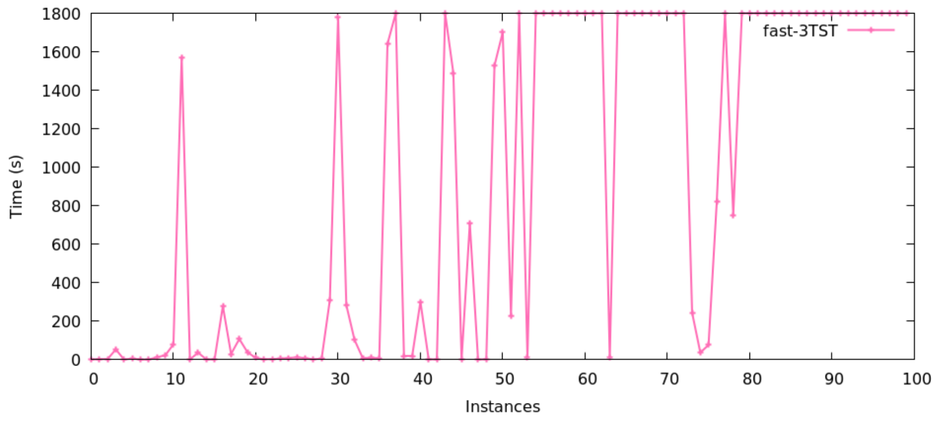


Figure 12: PACE Challenge: Show the running time of FAST-3TST on each instance of the PACE Challenge data set. Instances are sorted by increasing number of vertices.

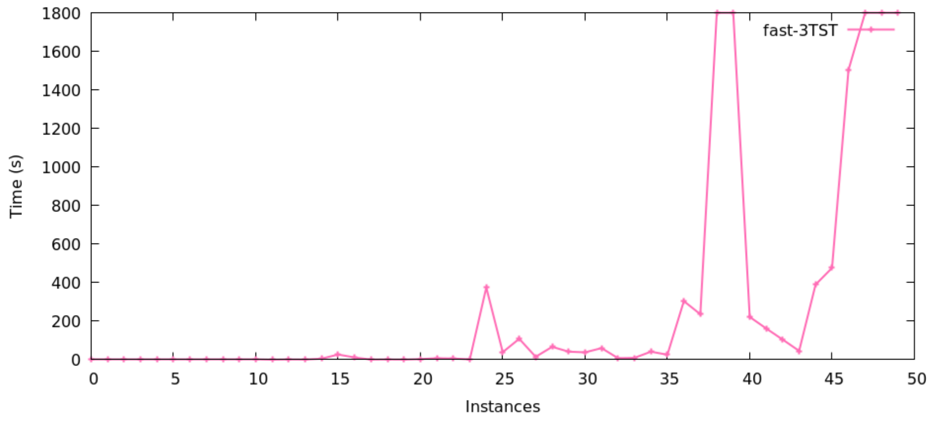


Figure 13: PUC: Show the running time of FAST-3TST on each instance of the PUC data set. Instances are sorted by increasing number of vertices.

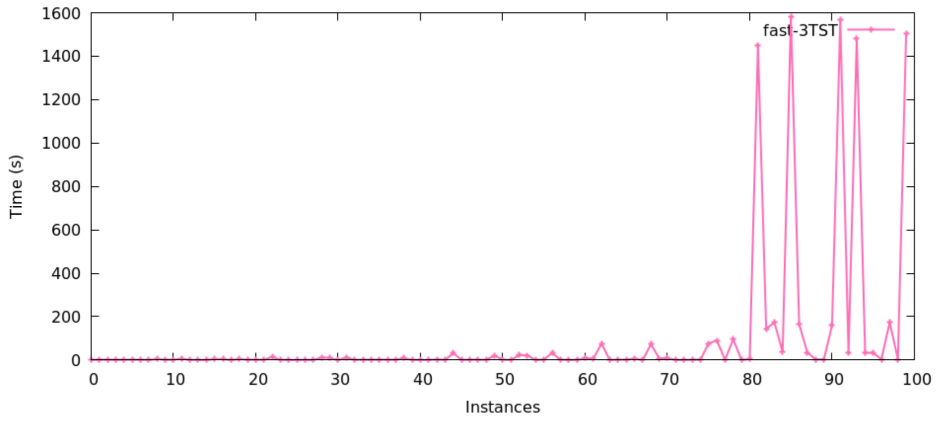


Figure 14: I640: Show the running time of FAST-3TST on each instance of the I640 data set. Instances are sorted by increasing number of vertices.

Chapter 10

On the Complexity of Intersection Non-Emptiness for Star-Free Language Classes

Emmanuel Arrighi, Henning Fernau, Stefan Hoffmann, Markus Holzer, Ismaël Jecker, Mateus de Oliveira Oliveira, and Petra Wolf.

An extended abstract appeared in the proceedings of 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15–17, 2021, Virtual Conference:
Leibniz International Proceedings in Informatics (LIPIcs) 213 (2021) pp. 34:1 — 34:15.
DOI: 10.4230/LIPIcs.FSTTCS.2021.34

This work started at the Schloss Dagstuhl Event 20483 *Moderne Aspekte der Komplexitätstheorie in der Automatentheorie* which was co-organized by Petra Wolf, Henning Fernau, and Volker Diekert see `https://www.dagstuhl.de/20483`.

On the Complexity of Intersection Non-Emptiness for Star-Free Language Classes

Emmanuel Arrighi[*1], Henning Fernau[†2], Stefan Hoffmann[2], Markus Holzer[3], Ismaël Jecker[‡4], Mateus de Oliveira Oliveira[§1], and Petra Wolf[¶2]

[1]University of Bergen, Norway

[2]Universität Trier, Germany

[3]Universität Gießen, Germany

[4]Institute of Science and Technology, Klosterneuburg, Austria

Abstract

In the Intersection Non-emptiness problem, we are given a list of finite automata $A_1, A_2, \ldots, A_m$ over a common alphabet Σ as input, and the goal is to determine whether some string $w \in \Sigma^*$ lies in the intersection of the languages accepted by the automata in the list. We analyze the complexity of the Intersection Non-emptiness problem under the promise that all input automata accept a language in some level of the dot-depth hierarchy, or some level of the Straubing-Thérien hierarchy. Automata accepting languages from the lowest levels of these hierarchies arise naturally in the context of model checking. We identify a dichotomy in the dot-depth hierarchy by showing that the problem is already NP-complete when all input automata accept languages of the levels $\mathcal{B}_0$ or $\mathcal{B}_{1/2}$ and already PSPACE-hard when all automata accept a language from the level $\mathcal{B}_1$. Conversely, we identify a tetrachotomy in the Straubing-Thérien hierarchy. More

[*]The author was supported by research Council of Norway (no. 274526), and part of a collaboration supported by IS-DAAD (no. 309319)

[†]The author was part of a collaboration supported by DAAD PPP (no. 57525246)

[‡]The author was supported by Marie Skłodowska-Curie Grant Agreement no. 754411

[§]The author was supported by research Council of Norway (no. 288761) and part of a collaboration supported by IS-DAAD (no. 309319)

[¶]The author was supported by DFG-funded project FE560/9-1 and part of a collaboration supported by DAAD PPP (no. 57525246)

precisely, we show that the problem is in AC^0 when restricted to level $\mathcal{L}_0$; complete for L or NL, depending on the input representation, when restricted to languages in the level $\mathcal{L}_{1/2}$; NP-complete when the input is given as DFAs accepting a language in $\mathcal{L}_1$ or $\mathcal{L}_{3/2}$; and finally, PSPACE-complete when the input automata accept languages in level $\mathcal{L}_2$ or higher. Moreover, we show that the proof technique used to show containment in NP for DFAs accepting languages in $\mathcal{L}_1$ or $\mathcal{L}_{3/2}$ does not generalize to the context of NFAs. To prove this, we identify a family of languages that provide an exponential separation between the state complexity of general NFAs and that of partially ordered NFAs. To the best of our knowledge, this is the first superpolynomial separation between these two models of computation.

1 Introduction

The Intersection Non-emptiness problem for finite automata is one of the most fundamental and well studied problems in the interplay between algorithms, complexity theory, and automata theory [Kozen, 1977, Kasai and Iwata, 1985, Lange and Rossmanith, 1992, Wareham, 2000, Karakostas et al., 2003, Wehar, 2014, Fernau and Krebs, 2017, Wehar, 2016]. Given a list $A_1, A_2, \ldots, A_m$ of finite automata over a common alphabet Σ, the goal is to determine whether there is a string $w \in \Sigma^*$ that is accepted by each of the automata in the list. This problem is PSPACE-complete when no restrictions are imposed [Kozen, 1977], and becomes NP-complete when the input automata accept unary languages (implicitly contained already in [Stockmeyer and Meyer, 1973] and studied in detail in [Morawietz et al., 2020]) or finite languages [Rampersad and Shallit, 2010].

In this work, we analyze the complexity of the Intersection Non-emptiness problem under the assumption that the languages accepted by the input automata belong to a given level of the Straubing-Thérien hierarchy [Place and Zeitoun, 2019, Straubing, 1981, Straubing, 1985, Thérien, 1981] or to some level of the Cohen-Brzozowski dot-depth hierarchy [Brzozowski, 1976, Cohen and Brzozowski, 1971, Place and Zeitoun, 2019]. Somehow, these languages are severely restricted, in the sense that both hierarchies, which are infinite, are entirely contained in the class of star-free languages, a class of languages that can be represented by expressions that use union, concatenation, and complementation, but *no* Kleene star operation [Brzozowski, 1976, Brzozowski and Knast, 1978, Place and Zeitoun, 2019]. Yet, languages belonging to fixed levels of either hierarchy may already be very difficult to characterize, in the sense that the very problem of deciding whether the language accepted by a given finite automaton belongs to a given full level or half-level k of either hierarchy is open, except for a few values of k [Almeida and Klíma, 2010, Glaßer and Schmitz, 2001, Glaßer and Schmitz, 2000, Place and Zeitoun, 2019]. It is worth noting that while the problem of determining whether a given automaton accepts a language in a certain level of either the dot-depth or of the Straubing-Thérien hierarchy is computationally hard (Theorem 1), automata accepting languages in lower levels of these hierarchies arise naturally in a variety of applications such as model checking where the INTERSECTION NON-EMPTINESS problem is of fundamental relevance [Abdulla, 2012, Bouajjani et al., 2000, Bouajjani et al., 2007].

An interesting question to consider is how the complexity of the INTERSECTION NON-EMPTINESS problem changes as we move up in the levels of the Straubing-Thérien hierarchy or in the levels of the dot-depth hierarchy. In particular, does the complexity of this problem changes gradually, as we increase the complexity of the input languages? In this work, we show that this is actually not the case, and that the complexity landscape for the INTERSECTION NON-EMPTINESS problem is already determined by the very first levels of either hierarchy (see Figure 1). Our first main result states that the INTERSECTION NON-EMPTINESS problem for NFAs and DFAs accepting languages from the level 1/2 of the Straubing-Thérien hierarchy are NL-complete and L-complete, respectively, under AC^0 reductions (Theorem 3). Additionally, this completeness result holds even in the case of unary languages. To prove hardness for NL and L, respectively, we will use a simple reduction from the reachability problem for DAGs and for directed trees, respectively. Nevertheless, the proof of containment in NL and in L, respectively, will require a new insight that may be of independent interest. More precisely, we will use a characterization of languages in the level 1/2 of the Straubing-Thérien hierarchy as shuffle ideals to show that the INTERSECTION NON-EMPTINESS problem can be reduced to CONCATENATION NON-EMPTINESS (Lemma 2). This allows us to decide INTERSECTION NON-EMPTINESS by analyzing each finite automaton given at the input individually. It is worth mentioning that this result is optimal in the sense that the problem becomes NP-hard even if we allow a single DFA to accept a language from $\mathcal{L}_1$, and require all the others to accept languages from $\mathcal{L}_{1/2}$ (Theorem 5).

Subsequently, we analyze the complexity of INTERSECTION NON-EMPTINESS when all input automata are assumed to accept languages from one of the levels of $\mathcal{B}_0$ or $\mathcal{B}_{1/2}$ of the dot-depth hierarchy, or from the levels $\mathcal{L}_1$ or $\mathcal{L}_{3/2}$ of the Straubing-Thérien hierarchy. It is worth noting that NP-hardness follows straightforwardly from the fact that INTERSECTION NON-EMPTINESS for DFAs accepting finite languages is already NP-hard [Rampersad and Shallit, 2010]. Containment in NP, on the other hand, is a more delicate issue, and here the representation of the input automaton plays an important role. A characterization of languages in $\mathcal{L}_{3/2}$ in terms of languages accepted by partially ordered NFAs [Schwentick et al., 2001] is crucial for us, combined with the fact that INTERSECTION NON-EMPTINESS when the input is given by such automata is NP-complete [Masopust and Thomazo, 2015]. Intuitively, the proof in [Masopust and

Thomazo, 2015] follows by showing that the minimum length of a word in the intersection of languages in the level 3/2 of the Straubing-Thérien hierarchy is bounded by a polynomial on the sizes of the minimum partially ordered NFAs accepting these languages. To prove that INTERSECTION NON-EMPTINESS is in NP when the input automata are given as DFAs, we prove a new result establishing that the number of Myhill-Nerode equivalence classes in a language in the level $\mathcal{L}_{3/2}$ is at least as large as the number of states in a minimum partially ordered automaton representing the same language (Lemma 5).

Interestingly, we show that the proof technique used to prove this last result does not generalize to the context of NFAs. To prove this, we carefully design a sequence $(L_n)_{n\in\mathbb{N}_{\geq 1}}$ of languages over a binary alphabet such that for every $n \in \mathbb{N}_{\geq 1}$, the language L_n can be accepted by an NFA of size n, but any partially ordered NFA accepting L_n has size $2^{\Omega(\sqrt{n})}$. This lower bound is ensured by the fact that the syntactic monoid of L_n has many $\mathcal{J}$-factors. Our construction is inspired by a technique introduced by Klein and Zimmermann, in a completely different context, to prove lower bounds on the amount of look-ahead necessary to win infinite games with delay [Klein and Zimmermann, 2016]. To the best of our knowledge, this is the first exponential separation between the state complexity of general NFAs and that of partially ordered NFAs. While this result does not exclude the possibility that INTERSECTION NON-EMPTINESS for languages in $\mathcal{L}_{3/2}$ represented by general NFAs is in NP, it gives some indication that proving such a containment requires substantially new techniques.

Finally, we show that INTERSECTION NON-EMPTINESS for both DFAs and for NFAs is already PSPACE-complete if all accepting languages are from the level $\mathcal{B}_1$ of the dot-depth hierarchy or from the level $\mathcal{L}_2$ of the Straubing-Thérien hierarchy. We can adapt Kozen's classical PSPACE-completeness proof by using the complement of languages introduced in [Masopust and Krötzsch, 2021] in the study of partially ordered automata. Since the languages in [Masopust and Krötzsch, 2021] belong to $\mathcal{L}_{3/2}$, their complement belong to $\mathcal{L}_2$ (and to $\mathcal{B}_1$), and therefore, the proof follows.

Acknowledgment We like to thank Lukas Fleischer and Michael Wehar for our discussions. This work started at the Schloss Dagstuhl Event 20483 *Moderne Aspekte der Komplexitätstheorie in der Automatentheorie* `https://www.dagstuhl.de/20483`.

2 Preliminaries

We let $\mathbb{N}_{\geq k}$ denote the set of natural numbers greater or equal than k.

We assume the reader to be familiar with the basics in computational complexity theory [Papadimitriou, 1994]. In particular, we recall the inclusion chain: $\mathsf{AC}^0 \subset \mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$. Let AC^0 (NC^1, respectively) refer to the class of problems accepted by Turing machines with a bounded (unbounded, respectively) number of alternations in logarithmic time; alternatively one can define these classes by uniform Boolean circuits. Here, L (NL, respectively) refers to the class of problems that are accepted by deterministic (nondeterministic, respectively) Turing machines with logarithmic space, P (NP, respectively) denotes the class of problems solvable by deterministic (nondeterministic, respectively) Turing machines in polynomial time, and PSPACE refers to the class of languages accepted by deterministic or nondeterministic Turing machines in polynomial space [Savitch, 1970]. Completeness and hardness are always meant with respect to deterministic logspace many-one reductions unless otherwise stated. We will also consider the parameterized class XP of problems that can be solved in time $n^{f(k)}$, where n is the size of the input, k is a parameter, and f is a computable function [Flum and Grohe, 2006].

We mostly consider *nondeterministic finite automata* (NFAs). An NFA A is a tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite *state set* with the *start state* $q_0 \in Q$, the *alphabet* Σ is a finite set of input symbols, and $F \subseteq Q$ is the *final state set*. The *transition function* $\delta : Q \times \Sigma \to 2^Q$ extends to words from Σ^* as usual. Here, 2^Q denotes the powerset of Q. By $L(A) = \{ w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset \}$, we denote the *language accepted by* A. The NFA A is a *deterministic finite automaton* (DFA) if $|\delta(q, a)| = 1$ for every $q \in Q$ and $a \in \Sigma$. Then, we simply write $\delta(q, a) = p$ instead of $\delta(q, a) = \{p\}$. If $|\Sigma| = 1$, we call A a *unary* automaton.

We study Intersection Non-emptiness problems and their complexity. For finite automata, this problem is defined as follows:

- *Input*: Finite automata $A_i = (Q_i, \Sigma, \delta_i, q_{(0,i)}, F_i)$, for $1 \leq i \leq m$.
- *Question*: Is there a word w that is accepted by all A_i, i.e., is $\bigcap_{i=1}^{m} L(A_i) \neq \emptyset$?

Observe that the automata have a common input alphabet. Note that the complexity of the non-emptiness problem for finite automata of a certain type is a lower bound for the Intersection Non-emptiness for this particular type of automata. Throughout the paper we are mostly interested in the complexity of the Intersection Non-emptiness problem for finite state devices whose languages are contained in a particular language class.

We study the computational complexity of the intersection non-emptiness problem for languages from the classes of the Straubing-Thérien [Straubing, 1981,Thérien, 1981] and

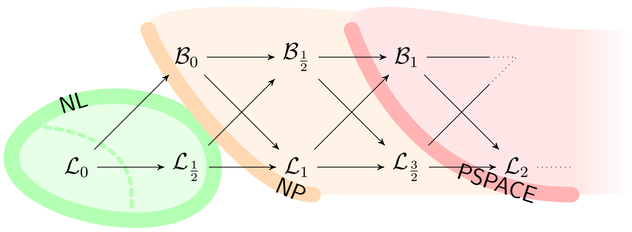


Figure 1: Straubing-Thérien and dot-depth hierarchies: the INTERSECTION NON-EMPTINESS status.

Cohen-Brzozowski's dot-depth hierarchy [Cohen and Brzozowski, 1971]. Both hierarchies are concatenation hierarchies that are defined by alternating the use of polynomial and Boolean closures. Let's be more specific. Let Σ be a finite alphabet. A language $L \subseteq \Sigma^*$ is a *marked product* of the languages $L_0, L_1, \ldots, L_k$, if $L = L_0 a_1 L_1 \cdots a_k L_k$, where the a_i's are letters. For a class of languages $\mathcal{M}$, the *polynomial closure* of $\mathcal{M}$ is the set of languages that are finite unions of marked product of languages from $\mathcal{M}$.

The concatenation hierarchy of basis $\mathcal{M}$ (a class of languages) is defined as follows (also refer to [Pin, 1998]): Level 0 is $\mathcal{M}$, i.e., $\mathcal{M}_0 = \mathcal{M}$ and, for each $n \geq 0$,

1. $\mathcal{M}_{n+1/2}$, that is, level $n + 1/2$, is the polynomial closure of level n and
2. $\mathcal{M}_{n+1}$, that is, level $n + 1$, is the Boolean closure of level $n + 1/2$.

The basis of the dot-depth hierarchy is the class of all finite and co-finite languages[1] and their classes are referred to as $\mathcal{B}_n$ ($\mathcal{B}_{n+1/2}$, respectively), while the basis of the Straubing-Thérien hierarchy is the class of languages that contains only the empty set and Σ^* and their classes are denoted by $\mathcal{L}_n$ ($\mathcal{L}_{n+1/2}$, respectively). Their inclusion relation is given by

$$\mathcal{B}_{n+1/2} \subseteq \mathcal{B}_{n+1} \subseteq \mathcal{B}_{n+3/2} \quad \text{and} \quad \mathcal{L}_{n+1/2} \subseteq \mathcal{L}_{n+1} \subseteq \mathcal{L}_{n+3/2},$$

for $n \geq 0$, and

$$\mathcal{L}_{n-1/2} \subseteq \mathcal{B}_{n-1/2} \subseteq \mathcal{L}_{n+1/2} \quad \text{and} \quad \mathcal{L}_n \subseteq \mathcal{B}_n \subseteq \mathcal{L}_{n+1},$$

for $n \geq 1$. In particular, $\mathcal{L}_0 \subseteq \mathcal{B}_0$, $\mathcal{B}_0 \subseteq \mathcal{B}_{1/2}$, and $\mathcal{L}_0 \subseteq \mathcal{L}_{1/2}$. Both hierarchies are infinite for alphabets of at least two letters and completely exhaust the class of star-free languages, which can be described by expressions that use union, concatenation, and complementation, but *no* Kleene star operation. For singleton letter alphabets, both hierarchies collapse to $\mathcal{B}_0$ and $\mathcal{L}_1$, respectively. Next, we describe the first few levels of each of these hierarchies:

[1]The dot-depth hierarchy, apart from level $\mathcal{B}_0$, coincides with the concatenation hierarchy starting with the language class $\{\emptyset, \{\lambda\}, \Sigma^+, \Sigma^*\}$.

Straubing-Thérien hierarchy: A language of Σ^* is of level 0 if and only if it is empty or equal to Σ^*. The languages of level 1/2 are exactly those languages that are a finite (possibly empty) union of languages of the form $\Sigma^* a_1 \Sigma^* a_2 \cdots a_k \Sigma^*$, where the a_i's are letters from Σ. The languages of level 1 are finite Boolean combinations of languages of the form $\Sigma^* a_1 \Sigma^* a_2 \cdots a_k \Sigma^*$, where the a_i's are letters. These languages are also called *piecewise* testable languages. In particular, all finite and co-finite languages are of level 1. Finally, the languages of level 3/2 of Σ^* are the finite unions of languages of the form $\Sigma_0^* a_1 \Sigma_1^* a_2 \cdots a_k \Sigma_k^*$, where the a_i's are letters from Σ and the Σ_i are subsets of Σ.

Dot-depth hierarchy: A language of Σ^* is of dot-depth (level) 0 if and only if it is finite or co-finite. The languages of dot-depth 1/2 are exactly those languages that are a finite union of languages of the form $u_0 \Sigma^* u_1 \Sigma^* u_2 \cdots u_{k-1} \Sigma^* u_k$, where $k \geq 0$ and the u_i's are words from Σ^*. The languages of dot-depth 1 are finite Boolean combinations of languages of the form $u_0 \Sigma^* u_1 \Sigma^* u_2 \cdots u_{k-1} \Sigma^* u_k$, where $k \geq 0$ and the u_i's are words from Σ^*.

It is worth mentioning that in [Schwentick et al., 2001] it was shown that partially ordered NFAs (with multiple initial states) characterize the class $\mathcal{L}_{3/2}$, while partially ordered DFAs characterize the class of $\mathcal{R}$-trivial languages [Brzozowski and Fich, 1980], a class that is strictly in between $\mathcal{L}_1$ and $\mathcal{L}_{3/2}$. For an automaton A with input alphabet Σ, a state q is reachable from a state p, written $p \leq q$, if there is a word $w \in \Sigma^*$ such that $q \in \delta(p, w)$. An automaton is partially ordered if $\leq$ is a partial order. Partially ordered automata are sometimes also called acyclic or weakly acyclic automata. We refer to a partially ordered NFA (DFA, respectively) as poNFA (poDFA, respectively).

The fact that some of our results have a promise looks a bit technical, but the following result implies that we cannot get rid of this condition in general. To this end, we study, for a language class $\mathcal{L}$, the following question of $\mathcal{L}$-Membership.

- *Input*: A finite automaton A.
- *Question*: Is $L(A) \in \mathcal{L}$?

Theorem 1. *For each level $\mathcal{L}$ of the Straubing-Thérien or the dot-depth hierarchies, the $\mathcal{L}$-*Membership *problem for NFAs is* PSPACE*-hard, even when restricted to binary alphabets.*

Proof. For the PSPACE-hardness, note that each of the classes contains $\{0,1\}^*$ and is closed under quotients, since each class is a positive variety. As Non-universality is PSPACE-hard for NFAs, we can apply Theorem 3.1.1 of [Hunt III and Rosenkrantz, 1978], first reducing regular expressions to NFAs. □

For some of the lower levels of the hierarchies, we also have containment in PSPACE, but in general, this is unknown, as it connects to the famous open problem if, for instance, $\mathcal{L}$-Membership is decidable for $\mathcal{L} = \mathcal{L}_3$; see [Masopust, 2018, Place and Zeitoun, 2019] for an overview on the decidability status of these questions. Checking for $\mathcal{L}_0$ up to $\mathcal{L}_2$ and $\mathcal{B}_0$ up to $\mathcal{B}_1$ containment for DFAs can be done in NL and is also complete for this class by ideas similar to the ones used in [Cho and Huynh, 1991].

3 Inside Logspace

A language of Σ^* belongs to level 0 of the Straubing-Thérien hierarchy if and only if it is empty or Σ^*. The Intersection Non-emptiness problem for language from this language family is not entirely trivial, because we have to check for emptiness. Since by our problem definition the property of a language being a member of level 0 is a promise, we can do the emptiness check within AC^0, since we only have to verify whether the empty word belongs to the language L specified by the automaton. In case $\varepsilon \in L$, then $L = \Sigma^*$; otherwise $L = \emptyset$. Since in the definition of finite state devices we do not allow for ε-transitions, we thus only have to check whether the initial state is also an accepting one. Therefore, we obtain:

Theorem 2. *The* Intersection Non-emptiness *problem for DFAs or NFAs accepting languages from $\mathcal{L}_0$ belongs to AC^0.*

For the languages of level $\mathcal{L}_{1/2}$ we find the following completeness result.

Theorem 3. *The* Intersection Non-emptiness *problem for NFAs accepting languages from $\mathcal{L}_{1/2}$ is* NL*-complete. Moreover, the problem remains* NL*-hard even if we restrict the input to NFAs over a unary alphabet. If the input instance contains only DFAs, the problem becomes* L*-complete (under weak reductions*[2]*).*

Hardness is shown by standard reductions from variants of graph accessibility [Hartmanis et al., 1978, Sudborough, 1975].

Lemma 1. *The* Intersection Non-emptiness *problem for NFAs over unary alphabet accepting languages from $\mathcal{L}_{1/2}$ is* NL*-hard. If the input instance contains only DFAs, the problem becomes* L*-hard under weak reductions.*

[2]Some form of AC^0 reducibility can be employed.

Proof. The NL-complete graph accessibility problem 2-GAP [Sudborough, 1975] is defined as follows: given a directed graph $G = (V, E)$ with outdegree (at most) two and two vertices s and t. Is there a path linking s and t in G? The problem remains NL-complete if the outdegree of every vertex of G is exactly two and if the graph is ordered, that is, if $(i, j) \in E$, then $i < j$ must be satisfied. The complexity of the reachability problem drops to L-completeness, if one considers the restriction that the outdegree is at most one. In this case the problem is referred to as 1-GAP [Hartmanis et al., 1978].

First we consider the Intersection Non-emptiness problem for NFAs. The NL-hardness is seen as follows: let $G = (V, E)$ and $s, t \in V$ be an ordered 2-GAP instance. Without loss of generality, we assume that $V = \{1, 2, \ldots, n\}$, the source vertex $s = 1$, and the target vertex $t = n$. From G we construct a unary NFA $A = (V, \{a\}, \delta, 1, n)$, where $\delta(i, a) = \{\, j \mid (i, j) \in E \,\} \cup \{i\}$. The 2-GAP instance has a solution if and only if the language accepted by A is non-empty. Moreover, by construction the automaton accepts a language of level $1/2$, because (i) the NFA without a-self-loops is acyclic, since G is ordered, and thus does not contain any large cycles and (ii) all states do have self-loops.

Finally, we concentrate on the L-hardness of the Intersection Non-emptiness problem for DFAs. Here we use the 1-GAP variant to prove our result. Let $G = (V, E)$ and $s, t \in V$ be a 1-GAP instance, where we can assume that $V = \{1, 2, \ldots, n\}$, $s = 1$, and $t = n$. From G we construct a unary DFA $A = (V, \{a\}, \delta, 1, n)$ with $\delta(i, a) = j$, for $(i, j) \in E$ and $1 \leq i < n$, and $\delta(n, a) = n$. By construction the DFA A accepts either the empty language or a unary language where all words are at least of a certain length. In both cases $L(A)$ is a language from level $1/2$ of the Straubing-Thérien hierarchy. Moreover, it is easy to see that there is a path in G linking s and t if and only if $L(A) \neq \emptyset$. □

It remains to show containment in logspace. To this end, we utilize an alternative characterization of the languages of level $1/2$ of the Straubing-Thérien hierarchy as exactly those languages that are shuffle ideals. A language L is a *shuffle ideal* if, for every word $w \in L$ and $v \in \Sigma^*$, the set $w \sqcup\!\sqcup\, v$ is contained in L, where $w \sqcup\!\sqcup\, v := \{\, w_0 v_0 w_1 v_1 \ldots w_k v_k \mid w = w_0 w_1 \ldots w_k \text{ and } v = v_0 v_1 \ldots v_k \text{ with } w_i, v_i \in \Sigma^*, \text{ for } 0 \leq i \leq k \,\}$. The operation $\sqcup\!\sqcup$ naturally generalizes to sets. For the level $\mathcal{L}_{1/2}$, we find the following situation.

Lemma 2. *Let $m \geq 1$ and languages $L_i \subseteq \Sigma^*$, for $1 \leq i \leq m$, be shuffle ideals, i.e., they belong to $\mathcal{L}_{1/2}$. Then, $\bigcap_{i=1}^{m} L_i \neq \emptyset$ iff the shuffle ideal $L_1 L_2 \cdots L_m \neq \emptyset$ iff $L_i \neq \emptyset$ for every i with $1 \leq i \leq m$. Finally, $L_i \neq \emptyset$, for $1 \leq i \leq m$, iff $(a_1 a_2 \ldots a_k)^{\ell_i} \in L_i$, where $\Sigma = \{a_1, a_2, \ldots a_k\}$ and the shortest word in L_i is of length ℓ_i.*

Proof. The implication from left to right holds, because if $\bigcap_{i=1}^{m} L_i \neq \emptyset$, then there is a word w that belongs to all L_i, and hence the concatenation $L_1 L_2 \cdots L_m$ is nonempty, too. Since this argument has not used the prerequisite that the L_i's belong to the first half level of the Straubing-Thérien hierarchy, this implication does hold in general.

For the converse implication, recall that a language L of the first half level is a finite (possibly empty) union of languages of the form $\Sigma^* a_1 \Sigma^* a_2 \cdots a_k \Sigma^*$, where the a_i's are letters. Hence, whenever a word w belongs to L, any word of the form uwv with $u, v \in \Sigma^*$ is a member of L, too. Now assume that $L_1 L_2 \cdots L_m \neq \emptyset$, which can be witnessed by words $w_i \in L_i$, for $1 \leq i \leq m$. But then the word $w_1 w_2 \ldots w_m$ belongs to every L_i, by setting $u = w_1 w_2 \ldots w_{i-1}$ and $v = w_{i+1} w_{i+2} \ldots w_m$ and using the argument above. Therefore, the intersection of all L_i, i.e., the set $\bigcap_{i=1}^{m} L_i$, is nonempty, because of the word $w_1 w_2 \ldots w_m$.

The statement that $L_1 L_2 \cdots L_m$ is an ideal and that $L_1 L_2 \cdots L_m \neq \emptyset$ if and only if $L_i \neq \emptyset$, for every i with $1 \leq i \leq m$, is obvious.

For the last statement, assume $\Sigma = \{a_1, a_2 \ldots, a_k\}$. The implication from right to left is immediate, because if $(a_1 a_2 \ldots a_k)^{\ell_i} \in L_i$, for ℓ_i as specified above, then L_i is non-empty. Conversely, if L_i is non-empty, then there is a shortest word w of length ℓ_i that is contained in L_i. But then $(a_1 a_2 \ldots a_k)^{\ell_i}$ belongs to $w \sqcup\!\sqcup \Sigma^*$, which by assumption is a subset of the language L_i, since L_i is an ideal. Therefore, $L_i \neq \emptyset$ implies $(a_1 a_2 \ldots a_k)^{\ell_i} \in L_i$, which proves the stated claim. □

Now, we are ready to prove containment in logspace and thereby conclude the proof of Theorem 3.

Lemma 3. *The* Intersection Non-emptiness *problem for NFAs accepting languages from $\mathcal{L}_{1/2}$ belongs to* NL. *If the input instance contains only DFAs, the problem is in* L.

Proof. In order to solve the Intersection Non-emptiness problem for given finite automata $A_1, A_2, \ldots, A_m$ with a common input alphabet Σ, regardless of whether they are deterministic or nondeterministic, it suffices to check non-emptiness for all languages $L(A_i)$, for $1 \leq i \leq m$, in sequence, because of Lemma 2. To this end, membership of the words $(a_1 a_2 \ldots a_k)^{\ell_i}$ in L_i has to be tested, where ℓ_i is the length of the shortest word in L_i. Obviously, all ℓ_i are linearly bounded in the number of states of the appropriate finite automaton that accepts L_i. Hence, for NFAs as input instance, the test can be done on a nondeterministic logspace-bounded Turing machine, guessing the computations in the individual NFAs on the input word $(a_1 a_2 \ldots a_k)^{\ell_i}$. For DFAs as input instance, nondeterminism is not needed, so that the procedure can be implemented on a deterministic Turing machine. □

4 NP-Completeness

In contrast to the Straubing-Thérien hierarchy, the Intersection Non-emptiness problem for languages from the dot-depth hierarchy is already NP-hard in the lowest level $\mathcal{B}_0$. More precisely, Intersection Non-emptiness for finite languages is NP-hard [Rampersad and Shallit, 2010, Theorem 1] and $\mathcal{B}_0$ already contains all finite languages. Hence, the Intersection Non-emptiness problem for languages from the Straubing-Thérien hierarchy of level $\mathcal{L}_1$ and above is NP-hard, too. For the levels $\mathcal{B}_0$, $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, or $\mathcal{L}_{3/2}$, we give matching complexity upper bounds if the input are DFAs, yielding the first main result of this section proven in subsection 4.1.

Theorem 4. *The* Intersection Non-emptiness *problem for DFAs accepting languages from either $\mathcal{B}_0$, $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, or $\mathcal{L}_{3/2}$ is* NP*-complete. The same holds for poNFAs instead of DFAs. The results hold even for a binary alphabet.*

For the level $\mathcal{L}_1$ of the Straubing-Thérien hierarchy, we obtain with the next main theorem a stronger result. Recall that if all input DFAs accept languages from $\mathcal{L}_{1/2}$, the Intersection Non-emptiness problem is L-complete due to Lemmata 1 and 3.

Theorem 5. *The* Intersection Non-emptiness *problem for DFAs is* NP*-complete even if only one DFA accepts a language from $\mathcal{L}_1$ and all other DFAs accept languages from $\mathcal{L}_{1/2}$ and the alphabet is binary.*

The proof of this theorem will be given in subsection 4.2.

For the level $\mathcal{B}_0$, we obtain a complete picture of the complexity of the Intersection Non-emptiness problem, independent of structural properties of the input finite automata, i.e., we show that here the problem is NP-complete for general NFAs.

For the level $\mathcal{L}_{3/2}$, if the input NFA are from the class of poNFA, which characterize level $\mathcal{L}_{3/2}$, then the Intersection Non-emptiness problem is known to be NP-complete [Masopust and Krötzsch, 2021]. Recall that $\mathcal{L}_{3/2}$ contains the levels $\mathcal{B}_{1/2}$, and $\mathcal{L}_1$ and hence also languages from these classes can be represented by poNFAs. But if the input automata are given as NFAs without any structural property, then the precise complexity of Intersection Non-emptiness for $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, and $\mathcal{L}_{3/2}$ is an open problem and narrowed by NP-hardness and membership in PSPACE. We present a "No-Go-Theorem" by proving that for an NFA accepting a co-finite language, the smallest equivalent poNFA is exponentially larger in Subsection 4.3.

Theorem 6. *For every $n \in \mathbb{N}_{\geq 1}$, there exists a language $L_n \in \mathcal{B}_0$ on a binary alphabet such that L_n is recognized by an NFA of size $O(n^2)$, but the minimal poNFA recognizing L_n has more than 2^{n-1} states.*

While for NFAs the precise complexity for INTERSECTION NON-EMPTINESS of languages from $\mathcal{L}_1$ remains open, we can tackle this gap by narrowing the considered language class to *commutative* languages in level $\mathcal{L}_1$; recall that a language $L \subseteq \Sigma^*$ is *commutative* if, for any $a, b \in \Sigma$ and words $u, v \in \Sigma^*$, we have that $uabv \in L$ implies $ubav \in L$. We show that for DFAs, this restricted INTERSECTION NON-EMPTINESS problem remains NP-hard, in case the alphabet is unbounded. Concerning membership in NP, we show that even for NFAs, the INTERSECTION NON-EMPTINESS problem for *commutative* languages is contained in NP in general and in particular for commutative languages on each level. This generalizes the case of unary NFAs. Note that for commutative languages, the Straubing-Thérien hierarchy collapses at level $\mathcal{L}_{3/2}$. See Subsection 4.4 for the proofs.

Theorem 7. *The* INTERSECTION NON-EMPTINESS *problem*

- *is* NP*-hard for DFAs accepting* commutative *languages in* $\mathcal{L}_1$*, but*
- *is contained in* NP *for NFAs accepting* commutative *languages that might not be star-free.*

The proof of NP-hardness for commutative star-free languages in $\mathcal{L}_1$ requires an arbitrary alphabet. However, we show that INTERSECTION NON-EMPTINESS is contained in XP, with the size of the alphabet as the parameter, for specific forms of NFAs accepting commutative star-free languages, i.e., for fixed input alphabets, the INTERSECTION NON-EMPTINESS problem is solvable in polynomial time for this class of NFAs.

4.1 NP-Membership

Next, we focus on the NP-membership part of Theorem 4 and begin by proving that for $\mathcal{B}_0$, regardless of whether the input automata are NFAs or DFAs, the INTERSECTION NON-EMPTINESS problem is contained in NP and therefore NP-complete in combination with [Rampersad and Shallit, 2010].

Lemma 4. *The* INTERSECTION NON-EMPTINESS *problem for DFAs or NFAs all accepting languages from* $\mathcal{B}_0$ *is contained in* NP.

Proof. Let $A_1, A_2, \ldots, A_m$ be NFAs accepting languages from $\mathcal{B}_0$. If all NFAs accept co-finite languages, the intersection $\bigcap_{i=1}^{m} L(A_i)$ is non-empty. We can check deterministically if an NFA A_i, accepting a language from $\mathcal{B}_0$, is co-finite in polynomial time by using the promise of the language class, stating that $L(A_i)$ is either finite or co-finite. For that, we check whether A_i contains a cycle on an accepting path from the initial to some final state. If this is the case, we know that $L(A_i)$ is co-finite as it is not finite.

Otherwise, there is at least one NFA accepting a finite language, where the longest word is bounded by the number of states of this device. Hence, if $\bigcap_{i=1}^{m} L(A_i) \neq \emptyset$, there is a word w of length polynomial in the length of the input that witnesses this fact. Such a w can be nondeterministically guessed by a Turing machine checking membership of w in $L(A_i)$, for all NFAs A_i, in sequence. This shows containment in NP as desired. □

Notice that Masopust and Krötzsch have shown in [Masopust and Krötzsch, 2021] that Intersection Non-emptiness for poDFAs and for poNFAs is NP-complete. Also the unary case is discussed there, which can be solved in polynomial time. We cannot directly make use of these results, as we consider arbitrary NFAs or DFAs as inputs, only with the promise that they accept languages from a certain level of the studied hierarchies. In order to prove that for the levels $\mathcal{B}_0$, $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, and $\mathcal{L}_{3/2}$, the Intersection Non-emptiness problem for DFAs is contained in NP, it is sufficient to prove the claim for $\mathcal{L}_{3/2}$ as all other stated levels are contained in $\mathcal{L}_{3/2}$. We prove the latter statement by obtaining a bound, polynomial in the size of the largest DFA, on the length of a shortest word accepted by all DFAs. Therefore, we show that for a minimal poNFA A, the size of an equivalent DFA is lower-bounded by the size of A and use a result of [Masopust and Krötzsch, 2021] for poNFAs. They have shown that given poNFAs $A_1, A_2, \ldots, A_m$, if the intersection of these automata is non-empty, then there exists a word of size at most $\sum_{i\in\{1,\ldots,m\}} d_i$, where d_i is the *depth* of A_i [Masopust and Krötzsch, 2021, Theorem 3.3]. Here, the depth of A_i is the length of the longest path (without self-loops) in the state graph of A_i. This result implies that the Intersection Non-emptiness problem for poNFAs accepting languages from $\mathcal{L}_{3/2}$ is contained in NP. We will further use this result to show that the Intersection Non-emptiness problem for DFAs accepting languages from $\mathcal{L}_{3/2}$ is NP-complete. First, we show that the number of states in a minimal poNFA is at most the number of classes in the Myhill-Nerode equivalence relation.

Lemma 5. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a minimal poNFA. Then, $L({}_{q_1}A) \neq L({}_{q_2}A)$ for all states $q_1, q_2 \in Q$, where ${}_{q}A$ is defined as $(Q, \Sigma, \delta, q, F)$.*

Proof. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a minimal poNFA and $q_1, q_2 \in Q$ be two states. Suppose that $L({}_{q_1}A) = L({}_{q_2}A)$. We have two cases.

1. If q_1 and q_2 are pairwise not reachable from each other, then let $A' = (Q', \Sigma, \delta', q_0, F')$ be the NFA obtained from A, where q_1 and q_2 are merged into a new state $q_{1,2}$, so that $Q' = (Q \setminus \{q_1, q_2\}) \cup \{q_{1,2}\}$, $\delta'(q_{1,2}, a) = \delta(q_1, a) \cup \delta(q_2, a)$, for all $q \in Q'$, $q_{1,2} \in \delta'(q, a)$ if and only if $q_1 \in \delta(q, a)$ or $q_2 \in \delta(q, a)$, and $q_{1,2} \in F'$ if and only if $q_1 \in F$ or $q_2 \in F$. Automata A' is a partially ordered NFA. As q_1 and q_2 are not reachable one from the other, they are incomparable in the partial order relation

defined by A. Therefore, there is no state q such that $q_1 < q$ and $q < q_2$. One can check that $L(A') = L(A)$, which contradicts the minimality of A.

2. Otherwise, q_1 is reachable from q_2, or q_2 is reachable from q_1. Without loss of generality, we assume that q_2 is reachable from q_1. Let $A' = (Q', \Sigma, \delta', q_0, F')$ be the NFA obtained from A in two steps as described next. First, we remove all outgoing transitions from q_1 and then we merge q_1 and q_2 into a new state $q_{1,2}$ as done before. After removing all outgoing transitions from q_1, state q_2 is no longer reachable from q_1, therefore, as before, A' is a partially ordered NFA. Now we will prove that $L(A) = L(A')$.

   - Let $w \in L(A)$. Let ρ be an accepting run in A. If ρ does not contain q_1, then the run obtained by replacing every q_2 by $q_{1,2}$ is an accepting run in A'. If ρ contains q_1, then we split w into w_1 and w_2 such that $w = w_1w_2$ and w_1 is the shortest prefix of w such that, after reading w_1, we reach q_1 in ρ. Because we merged q_1 and q_2 into $q_{1,2}$, we have that $q_{1,2} \in \delta'(q_0, w_1)$ in A'. Because $L({}_{q_1}A) = L({}_{q_2}A)$, we have that $L({}_{q_1}A) = L({}_{q_2}A) = L({}_{q_{1,2}}A')$ and therefore $\delta'(q_{1,2}, w_2) \cap F' \neq \emptyset$. So, w is accepted by A'.
   - Conversely, let $w \in L(A')$. Let ρ be an accepting run in A'. If ρ does not contain $q_{1,2}$, then the same run is accepting in A, too. If ρ contains $q_{1,2}$, we split w into w_1 and w_2 such that $w = w_1w_2$, where w_1 is the shortest prefix of w such that, after reading w_1, we reach $q_{1,2}$ in ρ. Then, by definition of $q_{1,2}$, $\delta(q_0, w_1) \cap \{q_1, q_2\} \neq \emptyset$, and $\delta(q_1, w_2) \cap F \neq \emptyset$ iff $\delta(q_2, w_2) \cap F \neq \emptyset$ iff $\delta'(q_{1,2}, w_2) \neq \emptyset$. Therefore, $w \in L(A)$.

   This contradicts the minimality of A. □

Now, we can use the result from Masopust and Krötzsch to prove that the Intersection Non-emptiness problem for DFAs accepting languages in $\mathcal{L}_{3/2}$ is in NP.

Lemma 6. *The* Intersection Non-emptiness *problem for DFAs accepting languages from $\mathcal{L}_{3/2}$ belongs to* NP.

Proof. By Lemma 5, we have that the number of states in a minimal poNFA is at most the number of classes of the Myhill-Nerode equivalence relation. Hence, given a DFA accepting a language $L \in \mathcal{L}_{3/2}$, there exists a potentially smaller poNFA that recognizes L. By [Masopust and Krötzsch, 2021, Theorem 3.3], if the intersection is not empty, then there is a certificate of size polynomial in the sizes of the poNFAs . □

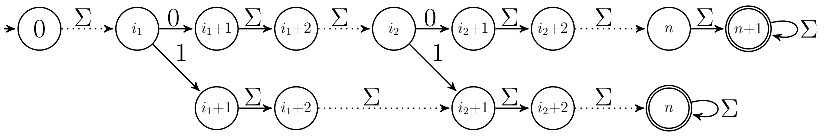


Figure 2: DFA A_{e_i} with $L(A_{e_i}) = \Sigma^{i_1} \cdot 1 \cdot \Sigma^{n-i_1-1} \cup \Sigma^{i_2} \cdot 1 \cdot \Sigma^{n-i_2-1} \cup \Sigma^{\geq n+1}$. A dotted arrow between some states j and j' represents a chain of length $j' - j$ with the same transition labels.

4.2 NP-Hardness

Recall that Intersection Non-emptiness for finite languages accepted by DFAs is already NP-complete by [Rampersad and Shallit, 2010, Theorem 1]. As the level $\mathcal{B}_0$ of the dot-depth hierarchy contains all finite language, the NP-hardness part of Theorem 4 follows directly from inclusion of language classes. Combining Lemma 6, and [Masopust and Krötzsch, 2021, Theorem 3.3] with the inclusion between levels in the Straubing-Thérien and the dot-depth hierarchy, we conclude the proof of Theorem 4.

Remark 1. Recall that the dot-depth hierarchy, apart form $\mathcal{B}_0$, coincides with the concatenation hierarchy starting with the language class $\{\emptyset, \{\lambda\}, \Sigma^+, \Sigma^*\}$. The Intersection Non-emptiness problem for DFAs or NFAs accepting only languages from $\{\emptyset, \{\lambda\}, \Sigma^+, \Sigma^*\}$ belongs to AC^0, by similar arguments as in the proof of Theorem 2.

We showed in Section 3 that Intersection Non-emptiness for DFAs, all accepting languages from $\mathcal{L}_{1/2}$, belongs to L. If we allow only one DFA to accept a language from $\mathcal{L}_1$, the problem becomes NP-hard. The statement also holds if the common alphabet is binary.

Theorem 5. *The* Intersection Non-emptiness *problem for DFAs is* NP*-complete even if only one DFA accepts a language from $\mathcal{L}_1$ and all other DFAs accept languages from $\mathcal{L}_{1/2}$ and the alphabet is binary.*

Proof sketch. The reduction is from Vertex Cover. Let $k \in \mathbb{N}_{\geq 0}$ and let $G = (V, E)$ be a graph with vertex set $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and edge set $E = \{e_0, e_1, \ldots, e_{m-1}\}$. The only words $w = a_0 a_1 \ldots a_\ell$ accepted by all DFAs will be of length exactly $n = \ell + 1$ and encode a vertex cover by: v_j is in the vertex cover if and only if $a_j = 1$. Therefore, we construct for each edge $e_i = \{v_{i_1}, v_{i_2}\} \in E$, with $i_1 < i_2$, a DFA A_{e_i}, as depicted in Figure 2, that accepts the language $L(A_{e_i}) = \Sigma^{i_1} \cdot 1 \cdot \Sigma^{n-i_1-1} \cup \Sigma^{i_2} \cdot 1 \cdot \Sigma^{n-i_2-1} \cup \Sigma^{\geq n+1}$. We show that $L(A_{e_i})$ is from $\mathcal{L}_{1/2}$, as it also accepts all words of length at least $n+1$. We further construct a DFA $A_{=n,\leq k}$ that accepts all words of length exactly n that contain at most k letters 1. The finite language $L(A_{=n,\leq k})$ is the only language from $\mathcal{L}_1$ in the instance. □

Proof. The NP-membership follows from Lemma 6 by inclusion of language classes. For the hardness, we give a reduction from the VERTEX COVER problem: given an undirected graph $G = (V, E)$ with vertex set V and edge set $E \subseteq V \times V$ and integer k. Is there a subset $S \subseteq V$ with $|S| \leq k$ and for all $e \in E$, $S \cap e \neq \emptyset$? If yes, we call S a *vertex cover* of G of size at most k.

Let $k \in \mathbb{N}_{\geq 0}$ and let $G = (V, E)$ be an undirected graph with vertex set $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and edge set $E = \{e_0, e_1, \ldots, e_{m-1}\}$. From (G, k) we construct $m + 1$ DFAs over the common alphabet $\Sigma = \{0, 1\}$. The input word for these automata will encode which vertices are in the vertex cover. Therefore, we assume a linear order on V indicated by the indices of the vertices. More precisely, a word accepted by all automata will have a 1 at position j if and only if the vertex v_j will be contained in the vertex cover S. For a word $w = a_0a_1 \ldots a_\ell$ with $a_j \in \Sigma$ for $0 \leq j \leq \ell$ we denote $w[j] = a_j$. We may call a word w of length n a *vertex cover* and say that the vertex cover covers an edge $e = \{v_{j_1}, v_{j_2}\}$ if $w[j_1] = 1$ or $w[j_2] = 1$.

For every edge $e_i = \{v_{i_1}, v_{i_2}\}$ in E with $i_1 < i_2$, we construct a DFA A_{e_i} as depicted in Figure 2 consisting of two chains, one of length $n + 1$ and one of length $n - (i_1 + 1)$ (The length of a chain is the number of transitions in the chain). The DFA is defined as $A_{e_i} = (Q, \Sigma, \delta, q^0, F)$ with state set $Q = \{\, q^j \mid 0 \leq j \leq n + 1 \,\} \cup \{\, q'^j \mid i_1 + 1 \leq j \leq n \,\}$ and final states $F = \{\, q^{n+1}, q'^n \,\}$. We first focus on the states $\{\, q^j \mid 0 \leq j \leq n + 1 \,\}$. The idea is that there, the first $n + 1$ states correspond to the sequence of vertices and reading a 1 at position j for which $v_j \in e_i$ will cause the automaton to switch to the chain consisting of states $\{\, q'^j \mid i_1 + 1 \leq j \leq n \,\}$. There, only one state is accepting namely the state that we reach after reading a vertex cover of length exactly n that satisfies the edge e_i. Note that the paths from q_0 to q'^n are one transition shorten than the path from q_0 to q^{n+1}. To be more formal, we define $\delta(q^{i_1}, 1) = q'^{i_1+1}$ and $\delta(q^{i_2}, 1) = q'^{i_2+1}$. All other transitions are leading to the next state in the corresponding chain. Formally, we define $\delta(q^{i_1}, 0) = q^{i+1}$ and $\delta(q^{i_2}, 0) = q^{i_2+1}$, and for all $0 \leq j \leq n$ with $j \notin \{i_1, i_2\}$, we define $\delta(q^j, \sigma) = q^{j+1}$, for both $\sigma \in \Sigma$, and for all $i + 1 \leq j \leq n - 1$, we define $\delta(q'^j, \sigma) = q'^{j+1}$. We conclude the definition of δ by defining self-loops for the two accepting states, i.e., we define $\delta(q^{n+1}, \sigma) = q^{n+1}$ and $\delta(q'^n, \sigma) = q'^n$ for both $\sigma \in \Sigma$. Clearly, A_{e_i} is deterministic and of size $\mathcal{O}(n)$.

Note that the only words of length *exactly* n that are accepted by A_{e_i} contain a 1 at position i_1 or position i_2 and therefore cover the edge e_i. All other words accepted by A_{e_i} are of length at least $n + 1$. More precisely A_{e_i} accepts *all* words which are of size at least $n + 1$. Hence, we can describe the language accepted by A_{e_i} as

$$L(A_{e_i}) = \Sigma^{i_1} \cdot 1 \cdot \Sigma^{n-i_1-1} \cup \Sigma^{i_2} \cdot 1 \cdot \Sigma^{n-i_2-1} \cup \Sigma^{\geq n+1}.$$

Consider a word $w \in L(A_{e_i})$ of length n. W.l.o.g., assume $w[i_1] = 1$. If we insert into w one letter somewhere before or after position i_1, then the size of w increases by 1 and hence w falls into the subset $\Sigma^{\geq n+1}$ of $L(A_{e_i})$. Hence, we can rewrite the language $L(A_{e_i})$ by the following equivalent expression.

$$L(A_{e_i}) = \Sigma^{i_1} \cdot \Sigma^* \cdot 1 \cdot \Sigma^{n-i_1-1} \cdot \Sigma^* \cup \Sigma^{i_2} \cdot \Sigma^* \cdot 1 \cdot \Sigma^{n-i_2-1} \cdot \Sigma^* \cup \Sigma^{n+1}\Sigma^*.$$

As we can rewrite a language of the form $\Sigma^\ell\Sigma^*$ equivalently as a union of languages of the form $\Sigma^* w_1 \Sigma^* w_2 \dots w_\ell \Sigma^*$ for $w_i \in \Sigma$, for $1 \leq i \leq \ell$, it is clear that $L(A_{e_i})$ is a language of level $\mathcal{L}_{1/2}$.

Next, we define a DFA $A_{=n,\leq k}$ which accepts the finite language of all binary words of length n which contain at most k appearances of the letter 1. We define $A_{=n,\leq k} = (\{q_i^j \mid 0 \leq i \leq n+1, 0 \leq j \leq k+1\}, \Sigma, \delta, q_0^0, \{q_n^j \mid j \leq k\})$. The state graph of $A_{=n,\leq k}$ is a (n,k)-grid graph, where each letter increases the x dimension represented by the subscript i up to the value $n+1$, and each letter that is a 1 increases the y dimension represented by the superscript j up to the value $k+1$. More formally, we define $\delta(q_i^j, 0) = q_{i+1}^j$, and $\delta(q_i^j, 1) = q_{i+1}^{j+1}$ for $0 \leq i \leq n$ and $0 \leq j \leq k$; and $\delta(q_i^j, \sigma) = q_i^j$ for $i = n+1$ or $j = k+1$. The size of $A_{=n,\leq k}$ is bounded by $\mathcal{O}(nk)$. For readability, we defined $A_{=n,\leq k}$ as a non-minimal DFA. As $L(A_{=n,\leq k})$ is finite, it is of level $\mathcal{B}_0 \subseteq \mathcal{L}_1$.

By the arguments discussed above, the set of words accepted by all of the automata $(A_{e_i})_{e_i \in E}$ and $A_{=n,\leq k}$ are of size exactly n and encode a vertex cover for G of size at most k. □

4.3 Large Partially Ordered NFAs

The results obtained in the last subsection left open the precise complexity membership of Intersection Non-emptiness in the case of input automata being NFAs without any structural properties for the levels $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, and $\mathcal{L}_{3/2}$. We devote this subsection to the proof of Theorem 6, showing that already for languages of $\mathcal{B}_0$ being accepted by an NFA, the size of an equivalent minimal poNFA can be exponential in the size of the NFA.

Theorem 6. *For every $n \in \mathbb{N}_{\geq 1}$, there exists a language $L_n \in \mathcal{B}_0$ on a binary alphabet such that L_n is recognized by an NFA of size $O(n^2)$, but the minimal poNFA recognizing L_n has more than 2^{n-1} states.*

Proof. While the statement requires languages over a binary alphabet, we begin by constructing an auxiliary family $(M_n)_{n \in \mathbb{N}_{\geq 1}}$ of languages over an unbounded alphabet.

For all $n \in \mathbb{N}_{\geq 1}$ we then define L_n by encoding M_n with a binary alphabet, and we prove three properties of these languages that directly imply the statement of the Theorem.

For every $n \in \mathbb{N}_{\geq 1}$, we define the languages M'_n and M''_n over the alphabet $\{1, 2, \dots, n\}$ as follows. The language M'_n contains all the words of odd length, and M''_n contains all the words in which there are two occurrences of some letter $i \in \{1, 2, \dots, n\}$ with only letters smaller than i appearing in between.[3] Formally,

$$M'_n = \{\, x \in \{1, 2, \dots, n\}^* \mid |x| \text{ is odd} \,\},$$
$$M''_n = \{\, xiyiz \in \{1, 2, \dots, n\}^* \mid i \in \{1, 2, \dots, n\}, y \in \{1, 2, \dots, i-1\}^* \,\}.$$

We then define M_n as the union $M'_n \cup M''_n$. Moreover, we define L_n by encoding M_n with the binary alphabet $\{a, b\}$: Let us consider the function $\phi_n : \{1, 2, \dots, n\}^* \to \{a, b\}^*$ defined by $\phi(i_1 i_2 \dots i_m) = a^{i_1} b^{n-i_1} a^{i_2} b^{n-i_2} \dots a^{i_m} b^{n-i_m}$. We set $L_n \subseteq \{a, b\}^*$ as the union of $\phi_n(M_n)$ with the language $\{a, b\}^* \setminus \phi(\{1, 2, \dots, n\}^*)$ containing all the words that are not a proper encoding of some word in $\{1, 2, \dots, n\}^*$.

The statement of the theorem immediately follows from the following claims.

1. The languages M_n and L_n are cofinite, thus they are in $\mathcal{B}_0$.
2. The languages M_n and L_n are recognized by NFAs of size $n + 4$, resp. $O(n^2)$.
3. Every poNFA recognizing either M_n or L_n has a size greater than 2^{n-1}.

Proof of Item 1. We begin by proving that M_n is cofinite. Note that, by itself, the language M'_n is not in $\mathcal{B}_0$, as it is not even star-free. We show that M''_n is cofinite, which directly implies that $M_n = M'_n \cup M''_n$ is also cofinite. This follows from the fact that every word $u \in \{1, 2, \dots, n\}^*$ satisfying $|u| \geq 2^n$ is in M''_n [Klein and Zimmermann, 2016]. This is easily proved by induction on n: If $n = 1$, we immediately get that $1^j \in M''_1$ for every $j \geq 2 = 2^1$: such a word contains two adjacent occurrences of 1. Now suppose that $n > 1$, and that the property holds for $n-1$. Every word $u \in \{1, 2, \dots, n\}^*$ satisfying $|u| \geq 2^n$ can be split into two parts u_0, u_1 such that $|u_0|, |u_1| \geq 2^{n-1}$. We consider two possible cases, and prove that $u \in M''_n$ in both of them.

1. If either u_0 or u_1 contains no occurrence of the letter n, then by the induction hypothesis, either $u_0 \in M''_{n-1}$ or $u_1 \in M''_{n-1}$, which directly implies that $u \in M''_n$.

[3]The languages $(M''_n)_{n \in \mathbb{N}_{\geq 1}}$ were previously studied in [Klein and Zimmermann, 2016] with a game-theoretic background. We also refer to [Naylor, 2011] for similar "fractal languages."

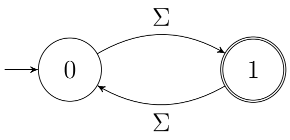


Figure 3: Automaton A' recognizing M'_n.

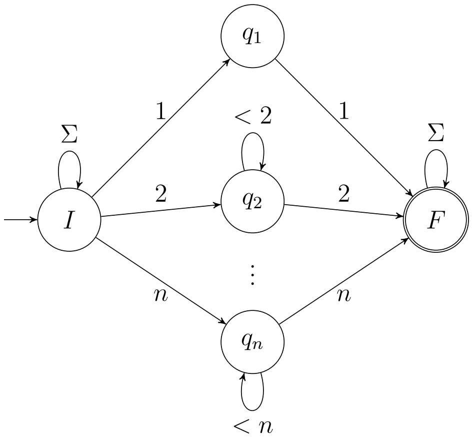


Figure 4: Automaton A'' recognizing M''_n.

2. If both u_0 and u_1 contain (at least) one occurrence of the letter n, then $u \in M''_n$ since it contains two occurrences of the letter n with only letters smaller than n appearing in between (the latter part trivially holds, as n is the largest letter).

Finally, we also get that L_n is cofinite: for all $u \in \{a,b\}^*$ satisfying $|u| \geq 2^n \cdot n$, either u is not a proper encoding of a word of $\{1,2,\ldots,n\}^*$, thus $u \in L_n$, or u encodes a word $v \in \{1,2,\ldots,n\}^*$ satisfying $|v| \geq 2^n$, hence $v \in M_n$, which again implies that $u \in L_n$. $\lhd$

Proof of Item 2. We first construct an NFA A of size $n+4$ recognizing $M_n = M'_n \cup M''_n$ as the disjoint union of an NFA A' (Figure 3) of size 2 recognizing M'_n and an NFA A'' (Figure 4) of size $n+2$ recognizing M''_n. The language M'_n of words of odd length is trivially recognized by an NFA of size 2, thus we only need to build an NFA $A'' = (Q, \{1,2,\ldots,n\}, \delta, q_I, \{q_F\})$ of size $n+2$ that recognizes M''_n. The state space Q is composed of the start state q_I, the single final state q_F, and n intermediate states $\{q_1, q_2, \ldots, q_n\}$. The NFA A'' behaves in three phases:

1. First, A'' loops over its start state until it non-deterministically guesses that it will read two copies of some $i \in \Sigma$ with smaller letters in between: $\delta(q_I, i) = \{q_I, q_i\}$ for all $i \in \Sigma$.

2. To check its guess, A'' loops in q_i while reading letters smaller than i until it reads a second i: $\delta(q_i, j) = \{q_i\}$ for all $j \in \{1, 2, \dots, i-1\}$ and $\delta(q_i, i) = \{q_F\}$.
3. The final state q_F is an accepting sink: $\delta(q_F, j) = \{q_F\}$ for all $j \in \Sigma$.

This definition guarantees that A'' accepts the language M''_n.

Finally, we build an NFA B of size $O(n^2)$ that recognizes L_n by following similar ideas. Once again, B is defined as the disjoint union of two NFAs B' and B'': The NFA B' uses $4n$ states to check that either the input is *not* a proper encoding, or the input encodes a word $u \in \{1, 2, \dots, n\}^*$ of odd length. Then, the NFA B'' with $O(n^2)$ states is obtained by adapting the NFA A'' to the encoding of the letters $\{1, 2, \dots, n\}$: we split each of the $2n$ intermediate transitions of A'' into n parts by adding $n-1$ states, and we add $2(n-1)$ states to each self-loop of A'' in order to check that the encoding of an adequate letter is read. $\lhd$

Proof of Item 3. It is sufficient to prove the result for M_n, as we can transform each poNFA $A = (Q, \{a, b\}, \delta_A, q_I, F)$ recognizing L_n into a poNFA $B = (Q, \{1, 2, \dots, n\}, \delta_B, q_I, F)$ recognizing M_n with the same set of states by setting $\delta_B(q, i) = \delta_A(q, a^i b^{n-i})$.

Note that, by itself, the language M''_n is recognized by the poNFA A of size $n+2$ defined in the proof of Item 2. Let A' be a poNFA recognizing M_n. To show that A' has more than 2^{n-1} states, we study its behavior on the *Zimin words*, defined as follows:

$$\text{Let } u_1 = 1 \text{ and } u_j = u_{j-1} j u_{j-1} \text{ for all } 1 < j \leq n.$$

For instance, $u_4 = 121312141213121$. It is known that $|u_j| = 2^j - 1$ and $u_j \notin M''_n$ for every $1 \leq j \leq n$ [Klein and Zimmermann, 2016]. These two properties are easily proved by induction on j: Trivially, u_1 is not in M''_1 and its size is $1 = 2^1 - 1$. Now suppose that $j > 1$ and that u_{j-1} satisfies both properties: $|u_{j-1}| = 2^{j-1} - 1$ and $u_{j-1} \notin M''_n$. The first property follows immediately from the induction hypothesis.

$$|u_j| = |u_{j-1} j u_{j-1}| = 2 \cdot |u_{j-1}| + 1 = 2 \cdot (2^{j-1} - 1) + 1 = 2^j - 1;$$

To prove the induction step for the second property, we suppose, towards building a contradiction, that $u_j \in M''_n$. Then u_j contains two occurrences of some letter $i \in \{1, 2, \dots, n\}$ with only letters smaller than i appearing in between. Since u_j contains only one occurrence of the letter j and no letter is greater than j, i is strictly smaller than j. Moreover, as only letters smaller than i (thus no j) can appear between these two occurrences, they both need to appear in one of the copies of u_{j-1}. Therefore u_{j-1}

is also in M''_n, which contradicts the induction hypothesis.

To conclude, remark that the word u_n is not in M''_n, but since $|u_n| = 2^n - 1$ is odd, it is in $M_n = L(A')$. Consider a sequence $\rho \in Q^*$ of states leading A' from its start state to a final state over the input u_n. Observe that the word u_n contains 2^{n-1} occurrences of the letter 1, and deleting (any) one of these occurrences results in a word of even length that is still not in M''_n, thus it is also not in $M_n = L(A')$. This proves that the sequence ρ cannot loop over any of the 1's in u_n. Moreover, as A' is partially ordered by assumption, once it leaves a state, it can never return to it. Therefore, ρ contains at least $2^{n-1} + 1$ distinct states while processing the 2^{n-1} occurrences of 1 in u_n, which shows that the automaton A' has more than 2^{n-1} many states. $\lhd$ $\square$

4.4 Commutative Star-Free Languages

In the case of commutative languages, we have a complete picture of the complexities for both hierarchies, even for arbitrary input NFAs. Observe, that commutative languages generalize unary languages, where it is known that for unary star-free languages both hierarchies collapse. For commutative star-free languages, a similar result holds, employing [Hoffmann, 2021, Prop. 28].

Theorem 8. *For* commutative star-free languages *the levels $\mathcal{L}_n$ of the Straubing-Thérien and $\mathcal{B}_n$ of the dot-depth hierarchy coincide for all full and half levels, except for $\mathcal{L}_0$ and $\mathcal{B}_0$. Moreover, the hierarchy collapses at level one.*

Proof. The strict inclusion $\mathcal{L}_0 \subset \mathcal{B}_0$ even in the commutative case is obvious. Since $\mathcal{L}_{1/2} \subseteq \mathcal{B}_{1/2}$ we only need to show the converse inclusion in the case of commutative languages. For the sake of notational simplicity, we shall give the proof only in a special case. Observe that, by commutativity, if $\Sigma^* ab \Sigma^* \subseteq L$, then $\Sigma^* a \Sigma^* b \Sigma^* \subseteq L$; moreover, $\Sigma^* ab \Sigma^* \subseteq \Sigma^* a \Sigma^* b \Sigma^*$. Using this idea repeatedly for marked products, as they describe languages from $\mathcal{B}_{1/2}$, we can write them as equivalent polynomials used for defining languages from $\mathcal{L}_{1/2}$.

It remains to show that every commutative star-free language is contained in $\mathcal{L}_1$. As shown in [Hoffmann, 2021, Prop. 28], every star-free commutative language can be written as a finite union of languages of the form $L = \texttt{perm}(u) \sqcup\!\sqcup \Gamma^*$ for some $u \in \Sigma^*$ and $\Gamma \subseteq \Sigma$. Here $\texttt{perm}(u) = \{\, w \in \Sigma^* \mid |u|_a = |w|_a \text{ for every } a \in \Sigma \,\}$, where $|w|_a$ is equal to the number of occurrences of a in w. Since $\texttt{perm}(u)$ is a finite language, clearly, language L is equal to the finite union of all $v \sqcup\!\sqcup \Gamma^*$ for $v \in \texttt{perm}(u)$, and thus belongs to $\mathcal{L}_{3/2}$, since $\Gamma \subseteq \Sigma$.

Now, note that $v \sqcup\!\sqcup \Sigma^* = \Sigma^* v_1 \Sigma^* \cdots \Sigma^* v_{|v|} \Sigma^*$, where $v = v_1 \cdots v_{|u|}$ with $v_i \in \Sigma$, is in level 1/2 of the Straubing-Thérien hierarchy. Further,

$$v \sqcup\!\sqcup \Gamma^* = (v \sqcup\!\sqcup \Sigma^*) \cap \overline{\bigcup_{a \in \Sigma \setminus \Gamma} \mathtt{perm}(va) \sqcup\!\sqcup \Sigma^*}.$$

Hence, we can conclude containment in $\mathcal{L}_1$. As we have shown earlier that for commutative languages $\mathcal{B}_{1/2} \subseteq \mathcal{L}_{1/2}$, we get $\mathcal{B}_1 \subseteq \mathcal{L}_1$ and hence L is also contained in $\mathcal{B}_1$, and both hierarchies collapse at level one for commutative languages. □

Next we will give the results, summarized in Theorem 7, for the case of the commutative (star-free) languages. The NP-hardness follows by a reduction from 3-CNF-SAT.

Lemma 7. *The* Intersection Non-emptiness *problem is* NP*-hard for DFAs accepting* commutative *languages in* $\mathcal{L}_1$.

Proof. The NP-complete 3-CNF-SAT problem is defined as follows: given a Boolean formula φ as a set of clauses $C = \{c_1, c_2, \ldots, c_m\}$ over a set of variables $V = \{x_1, x_2, \ldots, x_n\}$ such that $|c_i| \leq 3$ for $i \leq m$. Is there a variable assignment $\beta : V \to \{0, 1\}$ such that φ evaluates to true under β?

Let φ be a Boolean formula in 3-CNF with clause set $C = \{c_1, c_2, \ldots, c_m\}$ and variable set $V = \{x_1, x_2, \ldots, x_n\}$. Let $\Sigma = \{x_1, x_2, \ldots, x_n, \overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\}$. It is straightforward to construct polynomial-size DFAs for the following languages from $\mathcal{L}_1$:

$$L_{c_i} = \bigcup_{x \in c_i} \Sigma^* x \Sigma^* \quad \text{and} \quad L_{x_j} = \Sigma^* \setminus (\Sigma^* x_j \Sigma^* \overline{x}_j \Sigma^* \cup \Sigma^* \overline{x}_j \Sigma^* x_j \Sigma^*)\,,$$

where $1 \leq i \leq m$ and $1 \leq j \leq n$. Then, the intersection of all L_{c_i} and all L_{x_j} is non-empty if and only if the 3-CNF-SAT instance φ is satisfiable. □

The upper bound shown next also holds for arbitrary commutative languages.

Theorem 9. *The* Intersection Non-emptiness *problem for NFAs accepting arbitrary, i.e., not necessarily star-free,* commutative *languages is in* NP.

Proof. It was shown in [Stockmeyer and Meyer, 1973] that the problem Intersection Non-emptiness is NP-complete for unary NFAs as input. Fix some order $\Sigma = \{a_1, a_2, \ldots, a_r\}$ of the input alphabet. Let $A_1, A_2, \ldots, A_m$ be the NFAs accepting commutative languages with $A_i = (Q_i, \Sigma, \delta_i, q_{0,i}, F_i)$ for $1 \leq i \leq m$. Without loss of generality, we may assume that every F_i is a singleton set, namely $F_i = \{q_{f,i}\}$. For each $1 \leq i \leq m$ and $1 \leq j \leq r$, let $B_{i,j}$ be the automaton over the unary alphabet $\{a_j\}$

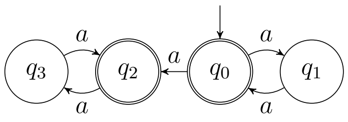


Figure 5: An example of a non-totally star-free NFA that accepts a star-free language.

obtained from A_i by deleting all transitions labeled with letters different from a_j and only retaining those labeled with a_j. Each $B_{i,j}$ will have one initial and one final state. Let $\vec{q_0} = (q_{0,1}, q_{0,2}, \ldots, q_{0,m})$ be the tuple of initial states of the NFAs; they are the initial states of $B_{1,1}, B_{2,1}, \ldots, B_{m,1}$, respectively. Then, nondeterministically guess further tuples $\vec{q_j}$ from $Q_1 \times Q_2 \times \ldots \times Q_m$ for $1 \leq j \leq r-1$. The jth tuple is considered as collecting the final states of the $B_{i,j}$ but also as the start states for the $B_{i,j+1}$. Finally, let $\vec{q_f} = (q_{f,1}, q_{f,2}, \ldots, q_{f,m})$ and consider this as the final states of $B_{1,r}, B_{2,r}, \ldots, B_{m,r}$. Then, for each $1 \leq j \leq r$ solve Intersection Non-emptiness for the unary automata $B_{1,j}, B_{2,j}, \ldots, B_{m,j}$. If there exist words w_j in the intersection of $L(B_{1,j}), L(B_{2,j}), \ldots, L(B_{m,j})$, for each $1 \leq j \leq r$, then, by commutativity, there exists one in $a_1^* a_2^* \cdots a_r^*$, namely, $w_1 w_2 \cdots w_m$, and so the above procedure finds it. Conversely, if the above procedure finds a word, this is contained in the intersection of the languages induced by the A_i's. □

For fixed alphabets, we have a polynomial-time algorithm, showing that the Intersection Non-emptiness problem, with the alphabet size as a parameter, is in XP, for totally star-free NFAs accepting star-free commutative languages. We say that an NFA $A = (Q, \Sigma, \delta, q_0, F)$ is *totally star-free*, if the language accepted by ${}_qA_p = (Q, \Sigma, \delta, q, \{p\})$ is star-free for any states $q, p \in Q$. For instance, poNFAs are totally star-free.

An example of a non-totally star-free NFA accepting a star-free language is given next. Consider the following NFA $A = (\{q_0, q_1, q_2, q_3\}, \delta, q_0, \{q_0, q_2\})$ with $\delta(q_0, a) = \{q_1, q_2\}$, $\delta(q_1, a) = \{q_0\}$, $\delta(q_2, a) = \{q_3\}$, and $\delta(q_3, a) = \{q_2\}$ that accepts the language $\{a\}^*$. The automaton is depicted in Figure 5. Yet, neither $L({}_{q_0}A_{q_0}) = \{aa\}^*$ nor $L({}_{q_0}A_{q_2}) = \{a\}\{aa\}^* \cup \{\varepsilon\}$ are star-free.

The proof of the following theorem uses classical results of Chrobak and Schützenberger [Chrobak, 1986, Schützenberger, 1965].

Theorem 10. *The* Intersection Non-emptiness *problem for totally star-free NFAs accepting* star-free commutative *languages, i.e., commutative languages in $\mathcal{L}_1$, is contained in* XP *with the size of the alphabet as the parameter.*

The proof of Theorem 10 is based on a combinatorial result that might be of independent interest.

Lemma 8. *Let $n \geq 1$ and $t_i, p_i \in \mathbb{N}_{\geq 0}$ for $1 \leq i \leq n$. Set $X = \bigcup_{i=1}^{n} (t_i + \mathbb{N}_{\geq 0} \cdot p_i)$, where $\mathbb{N}_{\geq 0} \cdot p_i = \{ x \cdot p_i \mid x \in \mathbb{N}_{\geq 0} \}$. If there exists a threshold $T \geq 0$ such that $\mathbb{N}_{\geq T} \subseteq X$, then already for $T_{\max} = \max\{ t_i \mid 1 \leq i \leq n \}$, we find $\mathbb{N}_{\geq T_{\max}} \subseteq X$.*

Proof. The assumption basically says that every integer y greater than $T-1$ is congruent to t_ℓ modulo p_ℓ for some $1 \leq \ell \leq n$. More specifically, if x is an arbitrary number with $x \geq T_{\max}$, then $y = x + T \cdot \operatorname{lcm}\{p_1, p_2, \ldots, p_n\}$ is congruent to t_ℓ modulo p_ℓ for some $1 \leq \ell \leq n$. But this implies that x itself is congruent to t_ℓ modulo p_ℓ, and so, as $x \geq t_\ell$, we can write $x = t_\ell + k_\ell \cdot p_\ell$ for some $k_\ell \geq 0$, i.e., $x \in X$. □

This result can be used to prove a polynomial bound for star-free unary languages on an equivalence resembling Schützenberger's characterization of star-freeness [Schützenberger, 1965].

Lemma 9. *Let L be a unary star-free language specified by an NFA A with n states. Then, there is a number N of order $\mathcal{O}(n^2)$ such that $a^N \in L$ if and only if for all $k \in \mathbb{N}_{\geq 0}$, $a^{N+k} \in L$.*

Proof. By a classical result of Chrobak [Chrobak, 1986], the given NFA A on n states can be transformed into a normal form where we have an initial tail with length at most $\mathcal{O}(n^2)$ that branches at a common endpoint into several cycles, where every cycle is of size at most n, see [Chrobak, 1986, Lemma 4.3]. Moreover, this transformation can be performed in polynomial time [Gawrychowski, 2011]. Note that a unary star-free language is either finite or co-finite [Brzozowski, 1976]. If L is finite, then there are no final states on the cycles and we can set N to be equal to the length of the tail, plus one. Otherwise, if $L \subseteq \{a\}^*$ is co-finite, then it can be expressed as a union of a finite language corresponding to the final states on the tail and finitely many languages of the form $\{ a^\ell \mid \ell \in (t + \mathbb{N}_{\geq 0} \cdot p) \}$, where the numbers t and p are induced by the Chrobak normal form. Then we can apply Lemma 8, where the set X is built from the t's and p's, and where the t's are bounded by $T_{\max}$, the sum of the longest tail and the largest cycle, plus one. Note that $T_{\max}$ is in $\mathcal{O}(n^2)$ and that the threshold from Lemma 8 guarantees that every word a^ℓ with $\ell \geq T_{\max}$ is a member of L, as desired. □

Theorem 10. *The* Intersection Non-emptiness *problem for totally star-free NFAs accepting* star-free commutative *languages, i.e., commutative languages in $\mathcal{L}_1$, is contained in* XP *with the size of the alphabet as the parameter.*

Proof. Let $A_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$, for $i \in \{1, 2, \ldots, m\}$, be totally star-free NFAs accepting commutative languages. Let $n_i = |Q_i|$ be the number of states of A_i. Fix some order $\Sigma = \{a_1, a_2, \ldots, a_r\}$.

For $1 \leq i \leq m$ and $1 \leq j \leq r$, as well as $q, p \in Q_i$, let the automaton $B_{i,j,q,p} = (Q_i, \{a_j\}, \delta_i, q, \{p\})$ be obtained from A_i by deleting all transitions not labeled with the letter a_j and only retaining those labeled with a_j. Further, let $A_{i,q,p}$ be obtained from A_i by taking q as (new) initial state and p as the new (and only) final state. As A_i is totally star-free, $L(A_{i,q,p})$ is also star-free. By Schützenberger's Theorem characterizing star-freeness [Schützenberger, 1965], it is immediate that $L(A_{i,q,p}) \cap \Gamma^*$ is also star-free for each $\Gamma \subseteq \Sigma$. In particular, $L(B_{i,j,q,p}) = L(A_{i,q,p}) \cap \{a_j\}^*$ is star-free and commutative.

Recall that $\texttt{perm}(u) = \{\, w \in \Sigma^* \mid |u|_a = |w|_a \text{ for every } a \in \Sigma \,\}$, where $|w|_a$ is equal to the number of letters a in w. Moreover, $\texttt{perm}(L) = \bigcup_{v \in L} \texttt{perm}(v)$. By commutativity, the following property is clear:

$$L(A_i) = \texttt{perm}\left(\bigcup_{p_1, p_2, \ldots, p_{r-1} \in Q_i} \ \bigcup_{p_r \in F_i} L(B_{i,1,q_i,p_1}) \cdot L(B_{i,2,p_1,p_2}) \cdots L(B_{i,r,p_{r-1},p_r}) \right).$$

As A_i accepts a commutative language, by ordering the letters, we find that $w \in L(A_i)$ if and only if $a_1^{\ell_1} a_2^{\ell_2} \cdots a_r^{\ell_r} \in L(A_i)$, for ℓ_j being the number of occurrences of a_j in w, with $1 \leq j \leq r$. Furthermore, the word $a_1^{\ell_1} a_2^{\ell_2} \cdots a_r^{\ell_r}$ is in $L(A_i)$ if and only if for all j with $1 \leq j \leq r$, there is a state $p_j \in Q_i$ such that $a_j^{\ell_j} \in L(B_{i,j,p_{j-1},p_j})$, where $p_0 = q_i$ and $p_r \in F_i$. We can apply Lemma 9 to get constants $N_{i,1}, N_{i,2}, \ldots, N_{i,r} \in \mathcal{O}(n_i^2)$ such that checking membership of $a_j^{\ell_j}$ in $L(B_{i,j,p_{j-1},p_j})$ can be restricted to checking membership for a word of length at most N_j. Now, we describe a polynomial-time procedure to solve Intersection Non-emptiness for fixed alphabets. Set $N_i = \max\{N_{i,1}, N_{i,2}, \ldots, N_{i,r}\}$ with the numbers $N_{i,j}$ from above. Then, we know that a word $a_1^{\ell_1} a_2^{\ell_2} \cdots a_r^{\ell_r}$ is accepted by an input automaton A_i if and only if the word $a_1^{\min\{\ell_1, N_i\}} a_2^{\min\{\ell_2, N_i\}} \cdots a_r^{\min\{\ell_r, N_i\}}$ is accepted by it. If we let $N = \max\{N_1, N_2, \ldots, N_r\}$, we only need to test the $(N+1)^r$ many words $a_1^{i_1} a_2^{i_2} \cdots a_r^{i_r}$ with $0 \leq i_j \leq N$ and $1 \leq j \leq r$ if we can find a word among them that is accepted by all automata A_i for $1 \leq i \leq m$. Altogether, ignoring polynomial factors, this leads to a running time of the form $\mathcal{O}^*(N^r)$. □

Remark 2. Note that Theorem 10 does not hold for arbitrary commutative languages concerning a fixed alphabet, since in the general case, the problem is NP-complete even for languages over a common unary alphabet [Stockmeyer and Meyer, 1973].

5 PSPACE-Completeness

Here, we prove that even when restricted to languages from $\mathcal{B}_1$ or $\mathcal{L}_2$, Intersection Non-emptiness is PSPACE-complete, as it is for unrestricted DFAs or NFAs. We will profit from the close relations of Intersection Non-emptiness to the Non-universality problem for NFAs: Given an NFA A with input alphabet Σ, decide if $L(A) \neq \Sigma^*$. Conversely, we can also observe that Non-universality for NFAs is PSPACE-complete for languages from $\mathcal{B}_1$.

Theorem 11. *The* Intersection Non-emptiness *problem for DFAs or NFAs accepting languages from $\mathcal{B}_1$ or $\mathcal{L}_2$ is* PSPACE*-complete, even for binary input alphabets.*

As $\mathcal{B}_1 \subseteq \mathcal{L}_2$, it is sufficient to show that the problem is PSPACE-hard for $\mathcal{B}_1$. While without paying attention to the size of the input alphabet, this result can be readily obtained by re-analyzing Kozen's original proof in [Kozen, 1977], the restriction to binary input alphabets needs some more care. We modify the proof of Theorem 3 in [Krötsch et al., 2017] that showed PSPACE-completeness for Non-universality for poNFAs (that characterize the level 3/2 of the Straubing-Thérien hierarchy). Also, it can be observed that the languages involved in the intersection are actually locally testable languages. The class of locally testable languages is a sub-class of $\mathcal{B}_1$ and consists of the Boolean closure of languages of the form $u\Sigma^*$, Σ^*v, and $\Sigma^*w\Sigma^*$ where u, v, w are words from Σ^*, see [Pin, 2017].

Corollary 1. *The* Intersection Non-emptiness *problem for DFAs or NFAs accepting locally testable languages is* PSPACE*-complete, even for binary input alphabets.*

Proof. To see our claims, we re-analyze the proof of Theorem 3 in [Krötsch et al., 2017] that shows PSPACE-completeness for the closely related Non-universality problem for NFAs. Similar to Kozen's original proof, this gives a reduction from the general word problem of deterministic polynomial-space bounded Turing Machines. In the proof of Theorem 3 in [Krötsch et al., 2017] that showed PSPACE-completeness for Non-universality for poNFAs (that characterize the level 3/2 of the Straubing-Thérien hierarchy), a polynomial number of binary languages L_i was constructed such that $\bigcup_i L_i \neq \{0,1\}^*$ if and only if the p-space-bounded Turing machine M, where p is some polynomial, accepts a word $x \in \{0,1\}^*$ using space $p(|x|)$. Observe that each of the languages L_i is a polynomial union of languages of the forms $E\{0,1\}^*$, $\{0,1\}^*E$, $\{0,1\}^*E\{0,1\}^*$, or E for finite binary languages E. This means that each L_i belongs to $\mathcal{B}_{1/2}$. Now, observe that $\bigcup_i L_i \neq \{0,1\}^*$ if and only if $\bigcap_i \overline{L_i} \neq \emptyset$. As $\overline{L_i} \in \mathcal{B}_1$ and each L_i (and hence its complement $\overline{L_i}$) can be described by a polynomial size DFA, the claims follow. $\square$

By the proof of Theorem 3 in [Krötsch et al., 2017], also $\bigcup_i L_i$ belongs to $\mathcal{B}_1$, so that we can conclude:

Corollary 2. *The* Non-universality *problem for NFAs accepting languages from $\mathcal{B}_1$ is* PSPACE*-complete, even for binary input alphabets.*

We now present all proof details, because the construction is somewhat subtle.

The proof is based on simulating a p-space-bounded Turing machine M. We are interested in simulating a run of M on a string x. Its configurations are encoded as words over an alphabet Δ, so that with the help of the enhanced alphabet $\Delta_\# = \Delta \cup \{\#\}$, runs of M can be encoded, with # serving as a separator between configurations. More precisely, if Σ_M is the input alphabet of M, Γ_M (containing a special *blank* symbol $\sqcup$) is the tape alphabet, and Q_M is the state alphabet, then transitions take the form $f_M : Q_M \times \Gamma_M \to Q_M \times \Gamma_M \times \{L, R\}$, where L, R indicate the movements of the head. For simplicity, define $\Delta = \Gamma_M \times (Q_M \cup \{\$\})$. A configuration $\gamma \in \Delta^+$ has then the specific properties that it contains exactly one symbol from $\Gamma_M \times Q_M$ and that it has length $p(|x|)$ always, i.e., we are possibly filling up a string that is too short by the blank symbol $\sqcup$. Configuration sequences of M, or runs for short, can be encoded by words from $\#(\Delta^+\#)^*$, or more precisely, from $L_{simple-run} = \#((\Gamma_M \times \{\$\})^*(\Gamma_M \times Q_M)(\Gamma_M \times \{\$\})^*\#)^*$. The latter language can be encoded by a 3-state DFA. However, we will not make use of this language in the following, as it does not fit in the level of the dot-depth hierarchy that we are aiming at.

Let $\Sigma = \{0, 1\}$ be the binary target alphabet. A letter $a \in \Delta_\#$ is first encoded by a binary word $\hat{a}$ of length $K = \lceil \log_2(|\Delta_\#|) \rceil$, but this is only an auxiliary encoding, used to define the block-encoding

$$enc(a) = 001\hat{a}[1]1\hat{a}[2]1 \cdots \hat{a}[K]1$$

of length $L = 2K + 3$, where $\hat{a}[1]$ is the first symbol of the word $\hat{a}$, and so on. This block-encoding is extended to words and sets of words as usual. In order to avoid some case distinctions, we assume that $|\Delta_\#|$ is a power of two, so that $enc(\Delta_\#) = 001\Sigma 1\Sigma 1 \cdots \Sigma 1$. Hence, $\overline{enc}(\Delta_\#) = 00\Sigma^{L-2} \setminus enc(\Delta_\#) = 00\{a_1b_1a_2b_2 \cdots a_Kb_K \mid a_1a_2\cdots a_K \in \Sigma^K \wedge b_1b_2\cdots b_K \in 1^*0\Sigma^*\}$. Clearly, there are DFAs with $\mathcal{O}(L)$ many states accepting $enc(\Delta_\#)$ and $\overline{enc}(\Delta_\#)$ (†). In this proof, we will call DFAs with $\mathcal{O}(L \cdot p(|x|))$ many states *small.* Any encoded word $enc(w)$, with $w \in \Delta_\#^*$, contains the factor 00 only at positions (minus one) that are multiples of L, more precisely: $enc(w)[i] = enc(w)[i+1] = 0$ if and only if $i - 1$ is divisible by L. This observation allows us to construct small DFAs for $\Sigma^*\, enc(\Delta_\#)c\Sigma^*$ (for $c \in \{1, 01\}$) and for $\Sigma^*\, \overline{enc}(\Delta_\#)\Sigma^*$, based on (†). As shown in the proof of Theorem 3 in [Krötsch et al., 2017], the language of words that are <u>not</u> encodings over $\Delta_\#$ at all is the union of the following languages:

1. $(1 \cup 01)\Sigma^*$,
2. $\Sigma^* \, \overline{enc}(\Delta_\#)\Sigma^*$,
3. $\Sigma^* \, enc(\Delta_\#)(1 \cup 01)\Sigma^*$, and
4. $\Sigma^* 00(\bigcup_{i=1}^{L-3} \Sigma^i) = \{w \in \Sigma^* \mid \text{The factor } 00 \text{ is in the last } L-1 \text{ positions}\}$.

Each of these languages can be accepted by small DFAs A_1, A_2, A_3, A_4.

Then, we have to take care of the binary words that cannot be encodings of configuration sequences, because the first configuration is not initial. By our construction, the (unique) initial configuration γ is encoded by a binary string $enc(\gamma)$ of length $L \cdot p(|x|)$, i.e., we consider a language L' which is the complement of $enc(\#\gamma\#)\Sigma^*$, the language of all binary strings that do not start with the encoding of the initial configuration. Let $\#\gamma\# = a_1 a_2 \cdots a_{p(|x|)+2}$. As we already described non-encodings by automata A_1 through A_4, instead of L', we describe $\bigcup_{j=0}^{p(|x|)+2} L'_j$, where L'_0 is the set of all words of which their length is not divisible by L and bounded by $L \cdot (p(|x|) + 2)$. Intuitively, L'_0 is the set of strings that are too short and further contains the empty word. We further define $L'_j = \Sigma^{(j-1)L} \, \overline{enc}(a_j)\Sigma^*$ for $j = 1$ to $p(|x|) + 2$, describing a violation at symbol a_j of the initial configuration γ. Moreover, there are small DFAs $A_5, A_6, \ldots, A_{p(|x|)+7}$ that accept $L'_0, L'_1, \ldots, L_{p(|x|)+2}$.

Assuming a unique final state and also assuming that M cleans up the tape after processing, there is a unique final configuration γ_f that should be reached. Then, invalidity of a computation with respect to the final configuration can be checked as for the initial configuration, giving us small DFAs $A_{p(|x|)+8}, A_{p(|x|)+9}, \ldots, A_{2p(|x|)+10}$.

Finally, we want to check the (complement of the) following property of a valid configuration sequence $\rho \in \#(\Delta^+\#)^*$: any sequence of three letters a, b, c in ρ determines the letter $f(a, b, c)$ that should be present at a distance of $p(|x|) - 1$ to the right. More precisely, we are interested in any factor $abcvdf(a, b, c)e$ of ρ where $|vd| = p(|x|) - 1$. Different scenarios can occur; we only describe three typical situations in the following.

- If $a = (a', \$)$, $b = (b', \$)$, $c = (c', \$)$, then $f(a, b, c) = b$. For d, we know $d \in \{a'\} \times (Q \cup \{\$\})$ and similarly for e, we know $e \in \{c'\} \times (Q \cup \{\$\})$.
- If $a = (a', \$)$, $b = \#$, $c = (c', \$)$, then $f(a, b, c) = \#$. For d, we know $d \in \{a'\} \times (Q \cup \{\$\})$ and similarly for e, we know $e \in \{c'\} \times (Q \cup \{\$\})$.

- If $a = (a', \$)$, $b = (b', q)$, $c = (c', \$)$ and if $f_M(q, b') = (p, \hat{b}', L)$, then $d = (a', p)$, $f(a, b, c) = (\hat{b}, \$)$, and $e = c$.

We refrain from describing all such situations in detail. Yet with some more sloppiness, we write $\overline{enc}(d(f(a,b,c)e)$ for all situations that do not obey the rules for $df(a,b,c)e$ as tentatively formulated before. Now, for each triple $a, b, c \in \Delta_\#$, consider the binary language $L_{a,b,c} = \Sigma^* \cdot enc(abc) \cdot \Sigma^{L\cdot(p(|x|)-1)} \cdot \overline{enc}(d(f(a,b,c)e) \cdot \Sigma^*$. This language can be accepted by a small DFA $A_{a,b,c}$.

Altogether, we described $2p(|x|) + 10 + (|\Delta_\#|)^3$ many languages from $\mathcal{B}_1$ such that their union does not yield Σ^* if and only if M accepts x using $p(|x|)$ space. Moreover, for each of the languages, we can build small DFAs.

6 Conclusion and Open Problems

We have investigated how the increase in complexity within the dot-depth and the Straubing-Thérien hierarchies is reflected in the complexity of the Intersection Non-emptiness problem. We have shown the complexity of this problem is already completely determined by the very first levels of either hierarchy.

Our work leaves open some very interesting questions and directions of research. First, we were not able to prove containment in NP for the Intersection Non-emptiness problem when the input automata are allowed to be NFAs accepting a language in the level 3/2 or in the level 1 of the Straubing-Thérien hierarchy. Interestingly, we have shown that such containment holds in the case of DFAs, but have shown that the technique we have used to prove this containment does not carry over to the context of NFAs. In particular, to show this we have provided the first exponential separation between the state complexity of general NFAs and partially ordered NFAs. The most immediate open question is if Intersection Non-emptiness for NFAs accepting languages in $\mathcal{B}_{1/2}$, $\mathcal{L}_1$, or $\mathcal{L}_{3/2}$ is complete for some level higher up in the polynomial-time hierarchy (PH), or if this case is already PSPACE-complete. Another tantalizing open question is whether one can capture the levels of PH in terms of the Intersection Non-emptiness problem when the input automata are assumed to accept languages belonging to levels of a sub-hierarchy of $\mathcal{L}_2$. Such sub-hierarchies have been considered for instance in [Klíma and Polák, 2011].

It would also be interesting to have a systematic study of these two well-known sub-regular hierarchies for related problems like Non-universality for NFAs or Union Non-universality for DFAs. Notice the technicality that Union Non-universality (similar to INTERSECTION NON-EMPTINESS) has an implicit Boolean operation (now union instead of intersection) within the problem statement, while NON-UNIVERSALITY lacks this implicit Boolean operation. This might lead to a small "shift" in the discussions of the hierarchy levels that involve Boolean closure. Another interesting hierarchy is the group hierarchy [Pin, 1998], where we start with the group languages, i.e., languages acceptable by automata in which every letter induces a permutation of the state set, at level 0. Note that for group languages, INTERSECTION NON-EMPTINESS is NP-complete even for a unary alphabet [Stockmeyer and Meyer, 1973]. As Σ^* is a group language, the Straubing-Thérien hierarchy is contained in the corresponding levels of the group hierarchy, and hence, we get PSPACE-hardness for level 2 and above in this hierarchy. However, we do not know what happens in the levels in between.

References

[Abdulla, 2012] Abdulla, P. A. (2012). Regular model checking. *International Journal on Software Tools for Technology Transfer*, 14(2):109–118.

[Almeida and Klíma, 2010] Almeida, J. and Klíma, O. (2010). New decidable upper bound of the second level in the Straubing-Thérien concatenation hierarchy of star-free languages. *Discrete Mathematics & Theoretical Computer Science*, 12(4):41–58.

[Bouajjani et al., 2000] Bouajjani, A., Jonsson, B., Nilsson, M., and Touili, T. (2000). Regular model checking. In Emerson, E. A. and Sistla, A. P., editors, *Computer Aided Verification, 12th International Conference, CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer.

[Bouajjani et al., 2007] Bouajjani, A., Muscholl, A., and Touili, T. (2007). Permutation rewriting and algorithmic verification. *Information and Computation*, 205:199–224.

[Brzozowski, 1976] Brzozowski, J. A. (1976). Hierarchies of aperiodic languages. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 10(2):33–49.

[Brzozowski and Fich, 1980] Brzozowski, J. A. and Fich, F. E. (1980). Languages of $\mathcal{R}$-trivial monoids. *Journal of Computer and System Sciences*, 20(1):32–49.

[Brzozowski and Knast, 1978] Brzozowski, J. A. and Knast, R. (1978). The dot-depth hierarchy of star-free languages is infinite. *Journal of Computer and System Sciences*, 16(1):37–55.

[Cho and Huynh, 1991] Cho, S. and Huynh, D. T. (1991). Finite-automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88(1):99–116.

[Chrobak, 1986] Chrobak, M. (1986). Finite automata and unary languages. *Theoretical Computer Science*, 47(3):149–158.

[Cohen and Brzozowski, 1971] Cohen, R. S. and Brzozowski, J. A. (1971). Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5(1):1–16.

[Fernau and Krebs, 2017] Fernau, H. and Krebs, A. (2017). Problems on finite automata and the exponential time hypothesis. *Algorithms*, 10(1):24.

[Flum and Grohe, 2006] Flum, J. and Grohe, M. (2006). *Parameterized Complexity Theory*. Springer.

[Gawrychowski, 2011] Gawrychowski, P. (2011). Chrobak normal form revisited, with applications. In Bouchou-Markhoff, B., Caron, P., Champarnaud, J., and Maurel, D., editors, *Implementation and Application of Automata - 16th International Conference, CIAA*, volume 6807 of *Lecture Notes in Computer Science*, pages 142–153. Springer.

[Glaßer and Schmitz, 2000] Glaßer, C. and Schmitz, H. (2000). Decidable hierarchies of starfree languages. In Kapoor, S. and Prasad, S., editors, *Foundations of Software Technology and Theoretical Computer Science, 20th Conference, FST TCS*, volume 1974 of *Lecture Notes in Computer Science*, pages 503–515. Springer.

[Glaßer and Schmitz, 2001] Glaßer, C. and Schmitz, H. (2001). Level 5/2 of the Straubing-Thérien hierarchy for two-letter alphabets. In Kuich, W., Rozenberg, G., and Salomaa, A., editors, *Developments in Language Theory, 5th International Conference, DLT*, volume 2295 of *Lecture Notes in Computer Science*, pages 251–261. Springer.

[Hartmanis et al., 1978] Hartmanis, J., Immerman, N., and Mahaney, S. R. (1978). One-way log-tape reductions. In *19th Annual Symposium on Foundations of Computer Science, FOCS*, pages 65–72. IEEE Computer Society.

[Hoffmann, 2021] Hoffmann, S. (2021). Regularity conditions for iterated shuffle on commutative regular languages. In Maneth, S., editor, *Implementation and Application of Automata - 25th International Conference, CIAA 2021, Virtual Event, July 19-22, 2021, Proceedings*, volume 12803 of *Lecture Notes in Computer Science*, pages 27–38. Springer.

[Hunt III and Rosenkrantz, 1978] Hunt III, H. B. and Rosenkrantz, D. J. (1978). Computational parallels between the regular and context-free languages. *SIAM Journal on Computing*, 7(1):99–114.

[Karakostas et al., 2003] Karakostas, G., Lipton, R. J., and Viglas, A. (2003). On the complexity of intersecting finite state automata and NL versus NP. *Theoretical Computer Science*, 302(1):257–274.

[Kasai and Iwata, 1985] Kasai, T. and Iwata, S. (1985). Gradually intractable problems and nondeterministic log-space lower bounds. *Mathematical Systems Theory*, 18(1):153–170.

[Klein and Zimmermann, 2016] Klein, F. and Zimmermann, M. (2016). How much lookahead is needed to win infinite games? *Logical Methods in Computer Science*, 12(3).

[Klíma and Polák, 2011] Klíma, O. and Polák, L. (2011). Subhierarchies of the second level in the Straubing-Thérien hierarchy. *International Journal of Algebra and Computation*, 21(7):1195–1215.

[Kozen, 1977] Kozen, D. (1977). Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science, FOCS*, pages 254–266. IEEE Computer Society.

[Krötsch et al., 2017] Krötsch, M., Masopust, T., and Thomazo, M. (2017). Complexity of universality and related problems for partially ordered NFAs. *Information and Computation*, 255:177–192.

[Lange and Rossmanith, 1992] Lange, K. and Rossmanith, P. (1992). The emptiness problem for intersections of regular languages. In Havel, I. M. and Koubek, V., editors, *Mathematical Foundations of Computer Science 1992, 17th International Symposium, MFCS*, volume 629 of *Lecture Notes in Computer Science*, pages 346–354. Springer.

[Masopust, 2018] Masopust, T. (2018). Separability by piecewise testable languages is PTime-complete. *Theoretical Computer Science*, 711:109–114.

[Masopust and Krötzsch, 2021] Masopust, T. and Krötzsch, M. (2021). Partially ordered automata and piecewise testability. *Logical Methods in Computer Science*, 17(2).

[Masopust and Thomazo, 2015] Masopust, T. and Thomazo, M. (2015). On the complexity of k-piecewise testability and the depth of automata. In Potapov, I., editor, *Developments in Language Theory - 19th International Conference, DLT*, number 9168 in Lecture Notes in Computer Science, pages 364–376. Springer.

[Morawietz et al., 2020] Morawietz, N., Rehs, C., and Weller, M. (2020). A timecop's work is harder than you think. In Esparza, J. and Král', D., editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August*

24-28, 2020, Prague, Czech Republic, volume 170 of *LIPIcs*, pages 71:1–71:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[Naylor, 2011] Naylor, M. (2011). Abacaba! – using a mathematical pattern to connect art, music, poetry and literature. *Bridges*, pages 89–96.

[Papadimitriou, 1994] Papadimitriou, C. H. (1994). *Computational Complexity.* Addison-Wesley.

[Pin, 1998] Pin, J. (1998). Bridges for concatenation hierarchies. In Larsen, K. G., Skyum, S., and Winskel, G., editors, *Automata, Languages and Programming, 25th International Colloquium, ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 431–442. Springer.

[Pin, 2017] Pin, J. (2017). The dot-depth hierarchy, 45 years later. In Konstantinidis, S., Moreira, N., Reis, R., and Shallit, J. O., editors, *The Role of Theory in Computer Science - Essays Dedicated to Janusz Brzozowski*, pages 177–202. World Scientific.

[Place and Zeitoun, 2019] Place, T. and Zeitoun, M. (2019). Generic results for concatenation hierarchies. *Theory of Computing Systems*, 63(4):849–901.

[Rampersad and Shallit, 2010] Rampersad, N. and Shallit, J. (2010). Detecting patterns in finite regular and context-free languages. *Information Processing Letters*, 110(3):108–112.

[Savitch, 1970] Savitch, W. J. (1970). Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192.

[Schützenberger, 1965] Schützenberger, M. P. (1965). On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194.

[Schwentick et al., 2001] Schwentick, T., Thérien, D., and Vollmer, H. (2001). Partially-ordered two-way automata: A new characterization of DA. In Kuich, W., Rozenberg, G., and Salomaa, A., editors, *Developments in Language Theory, 5th International Conference, DLT*, volume 2295 of *Lecture Notes in Computer Science*, pages 239–250. Springer.

[Stockmeyer and Meyer, 1973] Stockmeyer, L. J. and Meyer, A. R. (1973). Word problems requiring exponential time: Preliminary report. In Aho, A. V., Borodin, A., Constable, R. L., Floyd, R. W., Harrison, M. A., Karp, R. M., and Strong, H. R., editors, *5th Annual Symposium on Theory of Computing, STOC*, pages 1–9. ACM.

[Straubing, 1981] Straubing, H. (1981). A generalization of the Schützenberger product of finite monoids. *Theoretical Computer Science*, 13:137–150.

[Straubing, 1985] Straubing, H. (1985). Finite semigroup varieties of the form V^*D. *Journal of Pure and Applied Algebra*, 36:53–94.

[Sudborough, 1975] Sudborough, I. H. (1975). On tape-bounded complexity classes and multihead finite automata. *Journal of Computer and System Sciences*, 10(1):62–76.

[Thérien, 1981] Thérien, D. (1981). Classification of finite monoids: the language approach. *Theoretical Computer Science*, 14(2):195–208.

[Wareham, 2000] Wareham, T. (2000). The parameterized complexity of intersection and composition operations on sets of finite-state automata. In Yu, S. and Paun, A., editors, *Implementation and Application of Automata, 5th International Conference, CIAA*, volume 2088 of *Lecture Notes in Computer Science*, pages 302–310. Springer.

[Wehar, 2014] Wehar, M. (2014). Hardness results for intersection non-emptiness. In Esparza, J., Fraigniaud, P., Husfeldt, T., and Koutsoupias, E., editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 354–362. Springer.

[Wehar, 2016] Wehar, M. (2016). *On the Complexity of Intersection Non-Emptiness Problems*. PhD thesis, University at Buffalo.

Graphic design: Communication Division, UIB / Print: Skipnes Kommunikasjon AS

uib.no

ISBN: 9788230843956 (print)
9788230857021 (PDF)