Efficient parameterized algorithms on structured graphs

DISSERTATION

zur Erlangung des akademischen Grades

doctor rerum naturalium
(Dr. rer. nat.)
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
der Humboldt-Universität zu Berlin

von

Florian Nelles

Präsidentin der Humboldt-Universität zu Berlin
Prof. Dr. Julia von Blumenthal

Dekanin der Mathematisch-Naturwissenschaftlichen Fakultät
Prof. Dr. Caren Tischendorf

Gutachter:
1. Prof. Dr. Stefan Kratsch
2. Prof. Dr. Tobias Friedrich
3. Prof. Dr. Stephan Kreutzer

Tag der mündlichen Prüfung: 18. April 2023

Abstract

In classical complexity theory, the worst-case running times of algorithms depend (solely) on the size of the input, i.e., depending on the number of vertices and edges if the input is a graph. For a lot of popular tractable problems, the best worst-case running times have not been improved for decades, while for some problems it is also conjectured that it is not possible to solve them any faster. But even if those conjectures turn out to be true, there are different directions of possible further research (apart from actually proving corresponding lower bounds). Next to approximations or heuristics, the goal in *parameterized complexity* is to make use of potential *structure* in the input to solve some instances provably faster. In this reign, the goal is to refine the analysis of the running time of an algorithm by considering a *parameter* that measures some kind of structure in the input, additionally to the input size. A *parameterized algorithm* then utilizes the structure described by the parameter and achieves a running time that is faster than the best general (unparameterized) algorithm for instances of low parameter value. This framework was first introduced to deal with NP-hard problems, but at least since the work of Giannopoulu et al. in 2017, a systematic study of parameterized algorithms also for tractable problems has been started.

In this thesis, we carry forward in this direction and investigate the influence of several parameters on the running times of well-known tractable problems like MAXIMUM MATCHING, VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, or TRIANGLE COUNTING. In the first part of this thesis, we consider the parameters *modular-width*, *clique-width*, and *twin-width*, and give efficient parameterized algorithms. Among other results, we improve the running time for MAXIMUM MATCHING when parameterized by the modular-width of a graph from $\mathcal{O}(\mathsf{h}^4\, n + m)$ by Coudert et al. (SODA 2018) to $\mathcal{O}(\mathsf{h}^2 \log \mathsf{h}\, n + m)$ for graphs with n vertices, m edges, and a modular-width of at most h; as another highlight, we present an $\mathcal{O}(\mathsf{cw}^2\, n^2)$-time algorithm for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS (VAPSP) when parameterized by the clique-width cw of a graph. Throughout, we achieve running times that match the trivial lower bound of a problem for constant parameter values, i.e., quadratic time $\Omega(n^2)$ for VAPSP and linear time $\Omega(n + m)$ for all other problems. Several presented algorithms are additionally *adaptive* algorithms, meaning that they match the running time of a best unparameterized algorithm for worst-case parameter value $k \in \Theta(n)$. Thus, an adaptive parameterized algorithm is asymptotically never worse than the best unparameterized algorithm, while it outperforms the best general algorithm already for slightly non-trivial parameter values of $k \in o(n)$.

As illustrated in the first part of this thesis, for many problems there exist efficient parameterized algorithms regarding multiple parameters, each describing a different kind of structure. In the second part of this thesis, we explore how to combine such homogeneous structures to more general and heterogeneous structures. Using algebraic expressions, we define new combined graph classes of heterogeneous structure in a clean and robust way, and we showcase this for the heterogeneous merge of the parameters tree-depth and modular-width. We present a generic framework for designing efficient parameterized algorithms on such heterogeneous graph classes and apply our framework for the problems VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, NEGATIVE CYCLE DETECTION, and TRIANGLE COUNTING; getting running times that match the homogeneous cases throughout.

Zusammenfassung

In der klassischen Komplexitätstheorie werden worst-case Laufzeiten von Algorithmen typischerweise (einzig) abhängig von der Eingabegröße angegeben, d.h., abhängig von der Anzahl der Knoten und Kanten für Eingaben, die einen Graphen beschreiben. Für viele bekannte Probleme hat sich die worst-case Laufzeit seit Jahrzehnten nicht mehr verbessert, für einige Probleme wird sogar stark vermutet, dass man diese nicht noch schneller lösen kann. Aber selbst wenn diese Vermutungen sich als korrekt erweisen, gibt es, neben dem Beweisen von passenden unteren Schranken, noch vielfältige Möglichkeiten, zu forschen. Neben möglichen heuristischen Algorithmen oder Approximationsalgorithmen, ist es Ziel der *parametrisierten* Komplexitätstheorie, *Strukturen* in der Eingabe auszunutzen, um manche Instanzen nachweislich schneller lösen zu können. In diesem Kontext versucht man die Analyse der Laufzeit dahingehend zu verfeinern, dass man, zusätzlich zu der Eingabengröße, auch einen *Parameter* berücksichtigt, welcher angibt, wie strukturiert die Eingabe bezüglich einer gewissen Eigenschaft ist. Ein *parametrisierter Algorithmus* nutzt dann diese von dem Parameter beschriebene Struktur in der Eingabe aus und erreicht so eine Laufzeit, welche schneller ist als die eines besten unparametrisierten Algorithmus, falls der Parameter klein ist. Ursprünglich betrachtete man parametrisierte Algorithmen vor allem für NP-schwere Probleme, aber spätestens seit der Arbeit von Giannopoulu et al. im Jahre 2017 startete auch eine systematische Betrachtung von effizient lösbaren Problemen. Die vorliegende Dissertation führt die Forschung in diese Richtung weiter aus und untersucht den Einfluss von verschieden Parametern auf die Laufzeit von bekannten effizient lösbaren Problemen, wie z.B. MAXIMUM MATCHING, VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS oder TRIANGLE COUNTING.

Im ersten Hauptteil dieser Arbeit betrachten wir die Parameter *modular-width*, *clique-width* und *twin-width*, und geben effiziente parametrisierte Algorithmen für einige Polynomialzeitprobleme. Unter anderem verbessern wir die Laufzeit für das Problem MAXIMUM MATCHING parametrisiert durch den Parameter *modular-width* von $\mathcal{O}(\mathsf{h}^4\, n+m)$ durch Coudert et al. (SODA 2018) auf eine Laufzeit von $\mathcal{O}(\mathsf{h}^2 \log \mathsf{h}\, n + m)$ für Graphen mit n Knoten, m Kanten und einer *modular-width* höchstens h; als ein weiteres Highlight beschreiben wir einen Algorithmus mit Laufzeit $\mathcal{O}(\mathsf{cw}^2\, n^2)$ für VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS parametrisiert durch den Parameter *clique-width* cw. Durchweg erzielen wir Laufzeiten in dieser Arbeit, welche mit den trivialen unteren Schranken übereinstimmen, falls der betrachtete Parameter konstant ist, d.h. eine quadratische Laufzeit $\Omega(n^2)$ für VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS und lineare Laufzeiten $\Omega(n + m)$ für alle anderen betrachteten Probleme. Einige vorgestellte Algorithmen sind dabei *adaptive* Algorithmen, was bedeutet, dass die Laufzeit von diesen Algorithmen mit der Laufzeit des besten unparametrisierten Algorithm für den größtmöglichen Parameterwert $k \in \Theta(n)$ übereinstimmt. Daher sind adaptive Algorithmen niemals schlechter als die besten unparametrisierten Algorithmen und übertreffen diese bereits für leicht nichttriviale Parameterwerte von $k \in o(n)$.

Aufbauend auf dem großen Erfolg und der Vielzahl an parametrisierten Algorithmen, welche alle jeweils eine andere Struktur ausnutzen, untersuchen wir im zweiten Hauptteil dieser Arbeit, wie man solche unterschiedliche *homogene* Strukturen zu mehr *heterogenen* Strukturen vereinen kann. Ausgehend von algebraischen Ausdrücken, welche benutzt werden können, um von Pa-

rametern beschriebene Strukturen zu definieren, charakterisieren wir klar und robust *heterogene* Strukturen. Wir zeigen exemplarisch, wie sich die Parameter *tree-depth* und *modular-width* heterogen verbinden lassen. Mit Hilfe eines generischem Theorems zur Bestimmung von Laufzeiten auf solchen Strukturen beschreiben wir effiziente Algorithmen auf heterogenen Strukturen mit Laufzeiten, welche im Spezialfall mit den homogenen Algorithmen übereinstimmen.

Acknowledgements

First of all I want to thank everyone that have accepted to be in committee for my dissertation to take the time and read this thesis; especially to my supervisor Stefan Kratsch. The first lecture from Stefan I attended was back in Bonn, called “Parameterized Complexity”, and this was retrospective the reason I wanted to continue doing research after my masters degree. Thank you for introducing me into this field of research, for letting me do my PhD with you, and for being such a great supervisor. In every single meeting you was able to give me motivation, inspiration, and energy to tackle all those research questions. Thank you for introducing me into a whole new level of board games, for establishing all the digital and analog board game evening, and for all other helpful advises outside of research. I have learned a lot from you, and this is not at all limited to theoretical computer science.

Further, I want to thank all my colleagues, starting with Falko, my office mate for the majority of the time (until Corona was a thing). Thanks for all the helpful research discussions; and especially for the non-research discussions. Thanks to Vera and Falko that have read parts of this thesis, and to all my current and former colleagues Narek, Leonid, Robert, Alexandre, Astrid, Sebastian, and Eva-Maria. It was a pleasure to work with all of you. A special thanks also to Galina, Fabian, and Ralf, for helping with all the paperwork and technical problems.

Although the teaching obligations are often seen only as an obstacle and a distraction, my colleagues in teaching and also the students that needed to suffer from my teaching over all these year have made the teaching obligations quite bearable, actually even very enjoyable. Thank you all!

A very special thanks to my friends and family for always being sympathetic, although me not being there on all celebrations and parties.

Last but foremost, I want to thank Katja. Thank you for supporting me all the time with your love, understanding, and patience; for picking me up at the airport or train station at all times, even tough the plain or train was delayed, once again. Thank you taking care of all other things whenever I needed the time for a paper or this thesis. And all this while not complaining a single time, although it was me who decided to work in Berlin, more than 600 kilometers away. Thank you, I would not have done it without you.

Contents

Abstract i

Acknowledgements v

I Fundamentals 1

1 Introduction 3
1.1 Parameterized Algorithms . . . 7
1.2 Towards Heterogeneous Structure . . . 9
1.3 Motivation of the Considered Problems . . . 11
1.4 Thesis Overview . . . 12

2 Preliminaries 15
2.1 General Notation . . . 15
2.2 Graph Theory . . . 16
2.3 Parameters . . . 19
2.4 Considered Problems . . . 24

II Parameterized Algorithms 29

3 Algorithms Parameterized by the Modular-Width of the Input 31
3.1 Definition of Modular-Width . . . 34
3.2 Maximum Matching . . . 38
3.3 General Running Time Theorem . . . 42
3.4 Vertex-Weighted All-Pairs Shortest Path . . . 44
3.5 Triangle Counting . . . 46
3.6 Edge-Disjoint Paths . . . 48
3.6.1 Maximum Edge-Disjoint s-t Paths . . . 49
3.6.2 Global Minimum Cut . . . 52
3.7 Vertex-Disjoint Paths . . . 54
3.7.1 Maximum s-t Vertex Flow . . . 54
3.7.2 Global Minimum Vertex Cut . . . 56
3.8 Conclusion . . . 58

4 Algorithms Parameterized by the Clique-Width of the Input 61
4.1 Definition of Clique-Width and NLC-Width . . . 62
4.2 All-Pairs Shortest Path Parameterized by Clique-Width . . . 66
4.3 Triangle Counting Parameterized by Clique-Width . . . 79
4.4 Conclusion . . . 82

5 Triangle Counting Parameterized by the Twin-Width of the Input 83
5.1 Definition of Twin-Width. . . . 84
5.2 Algorithm . . . 85
5.3 Conclusion . . . 92

III Heterogeneous Graph Classes 93

6 Heterogeneous Structure: Combining Tree-Depth and Modular-Width 95
6.1 Graph Operations and Algebraic Expressions . . . 97
6.2 Heterogeneous Structure . . . 98
6.3 Running Time Framework . . . 100
6.4 Applications . . . 102
6.4.1 Triangle Counting . . . 103
6.4.2 Negative Cycle Detection . . . 104
6.4.3 Vertex-Weighted All-Pairs Shortest Paths . . . 108
6.5 Comparing the Graph Classes . . . 112
6.6 Conclusion . . . 115

IV Conclusion 117

7 Concluding Remarks and Open Problems 119
7.1 Algorithms Parameterized by a Single Parameter . . . 119
7.2 Heterogeneous Structure . . . 122

Bibliography 125

Part I

Fundamentals

1

Introduction

Solving a challenge, a task, or a computation problem as fast as possible is a natural human aspiration. It can already be observed among children who want to be faster than their friends in a specific challenge or who just want to complete a given task to finally have time to play computer games or board games. As an employer, it often goes hand in hand with earning more money if one is able to solve a problem more efficiently and with fewer resources; and as a researcher, determining the best possible worst-case running times for computational problems can be seen as the core of algorithmic research.

Consider for example the problem VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS in which one is given a directed vertex-weighted graph, i.e., a collection of weighted vertices and a specification of the connections between those. The task is now to compute the shortest distances for each pair of vertices. There is the standard textbook algorithm due to Floyd and Warshall from 1962 using dynamic programming that solves this problem in cubic running time $\Theta(n^3)$ where n denotes the number of vertices in the input graph. In contrast to the edge-weighted variant, there is also a theoretically faster algorithm with a time complexity of $\mathcal{O}(n^{2.842})$, however, this algorithm does make use of fast matrix multiplication that is only of theoretical interest and is not of practical use. Indeed, there is no algorithm known for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS with a truly subcubic running time that does not use fast matrix multiplication, i.e., there is no combinatorial algorithm with a running time of $\mathcal{O}(n^{3-\varepsilon})$ for any $\varepsilon > 0$ known. Now, imagine that there is Bob, an employee of a company, and the boss of Bob assigns him a job to compute all pairwise distances of a vertex-weighted directed graph. Bob does look at the graph and sees 20 vertices and a lot of edges. Bob could simply apply the general algorithm by Floyd and Warshall and solve this instance using about $20^3 = 8000$ computations necessarily. This nice algorithm always solves the problem in cubic time $\Theta(n^3)$, irrespective of the concrete instance. But Bob is clever and lazy, he does not need an algorithm that can solve *any* instance, it would be enough to solve *this* specific instance. He looks closely at the input and after a while, he draws the graph in a somewhat organized way, see Figure 1.1: He discovers

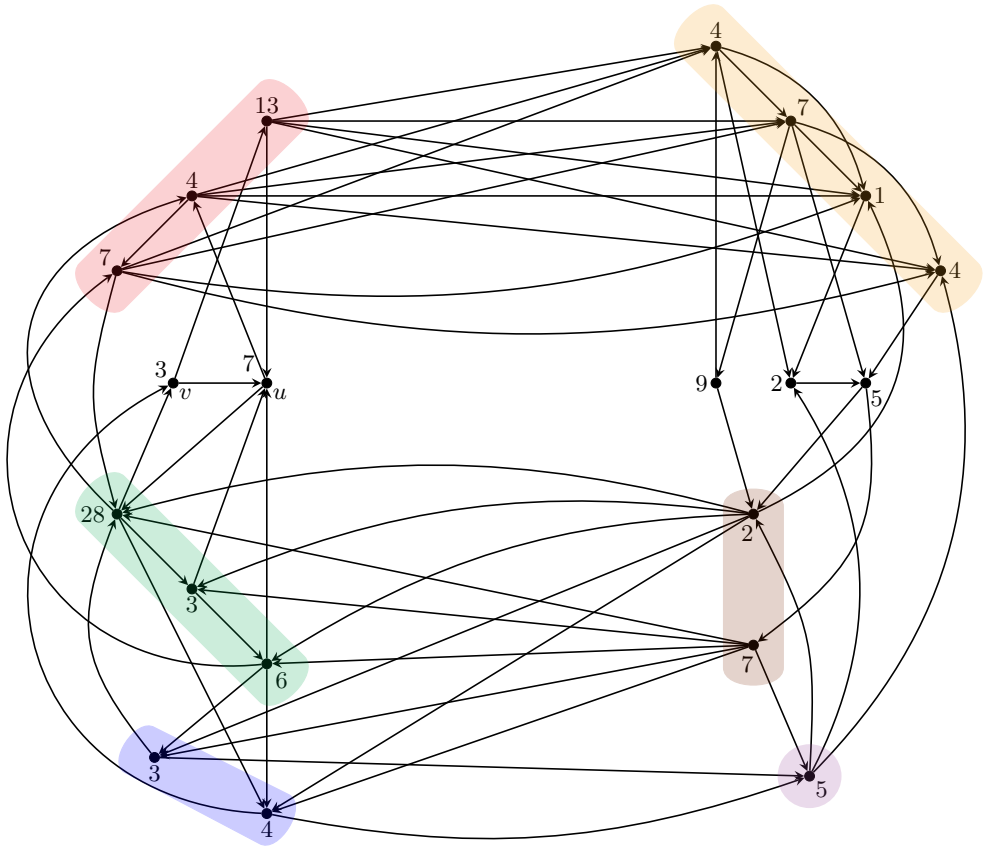


Figure 1.1: The directed graph for which Bob needs to compute the pairwise shortest distances, drawn in a structured way discovered by Bob. The number next to a vertex denotes the weight of this vertex.

that one could divide the graph into two parts, a left part and a right part, and he clusters some vertices using colors such that there are only bridges from one side to the other via full directed joins between some clustered vertices. I.e., the only way one can pass from the left side to the right side is from any red-painted vertex to any yellow-painted vertex or from any blue-painted vertex to the purple-painted vertex; while from the right side to the left side there are only connections between the two brown vertices to the green- and blue-painted vertices.

Now, Bob has the idea to consider each side individually. He computes the distances between each pair of vertices on the left side and right side separately and stores them in two distance matrices D_L and D_R, using only $2 \cdot 10^3 = 2000$ computations. If a shortest path between two vertices from one side only uses vertices from this side, Bob already knows the solution. If a shortest path goes back and forth between the two sides, Bob realizes that every path will enter the right side either on a yellow-painted vertex or on the purple-painted vertex, and will leave the right side on one of the two brown-painted vertices. Since between clustered vertices from different sides there is either a complete join or no edge at all, Bob recognizes that any shortest path that goes to the right side and goes back to the left side will simply use a shortest path from either any yellow-painted vertex to any brown-painted vertex or from the purple-painted vertex to any brown-painted vertex. Using the distances in D_R, Bob can read off that on the right side a shortest "yellow-brown" path has costs $10\,(= 1 + 2 + 5 + 2)$ and a shortest "purple-brown" path has cost $7\,(= 5 + 2)$, by comparing the corresponding values in D_R. Contrarily,

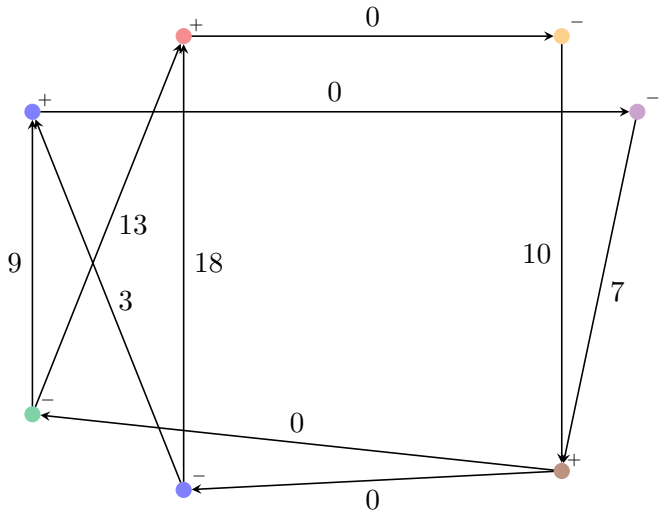


Figure 1.2: The edge-weighted auxiliary graph H with edge weights corresponding to the shortest distance between the respective vertex sets. Colored sets that have incoming edges from the other side are marked by a "-" while colored sets that have outgoing edges to the other side are marked by a "+". The edge weights between a colored vertex with a minus and a colored vertex with a plus correspond to the shortest distance between these colored sets on this side of the graph.

if a shortest path enters and leaves the left side, it will use a shortest path between a colored set that has incoming edges from the right side to a colored set that has outgoing edges to the right side. Bob can again compare the corresponding values in D_L and concludes that on the left side a shortest "green-red" path is of cost $13\,(= 6+7)$, a shortest "green-blue" path of cost $9\,(= 6+3)$, a shortest "blue-red" path of cost $18\,(= 4+3+7+4)$, and a shortest "blue-blue" path of cost 3. Bob summarizes these distances in an edge-weighted auxiliary graph H as depicted in Figure 1.2. Since the shortest paths between clustered vertices already include the start- and end-vertex, the directed edges between the left and right side are assigned the costs zero in the auxiliary graph H. Thus, the graph H encodes several paths from G in a compact way. For example, a shortest $\bullet^-$-$\bullet^+$ path in H corresponds to a shortest u-v path in G of the same weight for a specific blue-painted vertex u and a specific brown-painted vertex v. In particular, one can determine the shortest distances between two colored sets in G with the property that the first and last edge of this path is an edge of a full join between the two sides of G. E.g., a shortest path from any red-colored vertices to any purple-colored vertex in G with this property corresponds to a shortest $\bullet^+$-$\bullet^-$ path in H.[1]

Now, for u and v being two vertices in G, each shortest u-v path that uses vertices on the left *and* right side will consist of three parts: First, a shortest path from u to some colored set on the same side, followed by a shortest path that will go back and forth between the two sides, finalized by a shortest path from a colored set on the same side as v to the vertex v. The middle part corresponds to some shortest path in the auxiliary graph H, hence, Bob computes the shortest distances for each pair of vertices in H, using only roughly $7^3 = 343$ computations. The first and the last part can be looked up quite fast from D_L resp. D_R. E.g., for two vertices

[1]Note that since a shortest $\bullet^+$-$\bullet^-$ path in H corresponds to an u-v path in G with the properties that u is *any* red-painted vertex and v is *any* purple-painted vertex, the weight of this path in H ignores the weights of the vertices u and v.

u and v that are both on the left side in G, Bob can compute the shortest distance as follows:

$$\begin{aligned} dist_G(u,v) = \min\{&dist_{D_L}(u,\bullet^+) + dist_H(\bullet^+,\bullet^-) + dist_{D_L}(\bullet^-,v),\\ &dist_{D_L}(u,\bullet^+) + dist_H(\bullet^+,\bullet^-) + dist_{D_L}(\bullet^-,v),\\ &dist_{D_L}(u,\bullet^+) + dist_H(\bullet^+,\bullet^-) + dist_{D_L}(\bullet^-,v),\\ &dist_{D_L}(u,\bullet^+) + dist_H(\bullet^+,\bullet^-) + dist_{D_L}(\bullet^-,v),\\ &dist_{D_L}(u,v)\} \end{aligned}$$

For example, consider the two vertices in the graph G from Figure 1.1 that are marked by u and v. The values $dist_{D_L}(u,\bullet^+) = 11, dist_{D_L}(u,\bullet^+) = 39, dist_{D_L}(\bullet^-,v) = 13$, $dist_{D_L}(\bullet^-,v) = 7$, and $dist_{D_L}(u,v) = 38$ can be looked up in D_L by comparing the corresponding distances, while the values $dist_H(\bullet^+,\bullet^-) = dist_H(\bullet^+,\bullet^-) = 10$ and $dist_H(\bullet^+,\bullet^-) = dist_H(\bullet^+,\bullet^-) = 7$ are explicitly computed in H. Thus, for this example, it holds that $dist_G(u,v) = 11 + 10 + 7 = 28$. Similar equations hold in the case that u and v are both on the right side or being on different sides (while in the latter case the last argument in the computation above is dropped). Thus, Bob can now compute the shortest distance for each pair of vertices using this approach and needs substantially less time in comparison to the use of the classical algorithm by Floyd and Warshall on the whole graph.

The formal proof of correctness of Bob's approach follows from the discussion in Section 4.2, as the structure that Bob utilizes here is related to a parameter called NLC-width of a graph, which (very) roughly denotes the number of different colors in this example. However, in a graph with structures described by the NLC-width, the left and right sides were already created in a structured way,[2] thus, using the NLC-width one does not need to compute the pairwise distance on each side from scratch, yielding an even better speed-up. Note that in the above example, Bob only roughly halves the size of the graph, which would not result in an asymptotic improvement, however, we will see that graphs of low NLC-width can be indeed solved asymptotically faster. So while it may be true that solving VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS in general needs time $\Theta(n^3)$, for a lot of instances we might find some kind of structure to speed up the computation significantly.

In this thesis, we will consider fundamental polynomial-time solvable problems (like VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS) on graphs that contain some kind of structure. To measure the structure in a graph we will consider various *parameters*. We will give algorithms that are perfectly tailored to these parameters to gain faster algorithms on graphs with low parameter values. Such algorithms are called *parameterized* algorithms. Some parameterized algorithms that we will present in this thesis are even *adaptive*, which means that although they try to make use of the specific structure, these algorithms are asymptotically never worse than the currently best unparameterized algorithm, even if there is no specific structure for the algorithm to exploit; while an adaptive algorithm already outperforms the best unparameterized algorithms for slightly non-trivial parameter values.

Note that in the above example, Bob found the structure that he used to compute the pairwise distances faster by just looking closely at the graph. In general, finding a suitable structure, i.e., to compute a corresponding decomposition or expression, is a separate question of interest. For some structures, even if you know that there is this structure in a graph, it is still NP-hard to find it.

[2] Indeed, the NLC-width is defined via operations on labeled graphs. One operation takes two already created graphs G_1 and G_2 and returns the disjoint union of these graphs with additional edges being added between the two graphs whose existence solely depends on the labels of the vertices.

1.1 Parameterized Algorithms

To define and further motivate the use of parameterized algorithms, let us first define very shortly decision and optimization problems, and the classical analysis of these. For a finite set Σ, called an *alphabet*, we define Σ^* as the set of all finite strings with elements from Σ. Any subset $L \subseteq \Sigma^*$ is called a *language* over the alphabet Σ. We define a *decision problem* as a problem that asks if a given instance does fulfill a specified property or not, i.e., each instance can be evaluated to *true* or *false*. Formally, a decision problem is a pair (I, L) with I and L being languages over a finite alphabet Σ and $L \subseteq I$. The elements in I are called instances and for a given instance $x \in I$ the task is to determine if $x \in L$ or $x \notin L$. An *optimization problem* Π is either a minimization problem or maximization problem, and consists of the following three parts [GJ79, Chapter 6]:

- A set of instances I,
- for each instance $x \in I$, a finite set $S(x)$ of *candidate solutions*, and
- a function m that assigns to each instance $x \in I$ and each candidate solution $\sigma \in S(x)$ a positive rational number $m(x, \sigma)$, called the *solution value* of σ.

For a minimization (resp. maximization) problem, an *optimal solution* for an instance $x \in I$ is a candidate solution $\sigma^* \in S(x)$ such that for all $\sigma \in S(x)$ it holds that $m(x, \sigma^*) \leq m(x, \sigma)$ (resp. $m(x, \sigma^*) \geq m(x, \sigma)$). For an optimization problem Π the task is then to compute an optimal solution $\sigma^* \in S(x)$ for a given instance $x \in I$.

The conventional analysis of the worst-case running time of some algorithm that solves an optimization problem Π is to bound the number of performed operations (solely) relative to the size $|x|$ of the input x. For a lot of optimization problems, including all considered problems in this thesis, the input will consist of a (possibly weighted) graph $G = (V, E)$, i.e., the size of the input can be specified by the number $n = |V|$ of vertices and the number $m = |E|$ of edges.[3] Determining the best possible worst-case running times depending on the input size $|x|$ for computational problems lies at the heart of algorithmic research. However, for many intensively studied problems progress has been stalled for decades and one may suspect that the "correct" running times have already been found. To actually prove that one cannot solve a specific problem any faster in general seems to be challenging since there is still only little known regarding unconditional lower bounds. Considering conditional lower bounds, the recent success of "fine-grained analysis of algorithms" has brought plenty of tight conditional lower bounds for a wealth of problems (see, e.g., [PW10, Bri14, AWY18]). If one is willing to believe in the conjectured worst-case optimality of known algorithms for 3-SUM, ALL-PAIRS-SHORTEST PATHS (APSP), or SATISFIABILITY[4] then lots of other known algorithms must be optimal as well. Even if there is no general agreement on the truth of the conjectures, the previously stalled work can now be focused on beating the best known times for just a few conjectured problems rather than for a multitude of problems. Complementary to the quest for refuting conjectures and beating long-standing fastest algorithms, what should we do if the conjectures and implied lower bounds are true (or if we simply fail to disprove them)? Surely, this would not be the end of the story. Apart from heuristics and approximate algorithms, a possible solution

[3] For some optimization problems, the input will consist of a graph plus some additional information like specified vertices or an algebraic expression. However, the input size will always be linear in the number of vertices and edges of the input graph.

[4] It has been conjectured that there is no $\mathcal{O}(n^{2-\varepsilon})$ time algorithm for 3-SUM, no $\mathcal{O}(n^{3-\varepsilon})$ time for APSP, and that there is no $c < 2$ such that k-SAT can be solved in time $\mathcal{O}(c^n)$ for each fixed k (SETH).

lies in taking advantage of some structure in the input and deriving worst-case running times that depend on parameters that quantify this structure.

The main idea in *parameterized complexity* is to refine the analysis of optimization problems by not only characterizing the running time depending on the size $|x|$ of the instance but additionally considering a *parameter* k that measures some kind of structure in the input instance. This framework was first initiated by Downey and Fellows [DF99] for NP-hard problems, i.e., problems that do not admit polynomial-time algorithms subject to the input size, assuming $\mathsf{P} \neq \mathsf{NP}$. One goal in the context of parameterized complexity is to obtain algorithms that solve a decision problem or optimization problem in time $f(k) \cdot |x|^c$ for some computable function f and a constant c, where $|x|$ denotes the size of the input instance and k denotes the parameter measuring some kind of structure in the input x. Hence, if k is fixed by a constant, $f(k)$ is bounded and such an algorithm does run in polynomial time $\mathcal{O}(|x|^c)$ where the exponent c is not depending on the parameter value k. Algorithms that allow for such a running time for a parameterized problem are called *fixed-parameter tractable* algorithms (*fpt*-algorithms for short). The class of all parameterized problems to which such an fpt-algorithm exists is called FPT. It trivially holds that all problems in P are contained in the class FPT. For more information about parameterized complexity, especially about fpt-algorithms for NP-hard problems, we refer to [CFK$^+$15]. Generally speaking, in the realm of parameterized algorithms the goal is to design algorithms that leverage some structure measured by a parameter to obtain faster algorithms for instances for which the parameter value is low.

This framework is not limited to problems that are NP-hard. Applying this paradigm to the realm of problems in P was done in the literature occasionally. Consider for example the minimum weight circle cover problem in which one is given a set of weighted circular arcs of a circle and the task is to find a subset of the circular arcs of minimum weight that covers the whole circle. For this problem, most of the standard algorithms have a running time also depending on the minimum thickness of the circle, i.e., the minimum number q of arcs crossing any point on the circle. Atallah et al. [ACL95] described in 1995 how to solve this problem in time $\mathcal{O}(qn \log n)$, which improved an earlier $\mathcal{O}(qn^2)$-time algorithm [Ber88] from 1988. Thus, the minimum thickness q of the circle can be seen as a parameter. Similarly, for the LONGEST COMMON SUBSEQUENCE problem, in which one is interested in the length of a longest common subsequence for two strings over a finite alphabet, there is a breakthrough result [ABW15, BK15] that proves that there is no $\mathcal{O}(n^{2-\varepsilon})$ time algorithm for any $\varepsilon > 0$ unless SATISFIABILITY can be solved in $\mathcal{O}((2-\varepsilon')^n)$ time for some $\varepsilon' > 0$ and SETH fails. Long before this result, algorithms were discovered that run much faster than $\mathcal{O}(n^2)$ time when certain parameters are small (cf. [BK18]); curiously, a very recent result of Bringmann and Künnemann [BK18] shows that these are optimal modulo SETH (while giving one new optimal algorithm for binary alphabets). For further examples of early algorithms that utilize some structure we refer for instance to [HW07, OMSW10], or [CEN12].

However, there was no explicit mention of the framework of parameterized algorithms.[5] Giannopoulou et al. [GMN17] reignited in 2017 the paradigm of parameterized algorithms for problems in P (what they called "FPT in P"). Followed by this work, a thorough and more systematic study of parameterized algorithms for tractable problems started, see, e.g., [FLS$^+$18, MNN16, BFNN19, FKM$^+$19, IOO18, CDP19, Duc22a, Duc22b]. To give a general and more formal example for the framework of FPT in P, consider an optimization problem Π over the class of graphs for which the (currently) best algorithms depending solely on the input

[5] Several algorithms that we now would classify as "FPT in P" are developed even before the introduction of the toolbox of FPT for NP-hard problems.

size run in time $\mathcal{O}(n^\gamma)$, with n denoting the number of vertices in the input graph. Let k denote a parameter that measures some kind of structure of the input, e.g., the maximum degree, the minimum size of a vertex cover, or a value that describes how well one can decompose the graph recursively in a specific way. See also Section 2.3 for a (not nearly exhaustive) list of possible parameters. For all possible parameters, it holds that the more structure there is, the lower the parameter value will be. Typically, the parameter value is at most n, which indicates that there is no structure in the input specified by this parameter. The first goal in FPT in P is now to engineer algorithms with a time complexity of $\mathcal{O}(f(k) \cdot n^\beta)$ for some computable function f and with $\beta < \gamma$. Thus, for constant parameter values we achieve a running time of $\mathcal{O}(n^\beta)$ which is faster than the best unparameterized algorithm. Other than for NP-hard problems, the function f does not necessarily need to be exponential in k, since the problem Π can already be solved in polynomial time. Hence, it is often the case that one can also state the function f as a polynomial in k. Hence, the next goal is to get a running time of type $\mathcal{O}(k^\alpha \cdot n^\beta)$ with $\beta < \gamma$. Surely, for such a running time it holds that $\alpha + \beta \geq \gamma$ as otherwise the parameterized algorithm would outperform the unparameterized algorithm even for worst-case parameter value $k = n$. However, $\alpha + \beta = \gamma$ is possible; and thus, a final goal is to develop parameterized algorithms with a time complexity of $\mathcal{O}(k^\alpha \cdot n^\beta)$ with $\alpha + \beta = \gamma$. In this case, it holds that even if $k = n$, the parameterized algorithm is just as good as the unparameterized algorithm; while being faster than the unparameterized algorithm even for slightly non-trivial values $k \in o(n)$. We call algorithms with this property *adaptive* algorithms. Adaptive algorithms were also considered by Iwata et al. [IOO18] and already way earlier for the task of sorting an array of n items. For sorting, there is the (unconditional) lower bound of $\Omega(n \log n)$ for comparison-based sorting, which is matched by well-known sorting algorithms. The goal in the area of adaptive sorting is to find algorithms that are adaptive to presortedness (which can be seen as some structure in the input) with very low running times for almost sorted inputs while maintaining competitive running times as disorder increases (cf. [EW92]).

In this thesis, we will present parameterized algorithms for well-known tractable problems, some of them will even be adaptive. We will primarily consider the parameters modular-width, clique-width, and twin-width. All these three parameters roughly measure how close a graph is to a cograph[6], meaning that the parameter values of any cographs will be smallest possible for each of these parameters.

1.2 Towards Heterogeneous Structure

In Part II of this thesis, we will describe parameterized algorithms for several well-known tractable problems concerning the parameters modular-width, clique-width, or twin-width. Also, many other computational problems can be solved (much) faster on inputs that exhibit certain beneficial structures, while such beneficial structures may show up in several ways (e.g., symmetry, sparsity, structured separations, or graphs of bounded parameter values). Indeed, the framework of parameterized complexity is a huge success. For many problems and various parameters, there are fast parameterized algorithms solving the problem much faster if the considered parameter is low. This is true for both tractable and intractable problems.

For a concrete problem, there are often multiple parameterized algorithms with regard to several different parameters, as we will also see in this thesis for example for the problems

[6]Cograph (from complement reducible graphs) can be defined recursively as follows: A single-vertex graph is a cograph; if $G_1, G_2, \ldots, G_t$ are cographs, then so is their union $G_1 \dot\cup G_2 \dot\cup \ldots \dot\cup G_t$ for all $t \in \mathbb{N}$; and if G is a cograph then so is its complement $\overline{G}$ [CLB81]

VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS or TRIANGLE COUNTING. Each parameter describes a different structure that will be utilized in the parameterized algorithm, while some parameters are simply less general than others, meaning that they specify a more restricted structure. This often comes with a better dependency on the parameter value in the concrete running time, as those algorithms especially make use of the more restricted structure, instead of applying the more general algorithms (and ignoring the finer structure). On the other hand, there might also be parameterized algorithms for a problem with respect to several parameters that are incomparable to each other, i.e., algorithms that utilize completely different structures to get a faster running time. Take for example the incomparable parameters tree-depth and modular-width (the actual definitions are not crucial here and are postponed to Section 2.3 and Section 3.1). Low tree-depth does always imply that the graph is somewhat sparse since the number of edges in a graph of tree-depth k is always at most $k \cdot n$. Thus, algorithms parameterized by the tree-depth alone can never be applied successfully on dense graphs, i.e., graphs with a lot of edges in proportion to the number of vertices. On the other side, the modular-width of all cliques is zero, and intuitively, a graph of low modular-width is very likely to have a lot of edges because low modular-width usually requires full joins between large parts of the graphs. At the same time, there are parameterized algorithms for both, the tree-depth and the modular-width of a graph for, e.g., the problems TRIANGLE COUNTING or also VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS (the algorithms regarding modular-width can be found in Chapter 3, while the algorithm regarding tree-depth is a special case of the presented algorithms in Chapter 6). One goal is now to design an algorithm that not only can deal with both types of structures at once, but also utilize the different approaches to be applicable to graphs with a much more interwoven structure arising from the different definitions of the parameters.

That is the motivation and starting point of the discussions in Part III of this thesis, where we combine the structural ideas as well as the algorithmic ideas from two or more different parameters to obtain efficient parameterized algorithms that can be applied to much less restrictive input classes, without losing much in terms of running times. Building on the use of algebraic expressions that characterize the structure measured by a parameter, we present a clean and robust way of combining such homogeneous structures into a more complex *heterogeneous* structure. We showcase our approach for the combination of the parameters tree-depth and modular-width, but the applicability to other parameters is evident. By using an operational way via algebraic expressions to define the parameters tree-depth and modular-width, we first define the natural notion of modular tree-depth, where instead of the size of the pattern graph (as done for modular-width) only the tree-depth of the pattern graphs is taken into account. Based on the operational definition of these three parameters, we then can define heterogeneous graph classes for all combinations. Since the operations may be intertwined arbitrarily, the new heterogeneous graph classes are way more general than just the union of the corresponding homogeneous graph classes, while it turns out that the obtained running times will be just the sum of those running times achieved in the homogeneous case, with potentially much lower parameter values. Hence, we can apply the algorithm for the heterogeneous case to graphs that otherwise were not able to be captured by a single parameter alone; and we extend the class of graphs for which we get faster algorithms significantly. Interestingly, the algorithm parameterized by the tree-depth alone for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS of time $\mathcal{O}(\mathsf{td}\, n^2)$ is not impressive, since the implied sparsity of graphs of tree-depth td directly results in a running time of $\mathcal{O}(\mathsf{td}\, n^2 + n^2 \log\log n)$ using the algorithm of Pettie and Ramachandran [PR05]. However, it is of great benefit for the combination with other parameters.

1.3 Motivation of the Considered Problems

In this thesis, we will consider fundamental tractable problems like MAXIMUM MATCHING, VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, TRIANGLE COUNTING, or several problems related to edge- and vertex-disjoint paths. For the currently best parameterized and unparameterized algorithms for each of the considered problems, we refer to Section 2.4

In the MAXIMUM MATCHING problem, the task is to find a maximum set $X \subseteq E$ of pairwise disjoint edges in an undirected graph $G = (V, E)$. A set of edges is said to be disjoint if every vertex $v \in V$ is incident to at most one edge in X. MAXIMUM MATCHING is one of the oldest tractable problems considered in the literature. The first polynomial-time algorithm for this problem was already established in 1965 [Edm65]. Maximum matchings play an important role in several other problems, e.g., in the realm of scheduling problems; for instance, to find an optimal sequencing of two equivalent processors for n tasks with a given precedence relation (provided by an arbitrary acyclic directed graph), the MAXIMUM MATCHING-problem is an important subroutine for finding a maximum number of disjoint compatible task pairs [FKN69]. Another application can also be found in medicine, e.g., for kidney exchanges. Most kidney transplant recipients already come to the hospital with a relative or friend that is willing to donate a kidney. However, these patient-donor pairs are often incompatible, for example, due to wrong blood types. Thus, one does look for another patient-donor pair with complementary blood types such that they can help each other out. In this context, a lot of research is done to develop a provably fair strategy for the exchange of two incompatible patient-donor pairs for kidney exchange [RSÜ04, RSÜ05]. Maximum matchings are also used to solve other problems, e.g., by König's Theorem [Kön31], the size of a minimum vertex cover in a bipartite graph is equal to the size of a maximum matching in this graph. The MAXIMUM b-MATCHING is an extension of the MAXIMUM MATCHING problem in which one is interested in a multiset M of its edges such that the number of edges of M that are incident to a vertex v does not exceed $b(v)$, for every vertex v and a given function b that defines for each vertex a maximum capacity $b(v)$. Applications for MAXIMUM b-MATCHING are also manifolds, see for example [Gue07] for an application on combinatorial auctions.

Next to MAXIMUM MATCHING, also ALL-PAIRS SHORTEST PATHS is a fundamental and much-studied problem in the field of algorithmic graph theory. Here, one is interested in the shortest distances for all pairs of vertices in a possibly weighted graph. Besides the theoretical interest in the problem, the computation of all pairwise distances is an important routine for many practical applications, e.g., it is closely related to several vertex centrality measures in networks. For example, the betweenness centrality [Fre77] of a vertex $v \in V$ in a graph $G = (V, E)$ is defined as the sum over all pairs $s, t \in V \setminus \{v\}$ taking the fraction of the number of shortest s-t paths that pass through v to the total number of shortest s-t paths. For example in transportation networks, vertices of high betweenness centrality are those vertices that will be frequently used for transportation through the network, and thus, these vertices are of critical interest. See also [Ren15] for a summary of applications of the betweenness centrality. The ALL-PAIRS SHORTEST PATHS-problem is also considered as the core of many routing problems and has applications for example in areas such as routing protocols, driving direction on web mappings, traffic assignment problems, and many more. See also the survey of Susmita [SP15] for more applications. Most of these applications have also vertex-weighted variants, however, the vertex-weighted variant is less studied in the literature.

Like ALL-PAIRS SHORTEST PATHS, also the TRIANGLE COUNTING-problem, in which one needs to determine the number of triangles, i.e., the number of subgraphs that are isomorphic to K_3, is a very important subroutine for computing network indices. Many coefficients, such as

the clustering coefficient [WS98] or the transitive coefficient [HK79], are important indices for measuring the frequency of clusters and the likelihood of dense communities in graphs. Such indices of social networks or urban social contact structures are of immense importance for social scientists, urban planners, governments, and many others, cf. [EKM+04, BBCG10]. Also the spread of infectious diseases, in particular how to control the disease, e.g., by identifying potential super-spreaders, can be listed as a relevant example. For the clustering coefficient, one first considers the local clustering coefficient of a node v in a graph G that is defined as the fraction of existing edges in $N_G(v)$ to the number of all possible edges in $N_G(v)$. The clustering coefficient of a graph G is then defined as the normalized sum over the local clustering coefficients of each node. The transitive coefficient of a graph is the quotient of three times the number of triangles in G and the number of all wedges, i.e., paths of length two, in the graph.

Determining the maximum number of edge- or vertex disjoint paths in a graph and the more general problem of computing a maximum flow in a network also have several applications. The vertex or edge connectivity number in a graph indicates how robust a graph is towards vertex deletions or edge deletions. Using a maximum flow, one can determine the maximum size of a matching in a bipartite graph, and thus, by König's Theorem, also the minimum size of a vertex cover in bipartite graphs. Also, certain combinatorial problems can be solved using a maximum flow computation, e.g., for airline scheduling where one needs to determine which crew should take charge of which aircraft in the flight plan. Another example is to determine which football teams still have a chance to win the championship, which is not only depending on how many games a team still has to play but also against whom they play.

1.4 Thesis Overview

This thesis consists of four parts. The first part consists of this introduction and Chapter 2, in which we define all the necessary notations and definitions about graph theory. Further, we give in Chapter 2 the definitions of graph parameters that are related to the considered parameters modular-width, clique-width, and twin-width and discuss the relationship among those. We close Chapter 2 with related work about the problems considered in this thesis.

Part II is all about efficient parameterized algorithms. In Chapter 3, we consider the parameter modular-width. After the formal definition of this parameter, we give efficient parameterized algorithm w.r.t. the modular-width for the problems MAXIMUM MATCHING, MAXIMUM b-MATCHING, VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, TRIANGLE COUNTING, MAXIMUM EDGE-DISJOINT s-t PATHS, GLOBAL MINIMUM EDGE CUT, MAXIMUM s-t VERTEX-FLOW, and GLOBAL MINIMUM VERTEX CUT. Here, we also present a running-time framework that simplifies the computation of the running time for most algorithms that make use of the structure quantified by the parameter modular-width, including all presented algorithms in this chapter. The very recent breakthrough result of an almost linear time algorithm for computing a maximum flow has not been taken into consideration in this work. In the conclusion of Chapter 3, we shortly discuss the implication of this result to the problems regarding edge- and vertex-disjoint paths. The results of Chapter 3 are based on joint work with Stefan Kratsch [KN18, KN20]. In Chapter 4, we consider the parameter clique-width and the closely related parameter NLC-width. First, we define both parameters for directed and undirected graphs. Then, we present parameterized algorithms for the problems VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS and TRIANGLE COUNTING. The results of this chapter are based on joint work with Stefan Kratsch [KN20]. We complete Part II in Chapter 5 and consider the most general parameter of this thesis, twin-width. We define this parameter and present an algorithm for solving the

TRIANGLE COUNTING problem. The result of this chapter is based on joint work with Stefan Kratsch and Alexandre Simon [KNS22].

In Part III, we move from the homogeneous graph classes as presented in Part II towards *heterogeneous* graph classes. We first recall operations-based definitions of the parameters tree-depth and modular-width alone. Building on these operations, we describe a clean and robust way of formalizing graph classes of heterogeneous structure. We introduce the new notion of modular tree-depth and define for all combinations of the three parameters tree-depth, modular-width, and modular tree-depth, a corresponding graph class. For these much more general graph classes, we extend the framework of Iwata et al. [IOO18], which applies to the operations that define the tree-depth alone, to all required operations. We then extend the running time framework for modular-width from Chapter 3 to all these graph classes and give application for the problems VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, NEGATIVE CYCLE DETECTION, and TRIANGLE COUNTING. Throughout, the algorithmic results relative to the heterogeneous measures match the best running times known for the homogeneous cases. Finally, we compare the arising heterogeneous graph classes among each other and relative to existing parameters. The results of Part III are based on joint work with Stefan Kratsch [KN22]. We conclude this thesis in Part IV with some open problems and final remarks.

2

Preliminaries

In this chapter, we will introduce the basic notations and fundamentals that we will use throughout this thesis. After defining some general notations in Section 2.1, we will present an overview of all graph-related notations that we will use in this thesis in Section 2.2. We continue in Section 2.3 by showcasing related parameters to the ones considered in this thesis and by discussing the relationships among them. Finally, in Section 2.4 we present related work regarding the problems that we will study in this thesis. This section about graph theory follows mostly the books of Korte and Vygen [KVKV11] and Diestel [Die12].

2.1 General Notation

We define $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ as the set of natural numbers and denote by $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$ the set of natural numbers excluding the zero. For any integers $j, k \in \mathbb{N}$ with $j < k$, we shortcut $[j, k] = \{j, j+1, \ldots, k\}$ and $[k] = [1, k] = \{1, 2, \ldots, k\}$. For a set A we define $|A|$ as the number of elements in A. For two sets A and B with $A \cap B = \emptyset$, we sometimes highlight for the union $A \cup B$ that these sets are disjoint by writing $A \dot{\cup} B$ instead. We say that two sets A and B *overlap* if $A \cap B \neq \emptyset$, $A \setminus B \neq \emptyset$, and $B \setminus A \neq \emptyset$. We define the *symmetric difference* of two sets A and B by $A \Delta B = (A \setminus B) \cup (B \setminus A)$.

For a set A and $k \in \mathbb{N}$, we denote by $\binom{A}{k} \coloneqq \{B \subseteq A \mid |B| = k\}$ the set of all size-k subsets of A and we define $A^k = \{(a_1, a_2, \ldots, a_k) \mid a_1, a_2, \ldots, a_k \in A\}$ as the set of all k-tuples with elements in A. In particular, the set $[k]^2$ defines the set of all 2-tuples over the set $\{1, 2, \ldots, k\}$ and we define a binary relation S over the set $[k]$ by $S \subseteq [k]^2$. By $2^A = \{B \mid B \subseteq A\}$ we denote the *power set* of the set A. Let A be a non-empty set and let $A_1, A_2, \ldots, A_k \subseteq A$. We say that $\{A_1, A_2, \ldots, A_k\}$ is a *partition* of A, if every element of A is contained in exactly one A_i for $i \in [k]$, i.e., $A = \cup_{i \in [k]} A_i$ and $A_i \cap A_j = \emptyset$ for all $i, j \in [k]$ with $i \neq j$. For a set A and the power set 2^A, we say that a set $B \in 2^A$ is *maximal* regarding some property if for all other sets $B' \in 2^A$ with this property it holds that either $B \not\subseteq B'$ or $B = B'$.

2.2 Graph Theory

We define an *undirected* graph as a tuple $G = (V, E)$ consisting of a finite set V and a set $E \subseteq \binom{V}{2}$. Similarly, we define a *directed* graph as a tuple $G = (V, E)$ consisting of a finite set V and a set $E \subseteq V^2$. We refer to the elements in V as *vertices* and to the elements in E as *edges*. In a directed graph we sometimes call elements in E *arcs* for better distinction. In this thesis, we will only consider *simple* graphs, i.e., graphs without multiple edges between two vertices and without loops, i.e., edges between the same vertex. If we simply write graph in this section, we refer to both, undirected and directed graphs. For a graph $G = (V, E)$ we denote by $V(G)$ the set of vertices and by $E(G)$ the set of edges in G, independently of the actual name of the vertex set or edge set, i.e., for a graph $H = (W, F)$ we denote by $V(H) = W$ the vertex set and by $E(H) = F$ the edge set of the graph H. We define the *complement graph* $\overline{G}$ of a graph $G = (V, E)$ as the graph with the same vertex set and exactly those edges that are not present in the original graph, i.e., $\overline{G} = (V, \binom{V}{2} \setminus E)$ if G is undirected resp. $\overline{G} = (V, V^2 \setminus E)$ if G is directed. For a directed graph $G = (V, A)$, we define the *edge-flipped* graph as the directed graph $G^{\leftarrow} = (V, A^{\leftarrow})$ defined over the same vertex set as G and with $A^{\leftarrow} = \{(u, v) \in V^2 \mid (v, u) \in A\}$. Further, we define for a directed graph $G = (V, A)$ the *underlying undirected graph* $u(G) = (V, E)$ as the undirected graph defined over the same vertex set and $E = \{\{u, v\} \in \binom{V}{2} \mid (u, v) \in A \vee (v, u) \in A\}$. On the other hand, for an undirected graph $G = (V, E)$, we denote by $\overleftrightarrow{G} = (V, A)$ with $A = \{(u, v), (v, u) \in V^2 \mid \{u, v\} \in E\}$ the corresponding (bi-)directed graph.

Subgraphs. A graph $H = (W, F)$ is called a *subgraph* of a graph $G = (V, E)$ if it holds that $W \subseteq V$ and $F \subseteq E \cap \binom{W}{2}$ for undirected graphs resp. $F \subseteq E \cap W^2$ in the directed case. For a set $M \subseteq V$, we call $G[M]$ the subgraph of G *induced* by M, defined by $V(G[M]) = M$ and $E(G[M]) = \{\{u, v\} \in E \mid u \in M \wedge v \in M\}$ if G is undirected resp. $E(G[M]) = \{(u, v) \in E \mid u \in M \wedge v \in M\}$ if G is directed. For a set $M \subseteq V(G)$, we shorthand $G - M$ for the induced subgraph $G[V \setminus M]$ and for a singleton set $\{x\}$ with $x \in V$, we write $G - x$ instead of $G - \{x\}$. Similarly, for a graph $G = (V, E)$ and a set $F \subseteq E$, we shorthand $G - F$ for the graph $(V, E \setminus F)$ and for a singleton set $\{f\}$ with $f \in E$, we write $G - f$ instead of $G - \{f\}$.

Adjacency. Consider a graph $G = (V, E)$. For an edge $\{u, v\} \in E$, resp. $(u, v) \in E$ with $u, v \in V$ we call u and v *adjacent*. Further, we call the vertices u (and v) *incident* to this edge. Two edges in G that share an incident vertex are also called *adjacent*. For a directed edge $(u, v) \in E$ we sometimes say that this edge *starts* in u and *ends* in v, or it goes *from* u *to* v. In a graph $G = (V, E)$, we denote for a vertex $v \in V$ the *open neighborhood* $N_G(v)$ as the set of all vertices in G that are adjacent to v and denote by $\deg_G(v) = |N_G(v)|$ the *degree* of this vertex. We define the *closed neighborhood* of a vertex $v \in V$ as $N_G[v] = N_G(v) \cup \{v\}$. If G is directed, we additionally distinguish between $N_G^+(v) = \{u \in V \mid (v, u) \in E\}$ and $N_G^-(v) = \{u \in V \mid (u, v) \in E\}$. We denote by $\deg_G^+(v) = |N_G^+(v)|$ the *out-degree* and by $\deg_G^-(v) = |N_G^-(v)|$ the *in-degree* of the vertex $v \in V$. As in the undirected case, we define $N_G^+[v] = N_G^+(v) \cup \{v\}$ and $N_G^-[v] = N_G^-(v) \cup \{v\}$. For a set of vertices $X \subseteq V$, we define for an undirected graph G the closed neighborhood of X as $N_G[X] = \cup_{v \in X} N_G[v]$ and the open neighborhood of X as $N_G(X) = N_G[X] \setminus X$. In the directed case, we define analogously the sets $N_G^+[X]$, $N_G^-[X]$, $N_G^+(X)$, and $N_G^-(X)$. Further, we denote for an undirected graph G by $\delta_G(v)$ the set of incident edges of a vertex $v \in V$, resp. for directed graphs $\delta_G^+(v) = \{(x, y) \in E \mid x = v\}$ and $\delta_G^-(v) = \{(x, y) \in E \mid y = v\}$. The extensions to sets of vertices can be defined analogously,

but are not used in this thesis. If the graph G is clear from the context, we may omit the subscript in all our notations.

Two vertices $u, v \in V(G)$ in an undirected graph $G = (V, E)$ are called *twins* if $N(v) \setminus \{u, v\} = N(u) \setminus \{u, v\}$. In particular, we call u and v *true twins* if additional $\{u, v\} \in E$ and *false twins* if $\{u, v\} \notin E$.

A vertex of degree zero is called *isolated* and a vertex of maximum degree is called *universal*, i.e., of degree $|V| - 1$ for undirected graphs resp. of in-degree and out-degree $|V| - 1$ for directed graphs. A graph in which every vertex is an isolated vertex is called an *isolated graph* and a graph in which every vertex is a universal vertex is called a *complete graph* or a *clique*. We denote by I_i resp. K_i the isolated graph resp. the complete graph with i vertices for $i \in \mathbb{N}^+$.

For a graph $G = (V, E)$, we say that two disjoint vertex sets $L, R \subseteq V$ are connected by a *full join* in G if $\{\{u, v\} \in \binom{V}{2} \mid u \in L \wedge v \in R\} \subseteq E$ resp. connected by a *directed full join from L to R* if $\{(u, v) \in V^2 \mid u \in L \wedge v \in R\} \subseteq E$. We call an undirected graph $G = (V, E)$ a *bipartite graph* if one can partition the vertex set into two sets $V = L \dot\cup R$ such that $G[L]$ and $G[R]$ are isolated graphs. If further L and R are connected by a full join, we call G a *complete bipartite graph*. We denote by $K_{i,j}$ the complete bipartite graph with $|L| = i$ and $|R| = j$ for $i, j \in \mathbb{N}^+$.

Paths, trees, and connectivity. In a graph $G = (V, E)$, we define a *walk* as a sequence of vertices $(v_1, v_2, \ldots, v_t)$ with $\{v_i, v_{i+1}\} \in E$, resp. $(v_i, v_{i+1}) \in E$ for some $t \in \mathbb{N}^+$ and all $i \in [t-1]$. The *length* of walk is defined as the number of vertices in the walk minus one. A walk is called a *closed walk* if $v_1 = v_n$. A *path* is a walk with the property that $v_i \neq v_j$ for all $i, j \in [t]$ and $i \neq j$, and a *cycle* is a closed walk with the property that $v_i \neq v_j$ for all $i, j \in [t-1]$ and $i \neq j$. We denote a path consisting of $t \geq 1$ vertices by P_t and a cycle consisting of $t \geq 3$ vertices by C_t. A path $P = (v_1, v_2, \ldots, v_t)$ that starts in v_1 and ends in v_t is also called a v_1-v_t path. We refer to the vertices $v_2, v_3, \ldots, v_{t-1}$ as *internal nodes* of the path P and we define by $P_{[v_i, v_j]}$ the subpath of P starting in v_i and ending in v_j for $i, j \in [t]$ with $i < j$. An undirected graph $G = (V, E)$ is *connected* if there exists a path between each pair of vertices in V. A directed graph is called connected, if the underlying undirected graph $u(G)$ is connected, and *strongly connected* if there exists a path between each pair of vertices in V. A *connected component* in a graph G is a maximal connected subgraph of G.

A graph that does not contain any cycle is called *acyclic*. An undirected and acyclic graph is called a *forest* and a connected forest is called a *tree*, i.e, each connected component of a forest is a tree. In a tree, each vertex of degree one is called a *leaf*. A *star* is a tree with at most one vertex that is not a leaf. Note that as a tree is acyclic and connected, there is a unique path between every pair of vertices in a tree. A *rooted tree* is a tree $T = (V, E)$ together with a designated vertex $r \in V$ that we call the *root* of T. Consider a vertex $v \in V$ of a rooted tree $T = (V, E)$ with root node $r \in V$. We call those neighbors of v that are not contained in the unique v-r path the *children* of v; and we call the unique neighbor of v that is contained in the unique v-r path in T the *parent* of v in T. The *height* or *depth* of a rooted tree $T = (V, E)$ with root node $r \in V$ is the maximum length of any r-v path in T for all $v \in V$. A *rooted forest* is a graph in which each connected component is a rooted tree and the height of a rooted forest is the maximum height of any rooted tree in it. The *closure* of a rooted forest F, denoted by $clos(F)$, is the graph H obtained from the graph F by adding for all $u, v \in V(F)$ the edge $\{u, v\}$ if u and v are contained in a leaf-to-root path in any rooted tree in F.

A graph $G = (V, E)$ is said to be *k-vertex connected*, if $G - X$ is connected for all $X \in \binom{V}{k-1}$, i.e., G remains connected after deleting fewer than k arbitrary vertices. The *vertex-connectivity* of a graph G, denoted by $\kappa(G)$, is the maximum $k \in \mathbb{N}$ such that G is k-vertex connected. Analogously, a graph $G = (V, E)$ is said to be *k-edge connected*, if $G' = (V, E \setminus E')$ is connected

for all $E' \subseteq \binom{E}{k-1}$, i.e., G remains connected after deleting fewer than k arbitrary edges. The *edge-connectivity* of a graph G, denoted by $\lambda(G)$, is the maximum $k \in \mathbb{N}$ such that G is k-edge connected.

Weighted graphs. An *edge-weighted graph* is a graph $G = (V, E)$ attributed by a function c that assigns a *weight* or *cost* $c(e)$ to each edge $e \in E$. The *costs* of a path $P = (v_1, v_2, \ldots, v_t)$ in G with edge weights c is defined as $c(P) = \sum_{i=1}^{t-1} c(\{v_i, v_{i+1}\})$ if G is undirected and $c(P) = \sum_{i=1}^{t-1} c((v_i, v_{i+1}))$ if G is directed. For two vertices $u, v \in V$, we denote by $dist_{G,c}(u, v)$ the minimum cost of all u-v paths in G. In particular, it holds that $dist_{G,c}(u, u) = 0$ for all $u \in V$.

A *vertex-weighted graph* is a graph $G = (V, E)$ attributed by a function ω that assigns a weight $\omega(v)$ to each vertex $v \in V$. The *costs* of a path $P = (v_1, v_2, \ldots, v_t)$ in G with vertex weights ω is defined as $\omega(P) = \sum_{i=1}^{t} \omega(v_i)$. This holds for both, undirected and directed graphs. For two vertices $u, v \in V$, we denote by $dist_{G,\omega}(u, v)$ the minimum cost of all u-v paths in G. In particular, every u-v path in a graph G with vertex weights ω has minimum weight $\omega(u) + \omega(v)$ and it holds that $dist_{G,\omega}(u, u) = \omega(u)$ for $u, v \in V$. If the weight function is clear from context, we may omit c resp. ω in the subscript. For both, edge-weighted and vertex-weighted graphs, we define $dist_G(u, X) = \min_{v \in X} dist_G(u, v)$ for a vertex $u \in V$ and any vertex set $X \subseteq V$. Similarly, we define $dist_G(X, Y) = \min_{u \in X, v \in Y}(u, v)$ for two sets of vertices $X, Y \subseteq V$.

Flow networks. An *edge-capacitated network* $N = (G, s, t, c)$ is a 4-tuple consisting of a directed graph $G = (V, E)$, two specified vertices $s \in V$ (called the *source*) and $t \in V$ (called the *sink*), and a capacity function defined on the set of edges $c \colon E \to \mathbb{R}_{\geq 0}$. A *flow* f in an edge-capacitated network N is a function $f \colon V \times V \to \mathbb{R}$ with the properties that it holds that $f(u, v) = 0$ if $(u, v) \notin E$ and $(v, u) \notin E$, and additionally

- $\forall (u, v) \in E : f(u, v) \leq c(u, v)$ (capacity constraint)
- $\forall u, v \in V : f(u, v) = -f(v, u)$ (skew symmetry property)
- $\forall u \in V \setminus \{s, t\} : \sum_{v \in V} f(u, v) = 0$ (flow conservation property)

In a *vertex-capacitated network*, the capacity function is defined over the set of vertices, i.e., $c \colon V \to \mathbb{R}_{\geq 0}$. A flow in a vertex-capacitated network is a function $f \colon V \times V \to \mathbb{R}$ with the same properties as defined for the edge-capacitated case except the capacity constraint is replaced by

- $\forall v \in V : \sum_{(u,v) \in \delta_G^-(v)} f(u, v) \leq c(v)$ (capacity constraint)

In the undirected setting, we again (only) need to adjust the capacity constraint. For edge-capacitated networks on undirected graphs we require the flow to fulfill the inequality $f(u, v) \leq c(\{u, v\})$ for all $u, v \in V$ with $\{u, v\} \in E$, while in vertex-capacitated networks on undirected graphs we require the flow to fulfill the inequality $\sum_{u \in N(v)} f(u, v) \leq c(v)$ for all $v \in V$. For all variants, the *value* of a flow is defined by $|f| = \sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t)$. It was shown in [SGDJ04], that one can compute a flow in a network (G, s, t, c) with G being an undirected graph by computing a flow in the network $(\overleftrightarrow{G}, s, t, c')$ with $\overleftrightarrow{G}$ denoting the corresponding bidirected graph, and $c' \equiv c$ for vertex capacities, resp. $c'(u, v) = c(\{u, v\})$ for edge-capacities. Note that for unit vertex-capacities $c \equiv 1$ a maximum s-t flow in a network (G, s, t, c) is equivalent to a maximum collection of vertex-disjoint s-t paths.

Modifications on graphs. For two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, we denote by $G_1 \dot\cup G_2$ their disjoint union, i.e., the graph $(V_1 \dot\cup V_2, E_1 \dot\cup E_2)$. Further, we denote by $G_1 \bowtie G_2$ their (disjoint) join, i.e., the graph $(V_1 \dot\cup V_2, E_1 \dot\cup E_2 \dot\cup \{\{u, v\} \mid u \in V_1, v \in V_2\})$. We say that a graph H results from an undirected graph $G = (V, E)$ by *subdividing* an edge $\{u, v\} \in E$, if $V(H) = V \cup \{w\}$ and $E(H) = E \setminus \{\{u, v\}\} \cup \{\{w, u\}, \{w, v\}\}$ for a new vertex $w \notin V$. A graph H results from an undirected graph $G = (V, E)$ by *contracting* an edge $\{u, v\} \in E$, if H results from $G - \{u, v\}$ after adding a new vertex $w \notin V$ with neighborhood $N_H(w) = N_G(u) \cup N_G(v)$, i.e., in H the edge $\{u, v\} \in E$ is removed and the vertices u and v are merged together. For two graphs G and H, and a vertex $v \in V(H)$, we denote by $H[v \leftarrow G]$ the *substitution of G into v in H*, i.e., the graph obtained by replacing v in H with the graph G and giving each of its vertices the same neighborhood as v had.

2.3 Parameters

In this thesis, we will especially focus on the parameters *modular-width*, *clique-width*, and *twin-width*. For a detailed definition of these parameters, we refer to Section 3.1, Section 4.1, and Section 5.1. In this section, we will put these parameters into the context of other well-known parameters that we will shortly define now. See Figure 2.1 for a graphical summary of the relations between the different parameters. We stress that we can only cover a fraction of all well-known parameters.

Vertex-cover, twin-cover, and neighborhood-diversity. A vertex-cover in a graph $G = (V, E)$ is a set $X \subseteq X$ such that $G - X$ is an isolated graph. The vertex-cover number of a graph, denoted by $\mathsf{vc}(G)$ is the smallest $k \in \mathbb{N}$ such that there exists a vertex-cover X with $|X| = k$. As a graph with n vertices and a vertex-cover of size k contains at most $k(n-1)$ many edges, only sparse graphs can have a vertex-cover of low size. The parameters neighborhood-diversity [Lam12] and the twin-cover [Gan11] both generalize the parameter vertex-cover to dense graphs. For a graph $G = (V, E)$ we call an edge $e \in E$ a *twin edge* if its incident vertices are true twins. A subset $X \subseteq V$ is called a twin-cover if every edge $e \in E$ is either a twin edge or is incident to some vertex in X. The twin-cover of a graph G, denoted by $\mathsf{tc}(G)$, is the smallest k such that there is a twin-cover X of size k. This means that if there is a twin-cover X in a graph G it holds that each connected component in $G - X$ is a complete graph.[1] Thus, the twin-cover is a strict generalization of vertex-cover, and it holds that $\mathsf{tc}(G) \leq \mathsf{vc}(G)$ for all graphs G.

For a graph $G = (V, E)$ we call a partition of the vertices $T = \{T_1, T_2, \ldots, T_t\}$ of size $t \geq 1$ a *neighborhood decomposition* if for every $i \in [t]$ it holds that all pairs of vertices $u, v \in T_i$ are either true twins or false twins. The neighborhood-diversity of a graph G, denoted by $\mathsf{nd}(G)$, is the smallest $t \geq 1$, such that there exists a neighborhood decomposition of size t. Note that in a neighborhood decomposition it does not only hold that all vertices in a vertex set T_i have the same neighborhood outside T_i, but also that each T_i either induces a clique or an isolated graph in G, for each $i \in [t]$. Each graph of bounded vertex-cover does also have bounded neighborhood diversity, however, the neighborhood-diversity might be exponentially larger as it holds that $\mathsf{nd}(G) \leq 2^{\mathsf{vc}(G)} + \mathsf{vc}(G)$ [Lam12]. Since all cliques have constant neighborhood-diversity (and also twin-cover), one cannot bound the vertex-cover of a graph by any function depending on the neighborhood-diversity or twin-cover.

[1] This implication does not hold in the other direction as it is requested that all vertices in a connected component of $G - X$ must have a uniform neighborhood also to vertices in X.

Since the neighborhood decomposition is a special case of the modular decomposition, namely a modular decomposition into modules that corresponds to cliques or isolated graphs, the modular-width is a strict generalization of the neighborhood-diversity, while there exist graphs of constant modular-width with unbounded neighborhood-diversity and also unbounded twin-cover [GLO13]. Regarding the twin-cover $\mathsf{tc}(G)$ and the modular-width $\mathsf{mw}(G)$ of a graph is holds that $\mathsf{mw}(G) \leq 2^{\mathsf{tc}(G)} + \mathsf{tc}(G)$ [GLO13].

Tree-depth. The tree-depth of a graph $G = (V, E)$, also known as the vertex ranking number or elimination tree height, is the minimum height of a rooted forest F such that G is a subgraph of the closure $clos(F)$ of F [NdM06]. An analog definition of tree depth is the following recursive formula, where $G_1, \ldots, G_t$ denote the connected components of a graph G:

$$\mathsf{td}(G) = \begin{cases} 0 & \text{if } G = \emptyset \\ 1 + \min_{v \in V(G)} \mathsf{td}(G - v) & \text{if } G \text{ is connected and } |V(G)| \geq 1 \\ \max_{i=1}^{t} \mathsf{td}(G_i) & \text{otherwise} \end{cases}$$

A simple illustration of the above definition is also the following two-player game between a player and an adversary: The game starts with the graph G. The player can choose any vertex of G to get deleted. The adversary now must choose any connected component of the remaining graph and all other connected components (if existent) get deleted. Now it is again the player's turn to delete a vertex in the remaining graph and so on. The player's goal is to delete the whole graph as fast as possible while the adversary tries to prolong the game as much as possible. The tree-depth of a graph G is then the minimum number of turns such that there exists a winning strategy for the player, regardless of the actions of the adversary. Another equivalent definition of the tree-depth of a graph via operations is given in Section 6.1.

For any graph that contains a vertex-cover of size k, it holds that the tree-depth is at most $k+1$, while the vertex-cover cannot be bounded by any function of the tree-depth. In comparison to the parameters neighborhood-diversity, twin-cover, and modular-width, we observe that for a complete graph K_n with $n \in \mathbb{N}$ it holds that $\mathsf{td}(K_n) = n$. On the other side, the class of all subdivided stars, i.e. all graphs that emerge from a star $K_{1,n}$ after subdividing each edge, has tree-depth at most three but unbounded modular-width. Thus, the parameter tree-depth is incomparable to modular-width, neighborhood-diversity, and twin-cover.

Tree-width. A *tree-decomposition* of an undirected graph $G = (V, E)$ is a pair (X, T) where $T = (I, A)$ is a tree and $X = \{X_i \mid i \in I\}$ is a collection of subsets of V, such that

- $\cup_{i \in I} X_i = V$,
- For each edge $\{u, v\} \in E$ there exists an $i \in I$ with $u, v \in X_i$,
- For each vertex $v \in V$ the set $\{i \in I \mid v \in X_i\}$ of nodes in T forms a subtree of T.

The *width* of a tree-decomposition (X, T) is $\max_{i \in I} |X_i| - 1$. The *tree-width* of a graph G, denoted by $\mathsf{tw}(G)$, is the minimum width among all possible tree-decompositions of G. As an example, the tree-width of any tree is exactly one by setting $X = E$, while the tree-width of a complete graph K_n is $n - 1$ for all $n \in \mathbb{N}$. By the famous theorem by Courcelle [Cou90], one can decide every graph property that can be defined in MSO_2 logic in linear time on graphs of bounded tree-width.[2]

[2] In the MSO_2 logic one may define the property with quantifications over sets of vertices, sets of edges, and an incidence relation between vertices and edges. In the weaker MSO_1 logic, only quantifications over sets of vertices are allowed.

The tree-width of a graph is at most the tree-depth of a graph, i.e., it holds that $\mathsf{tw}(G) \leq \mathsf{td}(G) - 1$ [NdM12]. Indeed, it holds that even the greater parameter *path-width* $\mathsf{pw}(G)$ which can be defined in the same way as the parameter tree-width but with the constraint that T needs to be a path instead of a tree, is at most the tree-depth, i.e., it holds that $\mathsf{tw}(G) \leq \mathsf{pw}(G) \leq \mathsf{td}(G) - 1$. On the other side, it holds that $\mathsf{td}(G) \leq \mathsf{tw}(G) \log n$ for any graph G with n vertices [NdM12].

For clique-width, one can show that $\mathsf{cw}(G) \leq 3 \cdot 2^{\mathsf{tw}(G)-1}$ while for any k there exist a graph G with $\mathsf{tw}(G) = k$, but $\mathsf{cw}(G) \geq 2^{\lfloor \mathsf{tw}(G)/2 \rfloor - 1}$ [CR05]. On the other side, the tree-width (and branch-width) can not be bounded by a function depending on the clique-width, as for every complete graph K_n it holds that $\mathsf{cw}(K_n) = 2$, but $\mathsf{tw}(K_n) = n - 1$.

Branch-width. A branch-decomposition of an undirected graph $G = (V, E)$ is a tree T with vertices of either degree one or degree three, with exactly $|E|$ many leaves and a one-to-one mapping of the set of edges E to the set of leaves of T. Note that for each edge $e \in E(T)$, the graph $T - e$ consists of exactly two connected components. The *order* of an edge $e \in E(T)$ is the number of vertices $v \in V(G)$ such that v is incident to at least one edge corresponding to a leaf in both connected components in $T - e$. The width of T is the maximum order over all edges in T. The *branch-width* of a graph G, denoted by $\mathsf{bw}(G)$ is the minimum width over all possible branch decompositions of G. It holds that $\mathsf{bw}(G) \leq \mathsf{tw}(G) + 1 \leq \lfloor 3/2\, \mathsf{bw}(G) \rfloor$, i.e., the branch-width and the tree-width of a graph are always withing a constant factor of each other [RS91].

We remark that for the definition of the parameters *boolean-width* and *rank-width*, we will consider a very similar decomposition, however, with T having exactly $|V|$ many vertices and there is a one-to-one mapping of the set of vertices V to the set of leaves in T.

Boolean-width and rank-width. For the definition of the parameters boolean-width and rank-width of an undirected graph $G = (V, E)$, we consider a so-called *decomposition tree* of G that is a tree T with vertices of either degree one or degree three, with exactly $|V|$ many leaves and a one-to-one mapping of the set of vertices V to the set of leaves of T. Removing an edge $e \in E(T)$ results in two subtrees of T, defining a partition of V into two sets A and $V \setminus A$ with A is the set of vertices corresponding to the leaves of one of the two subtrees in $T - e$. Such a partition of the vertices into two sets is also called a *cut*. Consider now a function $f \colon 2^V \to \mathbb{R}$ that is symmetric, i.e., $f(A) = f(V \setminus A)$ for all $A \subseteq V$. We can now define the *f-width* of such a decomposition tree T as the maximum value of $f(A)$ over all cuts $\{A, V \setminus A\}$ of G defined by a removal of an edge in T. The f-width of a graph G is the minimum f-width over all possible decomposition trees T.

To define the *boolean-width* we consider the function $f = \mathit{bool\text{-}dim}$ that is defined by $\mathit{bool\text{-}dim}(A) = \log_2(|U(A)|)$ where $U(A) = \{Y \subseteq V \setminus A \mid \exists X \subseteq A \wedge Y = N(X) \cap V \setminus A\}$ denotes the set of unions of neighborhoods of A across the cut $\{A, V \setminus A\}$ [BTV11].

For *rank-width* we consider the function $f = \mathit{rank}$ with $\mathit{rank}(A)$ denotes the rank of the $|A| \times |V \setminus A|$ 0-1 matrix M_A over the binary field where $M_A[i][j]$ is equal to 1 if the i-th vertex in A is adjacent to the j-th vertex in $V \setminus A$, and 0 otherwise [OS06].

For a graph of clique-width at most k, also the rank-width and the boolean-width is at most k, while for graphs of rank-width k or boolean-width k the clique-width can only be bounded by 2^k, i.e., it holds that $\log(\mathsf{cw}(G)) \leq \mathsf{rw}(G) \leq \mathsf{cw}(G)$ [Oum08] as well as $\log(\mathsf{cw}(G)) \leq \mathsf{boolw}(G) \leq \mathsf{cw}(G)$ [BTV11]. The boolean-width can be polynomially bounded by the rank-width, while the boolean-width might be much smaller than the rank-width, i.e., it holds that $\log(\mathsf{rw}(G)) \leq \mathsf{boolw}(G) \leq 1/4\, \mathsf{rw}(G)^2 + \mathcal{O}(\mathsf{rw}(G))$ [BTV11].

In comparison to tree-width and branch-width it holds that $\mathsf{rw}(G) \leq \mathsf{tw}(G)+1$ and $\mathsf{rw}(G) \leq \mathsf{bw}(G)$ [Oum08] as well as $\mathsf{boolw}(G) \leq \mathsf{tw}(G)+1$ and $\mathsf{boolw}(G) \leq \mathsf{bw}(G)$ [BTV11]. On the other side, the tree-width (and branch-width) can not be bounded by a function depending on the rank-width or boolean-width.

Bonnet et al. [BKTW20] showed that every graph of boolean-width k has twin-width at most $2^{k+1}-1$, i.e., it holds that $\mathsf{tww}(G) \leq 2^{\mathsf{boolw}(G)+1}-1$. This implies that the class of bounded twin-width graphs contains all graph that has bounded parameter value for all considered parameters in this section, however, the twin-width might be exponentially larger. On the other hand, one cannot bound any considered parameter in this section by a function depending on the twin-width (for example the class of all grids is of constant twin-width, while of unbounded clique-width [GR99b]).

Shrub-depth. The parameter *shrub-depth* was introduced by Ganian et al. [GHN$^+$12] as a dense analog of the parameter tree-depth and as a depth-variant of the parameter clique-width. The shrub-depth is related to the parameter clique-width similar as the parameter tree-depth is related to tree-width. To define the shrub-depth of a graph G, we first need to define the *tree-model* of a graph. Let m and d be non-negative integers. A tree-model for a graph G of depth d with m colors is a pair (T, S) where T is a rooted tree of depth d and $S \subseteq [m]^2 \times [d]$ such that (1) the length of each root-to-leaf path in T is exactly d; (2) the set of leaves of T corresponds to the set of vertices of G; (3) each leaf in T is colored by a color in $[m]$; (4) for any $i, j \in [m]$ and $k \in [d]$ it holds that $(i, j, k) \in S$ if and only if $(j, i, k) \in S$ (called symmetry in colors); (5) for any two vertices $u, v \in V(G)$, it holds that $\{u, v\} \in E(G)$ if and only if $(i, j, k) \in S$ where i and j refers to the colors of u and v in T and k denotes the distance from u or v to the least common ancestor of them in T. Thus, condition (5) expresses that the existence of an edge between u and v in G solely depends on the color of the corresponding leaves in T and the distance between u and v in T. The set S is called the *signature* of the tree-model. The class of all graphs having a tree-model of depth d with m colors is denoted by $\mathcal{TM}_m(d)$. A class $\mathcal{G}$ of graphs has shrub-depth d if there exists an integer m such that $\mathcal{G} \subseteq \mathcal{TM}_m(d)$.

As an example, the class of all stars $K_{1,n}$ has shrub-depth one as this class is contained in $\mathcal{TM}_2(1)$. The class of all graphs with bounded neighborhood-diversity has shrub-depth at most one while the class of all graphs with bounded twin-cover has shrub-depth at most two. Note that the definition of shrub-depth is asymptotic and only makes sense for infinite graph classes. Any finite graph with n vertices is always contained in $\mathcal{TM}_n(1)$, and thus, any finite graph class has shrub-depth at most one.

It holds that the class of graphs with tree-depth at most d is also of shrub-depth at most d, while the converse statement is not true in general [GHN$^+$19]. Further, for every graph $G \in \mathcal{TM}_m(d)$ it holds that $cw(G) \leq m$, thus, each graph class that is of bounded shrub-depth is also of bounded clique-width. Since the class of all cographs is not of bounded shrub-depth, shrub-depth is incomparable to modular-width.

Every MSO_1 definable property for graphs with n vertices can be solved in linear time on graph classes of bounded shrub-depth. In particular, for a property φ defined in MSO_1, one can decide whether φ holds in G in time $\mathcal{O}(f(\varphi) \cdot |n|)$. The same results do also hold for any graph of bounded clique-width. However, for a bounded clique-width graph, the function f is non-elementary, i.e., it grows with a tower of exponentials whose height depends on φ. On graph classes of bounded shrub-depths, the height of exponentials in the function f does not depend on the formula.

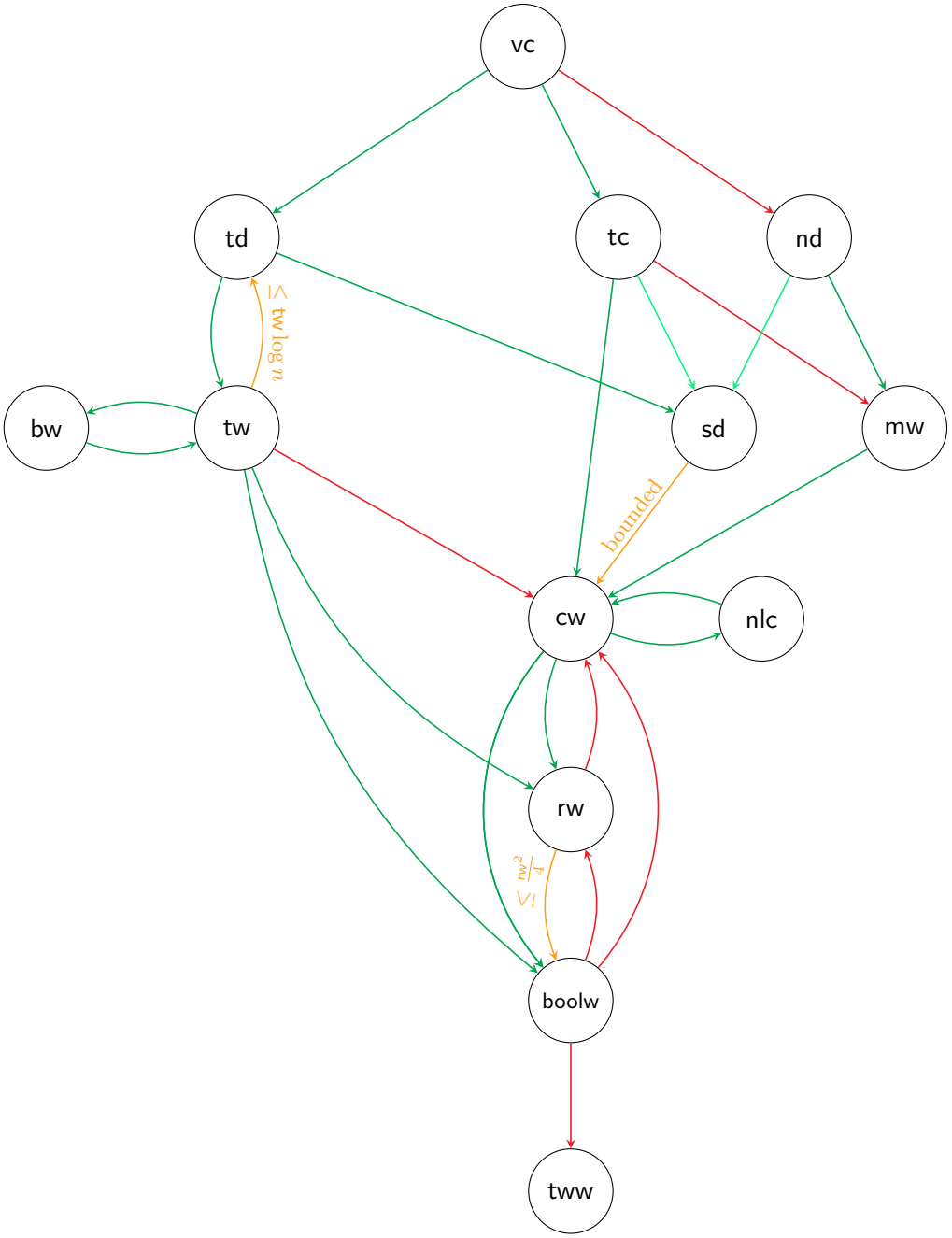


Figure 2.1: Diagram of the relationships between the parameters considered in Section 2.3. A green arc from a parameter p to a parameter q indicates that the parameter q is linearly bounded by the parameter p, while a red arc indicates that the value of parameter q is exponentially bounded by the value of parameter p. The dependencies on yellow arcs are explicitly written down. Arcs incident to the node shrub-depth consider graph classes, e.g., if a graph class has tree-depth at most k, this graph class also has shrub-depth at most k. The light green arrows indicate that if a graph class has twin-cover or neighborhood-diversity at most k this graph class has shrub-depth at most two. If for two parameters p and q there is no p-q path in the diagram, then one cannot bound q by any function of p.

2.4 Considered Problems

In this section, we consider related work concerning problems for which we will give efficient parameterized algorithms. All of the considered problems can be solved (unparameterized) in polynomial time.

Maximum matching and maximum b-matching. The MAXIMUM MATCHING problem is one of the longest established problems with the first polynomial-time algorithm from 1965 by Edmonds [Edm65]. In 1980, Micali and Vazirani [MV80, Vaz20] presented an algorithm with a running time of $\mathcal{O}(m\sqrt{n})$, which is still the best algorithm known for general graphs. A minor improvement for dense graphs was given by Goldberg and Karzanov [GK04] with an algorithm running in time $\mathcal{O}(m\sqrt{n}\log(n^2/m)/\log(n))$. Using Gaussian Elimination, Mucha and Sankowski [MS04] developed a randomized algorithm with running time $\mathcal{O}(n^\omega)$, where ω is the matrix multiplication exponent.[3].

There are several algorithms for MAXIMUM MATCHING on special graph classes, e.g., an $\mathcal{O}(n \log n + m)$-time algorithm on interval graphs, circular arc graphs [LR93], or co-interval graphs [Gar03] and even linear-time algorithms for, e.g., cographs [YY93], strongly chordal graphs (given the strongly perfect elimination order) [DK98], chordal bipartite graphs [Cha96], or co-comparability graph [MNN16]. For planar graphs, there is a randomized algorithm running in time $\mathcal{O}(n^{\omega/2})$ [MS06]. Yuster and Zwick [YZ07] showed that there is an algorithm for finding a maximum matching in H-minor free graphs with a running time of $\mathcal{O}(n^{3\omega/(\omega+3)}) \subseteq \mathcal{O}(n^{1.326})$, for every fixed graph H.

On the parameterized side, Fomin et al. [FLS+18] presented a randomized algorithm parameterized by the tree-width tw of a graph with a running time of $\mathcal{O}(\mathsf{tw}^4 \cdot n \log n)$. Iwata et al. [IOO18] described an $\mathcal{O}(\mathsf{td} \cdot m)$-time deterministic algorithm for graphs of tree-depth at most td. This also implies an $\mathcal{O}(\mathsf{tw} \cdot m \log n)$-time algorithm for the parameter tree-width, since every graph of tree-width tw has tree-depth at most $\mathsf{tw}\log(n)$ [BGHK95]. Parameterized by the modular-width h of the graph, Coudert et al. [CDP19] presented a an algorithm running in time $\mathcal{O}(\mathsf{h}^4 \cdot n + m)$. Recently, Ducoffe [Duc22a] developed an almost linear-time algorithm on bounded clique-width graphs, assuming a corresponding k-expression is given. Since this algorithms does use Courcelle's theorem [CMR00] as a subroutine[4], the achieved running time is $\mathcal{O}(f(k) \cdot (n+m)^{1+o(1)})$ for some computable function f and k denoting the clique-width of the input graph. Mertzios et al. [MNN16] showed that one can solve maximum matching in time $\mathcal{O}(k(n+m))$ where k denotes the vertex cover number, the feedback vertex number, the feedback edge number, the distance to chain graphs, or the distance to co-comparability graphs. In a follow-up paper [MNN20] they also gave algorithms with a time complexity of $\mathcal{O}(k^3+n+m)$ for the vertex cover number, of $\mathcal{O}(kn+2^{\mathcal{O}(k)})$ for the feedback vertex number, of $\mathcal{O}(k^{1.5}+n+m)$ for the feedback edge number, and of $\mathcal{O}(k^3+n+m)$ for the distance to chain graphs.

For several other parameters, Hegerfeld and Kratsch [HK19] showed that all algorithms for maximum matching that follow the phase framework of Hopcroft and Karp [HK73] obliviously adapt to the structure resulting in improved running times. For example, they showed that those algorithms only take time $\mathcal{O}(m)$ on bounded tree-depth graphs, cluster graphs, or star forests, and time $\mathcal{O}(km)$ with k denoting either the independence number, the neighborhood diversity,

[3] It is known that $2 \leq \omega < 2.3728639$ due to Le Gall [Gal14]. By definition of ω the running time is in fact $\mathcal{O}(n^{\omega+o(1)})$; adopting a common abuse of notation we use exponent ω for brevity.

[4] Note that one cannot express MAXIMUM MATCHING in MSO_1 logic and hence, Courcelle's theorem cannot be applied directly.

or if it holds that the minimum degree is at most $n-k$. For graphs of modular-width at most mw they additionally considered the modular-depth, i.e., the depth of the modular-decomposition tree, and bound the running time of algorithms using the phase framework by $\mathcal{O}((c \cdot \mathsf{mw})^{\mathsf{md}} m)$ for a constant c. We point out that the algorithms by Micali and Vazirani [MV80], Blum [Blu90], or Goldberg and Karzanov [GK04], all use the phase framework of Hopcroft and Karp.

For the more general MAXIMUM b-MATCHING, Anstee [Ans87] developed an algorithm with a running time of $\mathcal{O}(n^2 m\hat{b})$ where $\hat{b}$ is the maximum b-value of any vertex in the graph. Recently, this result was improved by Gabow [Gab18] by an algorithm with a running time of $\mathcal{O}(\min\{b(V), n\log n\}(m + n\log n))$, where $b(V)$ denotes the sum of all b-values. Note that by this algorithm one can bound the running time independently of the actual b-values of the vertices.

Independently of our work, Ducoffe and Popa [DP21a] also used the more general b-matching problem to compute a maximum matching in time $\mathcal{O}((k\log^2 k)(n+m)\log n)$ where k denotes the *split-width* of the graph, a parameter that is slightly more general than the parameter modular-width. To our best knowledge, this work and the presented work in Section 3.2 are the first and only algorithms that use the computation of a b-matching on some (much) smaller graph to compute eventually a maximum matching. Both algorithms can also be used to compute a maximum b-matching in the same running time.

All-pairs shortest paths. In this thesis, we will solely consider the vertex-weighted variant of ALL-PAIRS SHORTEST PATHS. For (edge-weighted) ALL-PAIRS SHORTEST PATHS (APSP) there is the textbook algorithm from 1962 of Floyd and Warshall [Flo62, War62] with a running time of $\mathcal{O}(n^3)$. Following this work, Fredman [Fre76] achieved the first subcubic algorithm, running in time $\mathcal{O}(n^3 \log^{1/3}\log n/\log^{1/3} n)$. Chan [Cha10] and Han and Takaoka [HT16] both achieved a running time of $\mathcal{O}(n^3/\log^2 n)$ (omitting $poly(\log\log n)$ factors). Recently, Williams [Wil18a] solved APSP in randomized time $\mathcal{O}(n^3/2^{\Omega(\log n)^{1/2}})$. For sparse graphs, Pettie and Ramachandran [PR05, Pet04] get a running time of $\mathcal{O}(n^2\alpha(n,m) + mn)$ on undirected and $\mathcal{O}(n^2\log\log n + nm)$ on directed graphs. It is conjectured [Wil18b, WW18] that one cannot solve ALL-PAIRS SHORTEST PATHS in $\mathcal{O}(n^{3-\varepsilon})$ time for some $\varepsilon > 0$ on graphs with edge weights in $\{-n^c, \ldots, n^c\}$ and no negative cycles for large enough c.

For the vertex-weighted case, we stress that one can solve VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS by any algorithm operating on edge-weights instead; by shifting the vertex weights to the outgoing edges of each vertex in the directed case, cf. Section 4.2, or shifting the vertex weights to all incident edges in the undirected case.[5] Using fast matrix multiplication, the currently fastest algorithm for the vertex-weighted variant runs in $\mathcal{O}(n^{2.842})$ time by Yuster [Yus09], which slightly improves the $\mathcal{O}(n^{2.844})$-time algorithm by Chan [Cha10]. All subcubic algorithms for the vertex-weighted variant of APSP exploit fast matrix multiplication. There is no truly subcubic *combinatorial* algorithm known for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS.

For unweighted and undirected graphs, Seidel [Sei95] was the first to show how to solve ALL-PAIRS SHORTEST PATHS in time $\mathcal{O}(n^\omega \log n)$, using the fact that for the adjacency matrix A of a graph G it holds that $A^k[i,j] = 1$ if and only if there is a shortest path from i to j in G of length at most k. For unweighted and directed graphs, Zwick [Zwi02] gave an algorithm with time complexity of $\mathcal{O}(n^{2.53})$ using *rectangular* matrix multiplication.

[5] This means that each edge in the resulting graph has cost equal to the sum of the vertex costs of the two incident vertices, and thus, in each s-t path the costs of the internal nodes are doubled. However, the original costs can be retraced in constant time.

There are some subcubic algorithms known for APSP on special graph classes. Atallah et al. [ACL95] provided a linear-time algorithm for computing single-source shortest paths in a weighted interval or circular-arc graph with non-negative weights, assuming that the model of the graph is given, i.e, the actual weighted intervals or circular-arcs and the sorted list of the interval endpoints. The running time of this algorithm is $\mathcal{O}(n)$ with n denoting the number of intervals or circular-arcs in the graph, i.e., the number of vertices. This implies a quadratic-time algorithm for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS with non-negative vertex weights, which is asymptotically optimal as the output of this problem is of size $\Omega(n^2)$. Note that the class of all interval graphs[6] has unbounded clique-width [GR00] and thus, unbounded modular-width. There are several other graph classes that allow algorithms with a running time of $\mathcal{O}(n^2)$, e.g., for unweighted strongly chordal graphs [BR96], unweighted chordal bipartite graphs [CWC99], or planar graphs [Fre87].

For parameterized algorithms, ALL-PAIRS SHORTEST PATHS got very little attention. For uniform disk graphs with non-negative vertex weights, induced by point sets of bounded density within a unit square, Lingas and Sledneu [LS12] showed how to solve APSP on such graphs in time $\mathcal{O}(\sqrt{r}n^{2.75})$, where r denotes the radius of the disk around the vertices in a unit square. Following our $\mathcal{O}(\mathsf{cw}^2\, n^2)$-time algorithm for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS on graphs of clique-width at most cw (cf, Section 4.2), Ducoffe [Duc22b] presented an $\mathcal{O}(\mathsf{cw}(n \log n)^2)$-time algorithm, however, considering only non-negative vertex weights. On graphs of twin-width at most tww, Bonnet et al. [BGK$^+$21b] recently described an algorithm running in time $\mathcal{O}(\mathsf{tww}\, n^2 \log n)$ for solving unweighted APSP, by showing that every graph of twin-width at most tww admit a so-called interval biclique partition of size $\mathcal{O}(\mathsf{tww} \cdot n)$.

For the related problem of computing the diameter of an unweighted graph that is the length of a longest shortest path, Ducoffe [Duc22b] developed an algorithm running in time $\mathcal{O}(2^{\mathcal{O}(\mathsf{cw})}(n+m)^{1+\varepsilon})$ for any $\varepsilon > 0$ that is optimal under SETH. Bentert and Nichterlein [BN18] also considered the problem of computing the diameter of a graph, parameterized by several parameters. Shapira et al. [SYZ11] considered some variants of APSP, namely the ALL-PAIRS BOTTLENECK PATHS, where the cost of a path is defined by the bottleneck weight on a path, and provided an algorithm of time $\mathcal{O}(n^{2.575})$ for vertex-weighted graphs. Iwata et al. [IOO18] have considered in their work the problem NEGATIVE CYCLE DETECTION where one needs to test if a given weighted graph admits a cycle of negative cost and obtained a $\mathcal{O}(k(m + n \log n))$-time algorithm for NEGATIVE CYCLE DETECTION, where k denotes the tree-depth of the input graph.

Triangle Counting. For graph G with n vertices and m edges, the naive algorithms that iterates over all triples of vertices can be easily realized in time $\mathcal{O}(n^3)$. Another simple approach is to iterate over all edges and check for each vertex if they form a triangle in G, which can be done in time $\mathcal{O}(nm)$. For sparse graphs, one can also solve TRIANGLE COUNTING in time $\mathcal{O}(m^{1.5})$[IR78]. Williams and Williams [WW18] showed that even for the simpler TRIANGLE DETECTION problem, where only (non-)existence of a single triangle needs to be reported, it is unlikely to admit a *combinatorial* truly subcubic running time; they proved that if one could detect if a graph has a triangle in time $\mathcal{O}(n^{3-\varepsilon})$ for $\varepsilon > 0$ then one could also solve the following three problems in the same running time via combinatorial truly subcubic reductions: Boolean matrix multiplication, listing up to $n^{3-\delta}$ triangles in a graph for a constant $\delta > 0$, or verifying the correctness of a matrix product over the Boolean semiring, while for none of these problems a truly subcubic combinatorial algorithm is known and would be a huge breakthrough.

[6] It is shown in [GR00] that even the class of unit interval graphs, in which each interval needs to have equal length, is not of bounded clique-width.

Using fast matrix multiplication, one can solve TRIANGLE COUNTING in time $\mathcal{O}(n^\omega)$ [AYZ97]. The fastest known algorithm for counting triangles in sparse graphs is the AYZ algorithm due to Alon, Yuster, and Zwick [AYZ97], which runs in time $\mathcal{O}(m^{\frac{2\omega}{\omega+1}})$ ($\mathcal{O}(m^{1.41})$ for $\omega < 2.373$). There is no algorithm with running time $\mathcal{O}(n^{\omega-\varepsilon})$ for $\varepsilon > 0$ known.

In the realm of parameterized algorithm, Chiba and Nishizeki [CN85] gave an $\mathcal{O}(m \cdot d)$ time algorithm for TRIANGLE COUNTING, where d denotes the *degeneracy*[7] of the graph. Kopelowitz et al. [KPP16] showed that under the 3-SUM conjecture this running time is essentially optimal, i.e., only savings by polylogarithmic factors are possible; which was also achieved by Kopelowitz et al. [KPP15]. Coudert et al. [CDP19] gave a faster algorithm for graphs of bounded clique-width cw, running in time $\mathcal{O}(\mathsf{cw}^2(n+m))$. Bentert et al. [BFNN19] have studied TRIANGLE ENUMERATION under various parameters including feedback edge number, distance to d-degenerate graphs, and clique-width. The latter one outputs all triangles in time $\mathcal{O}(\mathsf{cw}^2\, n + n^2 + \#T)$ where $\#T$ denotes the number of triangles in G. For the parameters feedback edge number and distance to d-degenerate they also provide so-called enum-advice kernels of at most $9k$ vertices in time $\mathcal{O}(n+m)$ for graphs with feedback edge number k resp. with at most $k+2^k+3$ vertices in time $\mathcal{O}(n \cdot d \cdot (k+2^k))$ for graphs with distance to d-degenerate at most k. Parameterized by the size $|C|$ of a vertex cover, Green and Bader [GB13] described an algorithm with running time $\mathcal{O}(|C| \cdot \Delta_C^2)$, where $\Delta_C \leq n$ denotes the maximum degree in G of vertices in the vertex cover C.

Edge- and vertex-disjoint paths. The very recent breakthrough result of an almost linear time algorithm $\mathcal{O}(m^{1+o(n)} \log U)$ for maximum s-t flow in a graph G with integral capacities in $[U]$ by Chen et al. [CKL+22] is not considered in this work.[8] We will discuss at the end of Section 3.8 the implications of this faster algorithm to our work. Previously, the currently best maximum flow algorithm was due to Orlin [Orl13] and runs in time $\mathcal{O}(nm)$. Using a flow algorithm, one can determine the number of edge- or vertex-disjoint s-t paths in a graph. In the unweighted and undirected case, one can do a bit better and can compute the number of edge-disjoint paths in time $\mathcal{O}(n^{\frac{3}{2}} m^{\frac{1}{2}})$ using an algorithm due to Goldberg and Rao [GR99a]. Finding a global minimum edge cut in an unweighted and undirected graph can be solved by the simple randomized contraction algorithm by Karger [Kar93]. An extension of this algorithm by Karger and Stein [KS96] achieves an error probability of $\mathcal{O}(1/n)$ with a running time of $\mathcal{O}(n^2 \log^3 n)$. This result was later improved by Karger [Kar00] by a randomized algorithm running in time $\mathcal{O}(n^2 \log n)$ resp. $\mathcal{O}(m \log^3 n)$. A deterministic algorithm due to Gabow [Gab95] runs in time $\mathcal{O}(m + \lambda^2 n \log(n/\lambda))$ where λ denotes the edge-connectivity of the graph (which is upper-bounded by the minimum degree δ, so $\lambda \leq \delta \leq 2m/n$) that coincide with the value of a global minimum cut in unweighted graphs; this can be seen as a parameterized algorithms with parameter λ. Finding a global minimum edge cut with weights on the edges in an undirected graph can be done in time $\mathcal{O}(nm + n^2 \log n)$ due to Stoer and Wagner [SW97].

Via a standard reduction between edge-capacitated and vertex-capacitated flows[9], one can also solve vertex-capacitated s-t flow in $\mathcal{O}(nm)$ time. For the weighted global vertex cut, Henzinger et al. [HRG00] gave an algorithm running in time $\mathcal{O}(\kappa_1 nm \log(n^2/m))$, where κ_1 denotes the vertex connectivity if vertex capacities are ignored.

[7] A d-degenerate graph is an undirected graph in which every subgraph has a vertex of degree at most d. The degeneracy of a graph is the smallest d such that G is d-degenerate.

[8] At the time of this write-up, this publication is available as a preprint only and is not yet peer-reviewed.

[9] For the reduction, split every vertex v into two vertices v_{in} and v_{out} connected via an edge of capacity equal to the original vertex capacity of v, and let all incoming vertices of v point to v_{in} and let all outgoing edges of v start at v_{out}.

Part II

Parameterized Algorithms

3

Algorithms Parameterized by the Modular-Width of the Input

In this chapter, we study the influence of the graph parameter modular-width on the time complexity for optimally solving well-know tractable problems. The modular-width is the maximum number of children of a prime node in a modular-decomposition tree and roughly measures (as clique-width and twin-width) the distance of a graph from being a cograph, which are exactly those graphs whose modular decompositions do not have any prime nodes. Unlike most other parameters, one can compute the unique modular-decomposition tree in linear time [TCHP08]. We stress that out of the considered parameters in Section 2.3, only the parameters modular-width and neighborhood-diversity, a specialization of modular-width, can be computed in linear time, while all other considered parameters are NP-hard to compute.

We will present efficient algorithms parameterized by the modular-width of the input for fundamental problems such as MAXIMUM MATCHING, VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, TRIANGLE COUNTING, as well as several edge- and vertex-connectivity problems, e.g., MAXIMUM s-t VERTEX-CAPACITATED FLOW. For MAXIMUM MATCHING, we improve the running time from $\mathcal{O}(\mathsf{h}^4 n + m)$ to $\mathcal{O}(\mathsf{h}^2 \log \mathsf{h}\, n + m)$ with n, m, resp. h denoting the number of vertices, the number of edges, resp. the modular-width of the graph. We follow the same natural recursive approach as in previous work, i.e., computing optimal solutions in a bottom-up fashion on the modular decomposition tree. Unlike Coudert et al. [CDP19] (SODA '18), however, we do not seek to use the structure of modules to speed up the computation of augmenting paths, starting from an union of maximum matchings for the child modules. Instead, we simplify the current graph, while retaining the same maximum matching size, such that the found solutions can be encoded into vertex capacities in a graph with at most $3\,\mathsf{h}$ vertices. This allows us to forget the maximum matchings for the modules and instead of augmenting paths it suffices to find a maximum b-matching subject to vertex capacities; using an $\mathcal{O}(\min\{b(V), n \log n\} \cdot (m + n \log n)) = \mathcal{O}(n^3 \log n)$ time algorithm due to Gabow [Gab18] then

yields the claimed running time, where $b(V)$ denotes the total capacity of all vertices.[1] Our algorithm for MAXIMUM MATCHING easily generalizes to computing maximum b-matchings in the same time $\mathcal{O}(\mathsf{h}^2 \log \mathsf{h}\, n + m)$. By a different summation of the running time, one can also bound the time by $\mathcal{O}((\mathsf{h} \log \mathsf{h}) \cdot (m + n \log \mathsf{h}))$. For large total capacity $b(V)$, Gabow's algorithm runs in time $\mathcal{O}((n \log n) \cdot (m + n \log n))$, which matches our running time for graphs with worst-case modular-width of $\mathsf{h} \in \Theta(n)$. Thus, when capacities are large, our algorithm interpolates smoothly between linear time $\mathcal{O}(n + m)$ for $\mathsf{h} \in \mathcal{O}(1)$ and the running time of the best unparameterized algorithm for $\mathsf{h} \in \Theta(n)$; i.e., it is an *adaptive algorithm* and already $\mathsf{h} = o(n)$ gives an improved running time. Such adaptive algorithms (for other problems and parameter) were also considered by Iwata et al. [IOO18]. For MAXIMUM MATCHING, the comparison with the $\mathcal{O}(m\sqrt{n})$ time algorithm of Micali and Vazirani [MV80] is of course less favorable, but still yields a fairly large regime for h where we get a faster algorithm. Throughout, we obtain efficient parameterized algorithms of running times $\mathcal{O}(f(\mathsf{h})n+m)$, $\mathcal{O}(n+f(\mathsf{h})m)$, or $\mathcal{O}(f(\mathsf{h})+n+m)$ for graphs of modular-width h and polynomial functions f, resp. with an additional addend $+\mathcal{O}(n^2)$ for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS for which the output is already of quadratic size. For several problems we give adaptive algorithms, e.g., for TRIANGLE COUNTING that can be solved in time $\mathcal{O}(\mathsf{h}^{\omega-1}\, n + m)$ or for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS for which we present an $\mathcal{O}(\mathsf{h}^2\, n + n^2)$-time combinatorial algorithm[2] and an $\mathcal{O}(\mathsf{h}^{1.842}\, n + n^2)$-time algorithm using fast matrix multiplication; meaning that their running times match the best unparameterized algorithms for worst-case modular-width of $\mathsf{h} \in \Theta(n)$.

For the edge- and vertex-disjoint paths problems we obtain the following running times: MAXIMUM s-t VERTEX-CAPACITATED FLOW in $\mathcal{O}(\mathsf{h}^3 + n + m)$ time; GLOBAL VERTEX-CAPACITATED MIN CUT in $\mathcal{O}(\mathsf{h}^2 \log \mathsf{h}\, n + m)$ time; EDGE-DISJOINT s-t PATHS in $\mathcal{O}(\mathsf{h}^3 + n + m)$ time; and UNWEIGHTED GLOBAL MIN CUT in $\mathcal{O}(\mathsf{h}^3 + n + m)$ time. Our results for vertex-disjoint paths problems generalizes to vertex-capacitated flows and vertex-weighted global minimum cuts. Again, as done for MAXIMUM b-MATCHING, one can obtain different bounds for the running time by slightly different summations. For example, the running time for MAXIMUM s-t VERTEX-CAPACITATED FLOW can also be bounded by $\mathcal{O}(\mathsf{h}\, m + n)$, meaning that the algorithm is never worse than the optimal unparameterized algorithm and outperforms it already for $\mathsf{h} = o(n)$. To ease the computation of the running times, we will present a general running time theorem that one can apply to all presented algorithms to analyze the running time.

It is easy to see that there is little use for modular-width for most edge-weighted/capacitated problems because it suffices to solve them on cliques, which have modular-width equal to zero, see also the remarks concerning this matter in Section 3.8. Note that standard transformations between different variants of path- and flow-type problems do not apply here directly because they affect the modular-width of the graph. The running times for edge- and vertex-disjoint paths problems are linear in the graph size and only have an additive contribution in terms of the modular-width, because at most one involved computation (on a prime node) is needed. These also give rise to linear-time kernelization-like algorithms that return an equivalent instance of size $poly(\mathsf{h})$, which is the one instance that one would run some other algorithm on (i.e., the only source of non-linear time). Such results (for other problems) have also been observed by Coudert et al. [CDP19] or Giannopoulou et al. [GMN17]. We stress that *any* algorithm of running time $\mathcal{O}(f(k) + n + m)$, for some parameter k, implies a linear-time kernelization: Run

[1] The obvious upper bound of $\mathcal{O}(\mathsf{h}^3 \log \mathsf{h}\, n + m)$ of applying Gabow's algorithm on each prime node can be improved by a slightly more careful summation; the same applies to the other results.

[2] The term "combinatorial" algorithms do not have a formal definition. Intuitively, a combinatorial algorithm is also practically efficient, i.e, the hidden constants in the running time are low, cf. [WW18]. Thus, a "combinatorial" algorithm especially forbids the use of fast matrix multiplication.

Problem	Best unparameterized	Our result
MAXIMUM MATCHING	$\mathcal{O}(m\sqrt{n})$ [MV80]	$\mathcal{O}(\mathsf{h}^2 \log \mathsf{h}\, n + m)$
MAXIMUM b-MATCHING [3]	$\mathcal{O}((n \log n) \cdot (m + n \log n))$ [Gab18]	$\mathcal{O}((\mathsf{h} \log \mathsf{h}) \cdot (m + n \log \mathsf{h}))$ or $\mathcal{O}(\mathsf{h}^2 \log \mathsf{h}\, n + m)$
VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS	$\mathcal{O}(n^3)$ [Flo62, War62] or $\mathcal{O}(mn \log \alpha(m,n))$ [Joh77] or $\mathcal{O}(n^{2.842})$ [Yus09]	$\mathcal{O}(\mathsf{h}^2 n + n^2)$ or $\mathcal{O}(m\,\mathsf{h} \log \alpha(m,\mathsf{h}))$ or $\mathcal{O}(\mathsf{h}^{1.842} n + n^2)$
TRIANGE COUNTING	$\mathcal{O}(n^\omega)$ [SW05] or $\mathcal{O}(m^{\frac{2\omega}{\omega+1}}) = \mathcal{O}(m^{1.41})$ [AYZ97]	$\mathcal{O}(\mathsf{h}^{\omega-1} n + m)$
EDGE-DISJOINT s-t PATHS	$\mathcal{O}(n^{\frac{3}{2}} m^{\frac{1}{2}})$ [GR99a]	$\mathcal{O}(\mathsf{h}^3 + n + m)$
GLOBAL MIN CUT	$\mathcal{O}(n^2 \log n)$ [Kar00]	$\mathcal{O}(\mathsf{h}^3 + n + m)$
MAX s-t VERTEX FLOW	$\mathcal{O}(nm)$ [Orl13]	$\mathcal{O}(\mathsf{h}^3 + n + m)$
WEIGHTED GLOBAL VERTEX MIN CUT	$\mathcal{O}(\kappa_1 nm \log(n^2/m))$ [HRG00]	$\mathcal{O}(\kappa_1\, \mathsf{h}\, m \log(\mathsf{h}))$ or $\mathcal{O}(\kappa_1\, \mathsf{h}^2\, n \log(\mathsf{h}) + m)$

Table 3.1: Overview about our results, where n and m denote the number of vertices and edges, h denotes the modular-width of the input graph, and κ_1 denotes the vertex connectivity if vertex capacities are ignored. The previous best result for MAXIMUM MATCHING, parameterized by modular-width h, was $\mathcal{O}(\mathsf{h}^4 n+m)$ [CDP19]. For all other problems we present the first algorithms parameterized by the modular-width. Consequences of the recent almost linear time algorithm for maximum flow are discussed at the end of Section 3.8.

the algorithm for $c(n + m)$ steps, for sufficiently large c relative to hidden constants in $\mathcal{O}$; it either terminates and returns the correct answer or allows the conclusion that $n+m < f(k)$, i.e., the input instance itself is the kernel. For all considered problems except MAXIMUM MATCHING, there are no previous algorithms parameterized by the modular-width of the input graph known. An overview of our obtained results is depicted in Table 3.1.

Overview. First, we will define the modular-width of a graph and the unique modular decomposition tree in Section 3.1. In Section 3.2 we present our algorithm for MAXIMUM MATCHING that can be executed in time $\mathcal{O}(\mathsf{h}^2 \log \mathsf{h} \cdot n+m)$ for graphs with n vertices, m edges and modular-width at most h and we extend this algorithm to the more general MAXIMUM b-MATCHING problem using the same running time bound. In Section 3.3, we generalize the analysis of the running time for many algorithms that use the modular decomposition tree by stating a general running time theorem in Section 3.3. We will use this running time theorem for the analysis for all other algorithms in this chapter. For VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, we present a combinatorial algorithm running in time $\mathcal{O}(\mathsf{h}^2 n + n^2)$ and an algorithm using fast matrix multiplication running in time $\mathcal{O}(\mathsf{h}^{1.842} n + n^2)$ in Section 3.4, followed by an $\mathcal{O}(\mathsf{h}^{\omega-1} n+m)$ time algorithm for TRIANGLE COUNTING in Section 3.5. The remaining problems that we consider in this chapter are regarding edge- and vertex-disjoint paths. The problem

[3] In the overview we assume for MAXIMUM b-MATCHING that $b(V) \geq n \log n$ to simplify the concrete running time of $\mathcal{O}(\min\{b(V), n \log n\} \cdot (m + n \log n))$

MAXIMUM EDGE-DISJOINT s-t PATHS and GLOBAL MINIMUM EDGE CUT for unweighted graphs are discussed in Section 3.6, for both problems we present an $\mathcal{O}(\mathsf{h}^3+n+m)$ time algorithm. The variants of those problems for vertex-disjoint paths are covered in Section 3.7. Here, we consider vertex-capacitated graphs and present an $\mathcal{O}(\mathsf{h}^3+n+m)$ time algorithm for MAXIMUM s-t VERTEX FLOW and an $\mathcal{O}(\kappa_1\,\mathsf{h}^2\,n\log(\mathsf{h})+m)$ time algorithm for GLOBAL MINIMUM VERTEX CUT, where κ_1 denotes the vertex connectivity if the vertex capacities are ignored. We conclude this chapter in Section 3.8.

3.1 Definition of Modular-Width

In this section, we will define the graph parameter modular-width and the associated modular decomposition tree. We first define the modular decomposition for undirected graphs $G = (V, E)$. The central structure that we will use in this chapter is called a *module*. A module is a vertex set $M \subseteq V$ such that all vertices in M have the same neighborhood regarding vertices outside of the module.

Definition 3.1 (Module)**.** Let $G = (V, E)$ be an undirected graph. A *module* is a set $M \subseteq V$ such that for all $u, v \in M$ it holds that $N_G(u) \cap (V \setminus M) = N_G(v) \cap (V \setminus M)$.

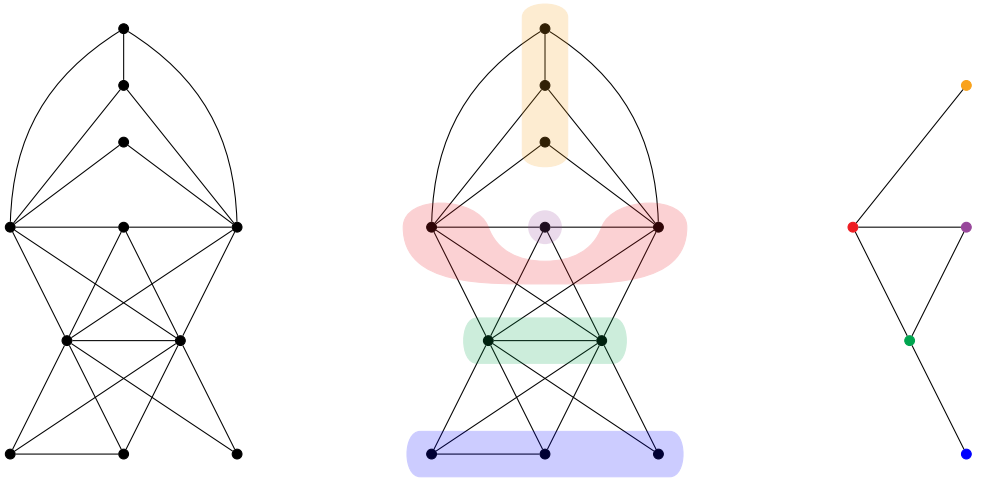
In other words, a vertex set $M \subseteq V$ is a module in G if for every vertex $x \in V \setminus M$ outside of the module it holds that either $M \subseteq N_G(x)$ or $M \cap N_G(x) = \emptyset$. Clearly, $\emptyset$, V, and every singleton set $\{v\}$ for every $v \in V$ are modules of G; these are called *trivial modules*. If a graph only admits trivial modules, we call G *prime*. Consider two modules $M_1, M_2 \subseteq V$ that do overlap, i.e., the sets $M_1 \cap M_2$, $M_1 \setminus M_2$ and $M_2 \setminus M_1$ are all nonempty. By the definition of a module, it holds that in this case also $M_1 \cup M_2$, $M_1 \cap M_2$, $M_1 \setminus M_2$, $M_2 \setminus M_1$, and $M_1 \,\Delta\, M_2$ are modules in G.

Lemma 3.2. *Let $G = (V, E)$ be a graph and $M_1, M_2 \subseteq V$ be two overlapping modules. Then $M_1 \cup M_2$, $M_1 \cap M_2$, $M_1 \setminus M_2$, $M_2 \setminus M_1$, and $M_1 \,\Delta\, M_2$ are modules in G.*

Proof. Since M_1 and M_2 do overlap, the sets $M_1 \setminus M_2$, $M_1 \cap M_2$, and $M_2 \setminus M_1$ are nonempty. We show as an example that the set $M_1 \setminus M_2$ fulfills the definition of a module, for the other sets similar argumentation hold: If $|M_1 \setminus M_2| = 1$ there is nothing to show. Let $v_1, v_1' \in M_1 \setminus M_2$ be two different vertices in $M_1 \setminus M_2$. For a vertex $x \in V \setminus M_1$ is holds that x is either adjacent to both or none of v_1 and v_1', since M_1 is a module. We are left to show, that v_1 and v_1' also have a uniform neighborhood to the vertices in $M_1 \cap M_2$. Let $x \in M_1 \cap M_2$ be arbitrary and assume that $\{x, v_1\} \in E$. We need to show that also $\{x, v_1'\} \in E$. Let $v_2 \in M_2$ be arbitrarily. Since M_2 is a module and $x \in M_2$, it holds that $\{v_1, v_2\} \in E$ and since M_1 is module, also $\{v_2, v_1'\} \in E$. Finally, again due to M_2 being a module, it holds that $\{v_1', x\} \in E$. An analog argumentation shows that if x is not adjacent to v_1, x is also not adjacent to v_1'. □

Consider a partition $P = \{M_1, M_2, \ldots, M_t\}$ of the vertices of G into modules with $t \geq 2$, called a *modular partition*. Consider two different modules M_i and M_j in P with $i \neq j$. If there exists $v \in M_i$ and $u \in M_j$ with $\{u, v\} \in E$, then, due to Definition 3.1, any vertex in M_i is adjacent to every vertex in M_j, i.e., there is a full join between those two vertex sets. In this case we call two modules M_i and M_j of P *adjacent*, and *non-adjacent* otherwise. This motivates the following definition:

Definition 3.3 (Quotient graph)**.** Let $P = \{M_1, M_2, \ldots, M_t\}$ be a modular partition of a graph $G = (V, E)$. Let $q_{M_i} \in M_i$ an arbitrarily selected vertex in M_i for $i \in [t]$. The *quotient graph*



(a) The input graph G. (b) A modular partition of V. (c) The quotient graph of G.

Figure 3.1: An example of a modular partition of the vertex set of a graph $G = (V, E)$ and a corresponding quotient graph.

$G_{/P}$ of G w.r.t. the modular partition P is defined as the induced subgraph of the vertex set $\{q_{M_1}, q_{M_2}, \ldots, q_{M_t}\}$ in G.

This means that the quotient graph w.r.t. a modular partition P is the graph whose vertices are in a one-to-one correspondence to the modules in P, with two vertices q_{M_i}, q_{M_j} in $G_{/P}$ being adjacent if and only if the corresponding modules M_i and M_j in G are adjacent, i.e., $G_{/P}$ can be seen as a compact representation of those edges in G with endpoints in different modules. See Figure 3.1 as an example of a modular partition and the corresponding quotient graph.

Each subgraph $G[M_i]$ for $i \in [t]$ is called a *factor*. In a modular decomposition, each factor will then be recursively decomposed as well until one reaches only trivial modules $\{v\}$ for each $v \in V$. We point out that if $A \subseteq V$ is module in G and $B \subseteq A$, then B is a module in G if and only if B is a module in $G[A]$. We call a set $M \subsetneq V$ with $M \neq V$ a *maximal* module if and only if M is a module and for all modules $M' \subsetneq V$ with $M' \neq V$ and $M \subseteq M'$ it holds that $M' = M$. Gallai [Gal67] showed already in 1967 the following theorem.

Theorem 3.4 ([Gal67])**.** *For any graph $G = (V, E)$ with $|V| \geq 2$. Then exactly one of the following three conditions is satisfied:*

- *G is not connected,*
- *$\overline{G}$ is not connected,*
- *G and $\overline{G}$ are connected and the maximal modules in G form a partition P of V. Moreover, the quotient graph $G_{/P}$ is a prime graph.*

We briefly sketch that in the last case, the set of maximal modules are indeed a partition; suppose for contraction that two maximal modules M_1 and M_2 do overlap. Then, by Lemma 3.2, it holds that also $M_1 \cup M_2$, $M_1 \cap M_2$, $M_1 \setminus M_2$, and $M_2 \setminus M_1$ are modules in G. Either $M_1 \cup M_2 \neq V$, thus, neither M_1 nor M_2 are maximal, or $M_1 \cup M_2 = V$, but then there is a modular partition $\{M_1 \setminus M_2, M_1 \cap M_2, M_2 \setminus M_1\}$ which implies that either G or $\overline{G}$ is not connected.

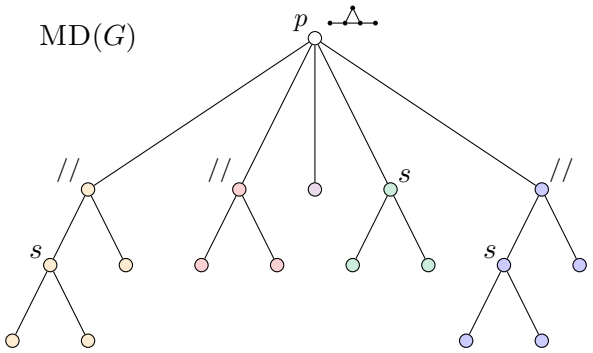


Figure 3.2: The modular decomposition tree MD(G) of the graph G in Figure 3.1. A prime node in MD(G) is labeled p and the corresponding quotient graph is attached to this node. A serial resp. parallel node is labeled by s resp. by //.

Note that a set $A \subseteq V$ is a module in G if and only if A is a module in the complement graph $\overline{G}$. Using Theorem 3.4, we can now recursively define the unique modular decomposition tree MD(G) for a graph $G = (V, E)$ that is a rooted tree that implicit represents all modules in G: If $|V| = 1$, the modular decomposition tree is a single leaf corresponding to V (base case). If G is not connected, consider the modular partition P consisting of each minimal connected component in G. To indicate that any union of modules in P forms itself again a module, we label the root node as a *parallel* node with one child per module in P. The implicit quotient graph corresponding to a parallel node is an isolated graph with $|P|$ vertices. If $\overline{G}$ is not connected, consider the modular partition P consisting of each minimal connected component in $\overline{G}$. Again, to indicate that every union in P forms a module, we label the root node as a *serial* node with one child per module in P. The implicit quotient graph corresponding to serial nodes is a clique with $|P|$ vertices. Finally, if G and $\overline{G}$ are connected, we consider the modular partition P consisting of the maximal modules in G. This modular partition is unique due to Theorem 3.4. The root is labeled as a *prime* node with one child per module in P and the corresponding quotient graph is attached to the root. We decompose all children recursively until eventually reaching the base case. See Figure 3.2 as an example of a modular decomposition tree. Note that Theorem 3.4 also implies that MD(G) is unique.

We call a module M_1 that do *not* overlap with any other module M_2 a *strong* module. E.g., all trivial modules are strong modules. By the above definition, the modular decomposition tree MD(G) of a graph G corresponds to the Hasse diagram of the family of strong modules with respect to the subset relation. We refer to such a Hasse diagram also as the *inclusion tree* of all strong modules. This means, each internal node v_M of MD(G) with the set of children $\{v_{M_1}, \ldots, v_{M_t}\}$ corresponds to a strong module M of G and $P = \{M_1, \ldots, M_t\}$ is a modular partition of $G[M]$ into strong modules where M_i is the corresponding module of v_{M_i}, with $i \in [t]$. Further, a vertex v_A is an an ancestor of v_B in MD(G) if and only if $B \subsetneq A$ for the corresponding strong modules A and B of G.

We can now define the *modular-width* of a graph G as the minimum $h \in \mathbb{N}$ such that any prime node in MD(G) has at most h children, i.e., the size of any quotient graph of a prime node in MD(G) is at most h.

Definition 3.5 (modular-width)**.** The *modular-width* of a graph G is the smallest $h \in \mathbb{N}$ such that any prime node in the modular decomposition tree MD(G) has at most h children.

If the modular decomposition tree of a graph G consists only of parallel and serial nodes, the modular-width of G is zero. The class of graphs with modular-width zero is exactly the class of

all cographs.[4] For any graph G with n vertices, the modular decomposition tree $\mathrm{MD}(G)$ have exactly n leaves, each corresponding to singleton set $\{v\}$ for $v \in V(G)$. Since each internal node in $\mathrm{MD}(G)$ has at least two children, the total number of nodes in $\mathrm{MD}(G)$ is at most $2n-1$.

We point out that although there may be up to exponential many modules in a graph, e.g., if G is a clique, the modular decomposition tree $\mathrm{MD}(G)$ is a compact representation of all modules in G only using linear space; either a module directly corresponds to a node in $\mathrm{MD}(G)$, i.e., it is a strong module, or it is a union of some modules in a partition corresponding to a parallel or serial node[5] in $\mathrm{MD}(G)$.

It is known that $MD(G)$ can be computed in time $\mathcal{O}(n+m)$ [TCHP08]. We refer to a survey of Habib and Paul [HP10] for more information about modular decompositions.

Modular decomposition for directed graphs. We will only consider undirected graphs in this chapter. In Chapter 6, we show how to define all graphs of modular-width at most h via algebraic expressions defined on a set of operations. This equivalent definition via operations can be naturally adapted to directed graphs.[6] For completeness, we will shortly describe the modular decomposition for directed graphs. For a directed graph $G = (V, E)$, a set $M \subseteq V$ is called a module if for all $u, v \in V$ it holds that $N^+(u) \cap (V \setminus M) = N^+(v) \cap (V \setminus M)$ and $N^-(u) \cap (V \setminus M) = N^-(v) \cap (V \setminus M)$. Both, the modular decomposition of undirected graphs and directed graphs can be seen as a special case of a decomposition of so-called (weak) partitive sets [CHM81]. For a finite set X, let $\mathcal{F} \subseteq 2^X$ be a family of subsets of X. If for every pair of members $A, B \in \mathcal{F}$ that do overlap it holds that $A \cup B$, $A \cap B$, $A \setminus B$, $B \setminus A \in \mathcal{F}$, the set $\mathcal{F}$ is called a weak partitive set. If additionally it holds that also $A \,\Delta\, B \in \mathcal{F}$, the set $\mathcal{F}$ is called a partitive set. We call a member of $\mathcal{F}$ a *strong* member, if it does not overlap with any other member of $\mathcal{F}$. Note that the set of all modules in an undirected graph forms a partitive set, cf. Lemma 3.2. One can show that the set of all modules in a directed graph forms a weak partitive set [ER90]. Both, a partitive set and a weak partitive set, can be represented by a decomposition tree corresponding to the inclusion tree of the strong members [MR84]. For the decomposition of a partitive set, each internal node is either degenerate or prime, i.e., either every arbitrary union of children are contained in $\mathcal{F}$ or just each child by oneself. For the partitive set consisting of all modules in an undirected graphs, a degenerate node corresponds either to a parallel or serial node in the modular decomposition tree. For the decomposition of a *weak* partitive set, each internal node is labeled degenerate, *linear*, or prime, whereas for a linear node it holds that there exists an ordering of the children such that each union of *consecutive* children are contained in $\mathcal{F}$. Applying this scheme to the modular decomposition tree of a directed graph, the implicit quotient graph of a linear node with t children is a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_t\}$ and $E = \{(v_i, v_j) \in V^2 \mid i \leq j\}$, i.e, the transitive closure of a directed path. The modular-width of a directed graph is again defined by the smallest $h \in \mathbb{N}$ such that the size of any quotient graph of a prime node in the modular decomposition is at most h. McConnell and Montgolfier [MdM05] gave an algorithm to compute the modular decomposition of directed graphs also in linear time $\mathcal{O}(n+m)$.

[4] There are also definitions that define the modular-width as the smallest $h \geq 2$ such that every quotient graph does have at most h children, e.g.,[CDP19]. Here, we adapt a definition of [GLO13], especially due to our purposes in Chapter 6. All definitions of modular-width coincide for graphs of modular-width at least two.

[5] The parallel and serial nodes in a modular decomposition are also subsumed under the term *degenerate* nodes.

[6] Using the (equivalent) definition via operations for directed modular-width, we extend the a algorithm for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS to directed graphs with possible negative vertex weights in Section 6.4.3. We will either conclude that the input graph contains a cycle of negative weight or we compute the shortest path distance for all pairs of vertices.

3.2 Maximum Matching

In the MAXIMUM MATCHING problem we are given a graph $G = (V, E)$ and need to find a maximum set $X \subseteq E$ of pairwise disjoint edges. Each set $X \subseteq E$ of pairwise disjoint edges is called a *matching* of size $|X|$. The size of a maximum matching of a graph G is denoted by $\mu(G)$. The fastest known unparameterized algorithm is due to Micali and Vazirani [MV80] and runs in time $\mathcal{O}(m\sqrt{n})$ on graphs with n vertices and m edges.

MAXIMUM MATCHING

Input: An undirected graph $G = (V, E)$.
Output: Set $X \subseteq E$ of disjoint edges with maximum size.

A *b-matching* is a generalization of a matching that specifies in the input for each vertex a *capacity bound* of how many edges in the matching may be incident with that vertex. Any edge can be chosen any number of times. Formally, the capacity bounds are given by a function $b\colon V \to \mathbb{N}$, and a b-matching is a function $x\colon E \to \mathbb{N}$ that fulfills for every vertex $v \in V$ the constraint that $\sum_{e \in \delta(v)} x(e) \leq b(v)$. For the special case of $b \equiv 1$ this problem is the classical problem of finding a maximum matching. Gabow [Gab18] showed how to find a b-matching that maximizes $\sum_{e \in E} x(e)$ in time $\mathcal{O}((n \log n) \cdot (m + n \log n))$.

MAXIMUM b-MATCHING

Input: An undirected graph $G = (V, E)$ and a function $b\colon V \to \mathbb{N}$.
Output: $x\colon E \to \mathbb{N}$ with maximum $\sum_{e \in E} x(e)$, s.t. $\forall v \in V\colon \sum_{e \in \delta(v)} x(e) \leq b(v)$.

Coudert et al. [CDP19] gave an $\mathcal{O}(\mathsf{h}^4\, n + m)$-time algorithm for MAXIMUM MATCHING, where h denotes the modular-width of the input graph. In this section we will improve this result by providing an algorithm for MAXIMUM MATCHING that runs in time $\mathcal{O}(\mathsf{h}^2 \log \mathsf{h} \cdot n + m)$. The main idea of our algorithm is to compress the computation of a matching in G to a computation of a b-matching, instead of using the structure of modular decompositions to speed up the search for augmenting paths (like in [CDP19]).

Theorem 3.6. *For every graph $G = (V, E)$ with modular-width h,* MAXIMUM MATCHING *can be solved in time $\mathcal{O}(\mathsf{h}^2 \log \mathsf{h} \cdot n + m)$.*

Algorithm. First, we compute the modular decomposition tree $\mathrm{MD}(G)$. We will traverse the decomposition tree in a bottom-up manner. For each v_M in $\mathrm{MD}(G)$, with M denoting the corresponding module of G, we will compute a maximum matching in $G[M]$. Note that for the root module v_M of $\mathrm{MD}(G)$ it holds that $G[M] = G$. For any leaf node v_M of $\mathrm{MD}(G)$, we have $\mu(G[M]) = 0$, since $G[M]$ is a graph consisting of a single vertex. Let v_M be a non-leaf vertex in $\mathrm{MD}(G)$ with the set of children $\{v_{M_1}, \ldots, v_{M_t}\}$. This means that $\{M_1, \ldots, M_t\}$ is a modular partition of $G[M]$, where $M_i \subseteq M$ corresponds to the vertex v_{M_i} in $\mathrm{MD}(G)$ for $i \in [t]$. In the following, we can always assume that we have already computed $\mu(G[M_i])$ for $i \in [t]$. The next lemma shows that the concrete structure inside a module is irrelevant for the maximum matching size of the whole graph, in particular, only the number of vertices and the size of a maximum matching is important. The following lemma is a more general version of [CDP19, Lemma 5.1], but can be proven in a similar way.

Lemma 3.7. *Let M be a module of a graph $G = (V, E)$ and let $G[M] = (M, E_M)$. Let further $A \subseteq \binom{M}{2}$ be any set of edges on the vertices of M such that $\mu((M, A)) = \mu((M, E_M))$. Then, the size of a maximum matching of $G' = (V, (E \setminus E_M) \cup A)$ is equal to the size of a maximum matching of G.*

Proof. We first show that $\mu(G') \geq \mu(G)$. Let us consider a maximum matching $F \subseteq E$ in $G = (V, E)$. To get a maximum matching in G' we replace all edges in F that are incident with M: First, replace all edges in $F \cap E(G[M])$ by an arbitrary matching $A' \subseteq A$ of the same size; such a matching must exist because $F \cap E(G[M])$ is not larger than a maximum matching in $G[M]$ and $\mu((M, A)) = \mu((M, E_M))$. Second, we replace all edges in F that have exactly one endpoint in M as follows: Let $Y \subseteq M$ be the set of vertices in M that are endpoints of an edge in F whose other endpoint is not in M. By assumption, $|M \setminus V(A')| \geq |Y|$ and since all vertices in $V \setminus M$ that are connected to a vertex in Y in G are also connected to all vertices in $M \setminus V(A')$ in G', we can replace all edges of F that have exactly one endpoint in M. Thus, $\mu(G') \geq \mu(G)$, i.e., replacing the edges in a module by an arbitrary set of edges with same maximum matching size does not decrease the size of the maximum matching for the whole graph. Applying this argument for $A' := E_M$ to swap back to the original edge set yields $\mu(G) \geq \mu(G')$ and completes the proof. □

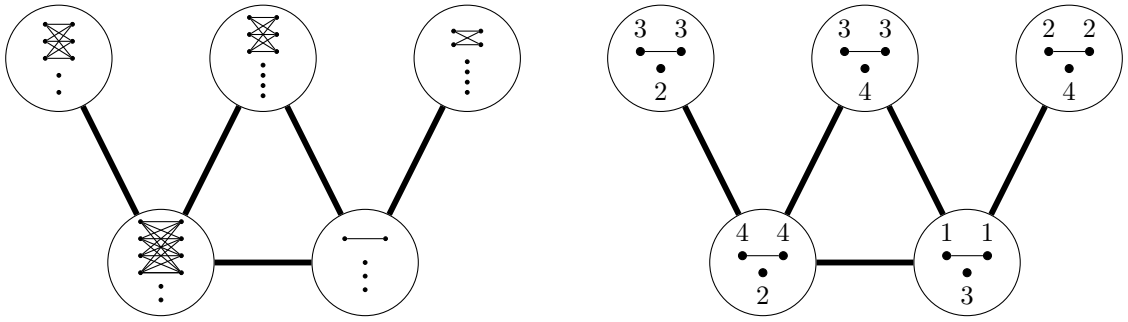
We can now describe how to compute $\mu(G[M])$ for a node v_M in $\mathrm{MD}(G)$. Let $\{v_{M_1}, \ldots, v_{M_t}\}$ be the set of children of v_M in $\mathrm{MD}(G)$, meaning that $P = \{M_1, \ldots, M_t\}$ is a modular partition of $G[M]$. We can assume that we have already computed $\mu(G[M_i])$ for $i \in [t]$. Let $G[M]_{/P}$ be the quotient graph of $G[M]$ w.r.t. the modular partition P. First, suppose that v_M is a prime node. We will reduce the problem of computing a maximum matching in $G[M]$ to the computation of a maximum b-matching in an auxiliary graph closely related to the quotient graph of v_M that we will define next.

Definition 3.8. Let $G = (V, E)$ be a graph and $P = \{M_1, \ldots, M_t\}$ be a modular partition of G. Let $n_i = |V(G[M_i])|$ denote the number of vertices in $G[M_i]$ and $\mu(G[M_i])$ the size of a maximum matching in $G[M_i]$. We define an auxiliary graph $G^* = (V^*, E^*)$ together with capacity bounds $b\colon V^* \to \mathbb{N}$ as an instance (G^*, b) for the maximum b-matching problem as follows:

- For every module $M_i \in P$, with $i \in [t]$, we add three vertices v_i^1, v_i^2, v_i^3 to V^* and set $b(v_i^1) = b(v_i^2) = \mu(G[M_i])$ and $b(v_i^3) = n_i - 2\mu(G[M_i])$.
- We add the edge $\{v_i^1, v_i^2\}$ to E^* for $i \in [t]$.
- For each edge between vertices q_i and q_j in $G_{/P}$ that corresponds to modules M_i and M_j, we add the nine edges $\{v_i^c, v_j^d\}$ with $c, d \in \{1, 2, 3\}$ to E^*.

Lemma 3.9. *Let $G = (V, E)$ be a graph and $P = \{M_1, \ldots, M_t\}$ be a modular partition of G. Let (G^*, b) be the instance of a maximum b-matching problem as defined in Definition 3.8. Then, the size of a maximum matching in G is equal to the size of a maximum b-matching of (G^*, b).*

Proof. Consider a graph $G = (V, E)$ with a modular partition $P = \{M_1, \ldots, M_t\}$. For $M_i \in P$ let $n_i = |V(G[M_i])|$ and let $\mu_i = \mu(G[M_i])$. Due to Lemma 3.7, we can replace each $G[M_i]$, for $i \in [t]$, by a graph consisting of a complete bipartite graph K_{μ_i, μ_i} together with $n_i - 2\mu_i$ single vertices without changing the size of a maximum matching. We do this for every module



(a) The graph $\hat{G}$. A bold edge represents a full join between the encircled vertices.

(b) The graph G^* as definied in Definition 3.8. Each vertex is labeled by its b-value. A bold edge represents a full join (9 edges) between the encircled vertices.

Figure 3.3: The graphs $\hat{G}$ and G^* as in the proof of Lemma 3.9.

$M_i \in P$ and denote the resulting graph by $\hat{G}$, see also Figure 3.3 for an example. Note that $\mu(G) = \mu(\hat{G})$. Now, each replacement of $G[M_i]$ can be partitioned into three modules: Each side of the complete bipartite graph K_{μ_i,μ_i} and the $n_i - 2\mu_i$ remaining single vertices, giving us a modular partition P' of $\hat{G}$ of size $3t$ with the property that for every module $M \in P'$ the factor graph $G[M]$ is an independent set. The quotient graph $\hat{G}_{/P'}$ is exactly the auxiliary graph G^* of G and the capacity bound of a vertex $v \in V(G^*)$ is equal to the number of vertices in the corresponding module, cf. Figure 3.3. Since any maximum b-matching in (G^*, b) directly corresponds to a maximum matching in $\hat{G}$, this completes the proof. □

Suppose now that v_M is a serial or parallel node. Instead of computing $\mu(G[M])$ directly, we will modify the decomposition tree $\mathrm{MD}(G)$ (cf. [CDP19]). Let $\{v_{M_1}, \ldots, v_{M_t}\}$ be the children of a serial v_M in $\mathrm{MD}(G)$. We will iteratively compute a maximum matching for $G_i = G[\cup_{1 \leq j \leq i} M_j]$ by using a modular partition of G_i consisting of the two modules $\cup_{1 \leq j < i} M_j$ and M_i, for $i \in [t]$. This means that we replace a serial node with t children by $t - 1$ series nodes with only two children. We will treat the newly inserted nodes as prime nodes (with a quotient graph isomorphic to K_2). After replacing the serial nodes of the modular decomposition tree $\mathrm{MD}(G)$, every node still has at least two children; hence, we still have a most $2n - 1$ nodes in $\mathrm{MD}(G)$. The same approach can also be used to treat parallel nodes in $\mathrm{MD}(G)$; replace a parallel node with t children by $t-1$ parallel nodes with only two children and treat the newly inserted nodes as prime nodes (with a quotient graph isomorphic to graph with two isolated vertices).

Finally, note that after only computed the value of a maximum matching in G. However, using the values of a maximum b-matching for each auxiliary graph of a node in the modular decomposition tree, we can easily reconstruct a concrete maximum matching $X \subseteq E$ in linear time.

Running Time. Consider a graph $G = (V, E)$ with modular-width h. Computing the modular decomposition tree $\mathrm{MD}(G)$ takes time $\mathcal{O}(n + m)$. As described above, we modify the decomposition tree such that every serial node and every parallel node of $\mathrm{MD}(G)$ with $t \geq 3$ children is replaced by $t - 1$ 'pseudo-prime' nodes with exactly two children. This replacement can be done in time $O(n)$. Note that there are at most $2n - 1$ nodes in $\mathrm{MD}(G)$, even after the above replacement. Now, every node $v_M \in \mathrm{MD}(G)$ has a set of children $\{v_{M_1}, \ldots, v_{M_t}\}$ with $2 \leq t \leq \mathsf{h}$. This means that $P = \{M_1, \ldots, M_t\}$ is a modular partition of $G[M]$ and the

quotient graph $G[M]_{/P}$ consists of $t \leq \mathsf{h}$ vertices. Since we have already computed $\mu(G[M_i])$ for all $i \in [t]$, we can construct the auxiliary graph G^* of $G[M]$ as defined in Definition 3.8 in time $\mathcal{O}(V(G^*) + E(G^*)) = \mathcal{O}(t^2)$. Recall that $|V(G^*)| = 3t$. Thus, we can compute a maximum b-matching of G^* subject to b in time $\mathcal{O}(t^3 \log t)$ using the algorithm due to Gabow [Gab18]. We have to do this for every prime and 'pseudo-prime' node, but a slightly more careful summation of running times over all nodes gives an improvement over the obvious upper bound of $\mathcal{O}(\mathsf{h}^3 \log \mathsf{h} \cdot n + m)$: Let p be the number of nodes in $\mathrm{MD}(G)$, enumerate the nodes in $\mathrm{MD}(G)$, and let t_i denote the number of vertices in the quotient graph of the i-th node in $\mathrm{MD}(G)$, i.e., the number of children of this node. Then, neglecting constant factors and assuming that $\mathrm{MD}(G)$ is already computed, we can solve MAXIMUM MATCHING in time:

$$\sum_{i=1}^{p} t_i^3 \log t_i \leq \left(\sum_{i=1}^{p} t_i\right) \cdot \max_{i \in [p]} \left\{t_i^2 \log t_i\right\} \leq 2n \cdot \max_{i \in [p]} \left\{t_i^2 \log t_i\right\} \leq 2n \cdot \left(\mathsf{h}^2 \log \mathsf{h}\right)$$

The second inequality holds since $\sum_{i=1}^{p} t_i$ counts each node in $\mathrm{MD}(G)$ exactly once, except for the root node. As constant factors propagate through the inequality, the total running time of the algorithm is $\mathcal{O}(\mathsf{h}^2 \log \mathsf{h} \cdot n + m)$, which proves Theorem 3.6.

Generalization to b-matching We can easily generalize this result to the more general maximum b-matching problem.

Theorem 3.10. *For every graph $G = (V, E)$ with modular-width h,* MAXIMUM b-MATCHING *can be solved in time $\mathcal{O}(\mathsf{h}^2 \log \mathsf{h} \cdot n + m)$.*

Again, the concrete structure inside a module will not be important. The only important information is the size of a maximum b-matching and the sum of all b-values in a module. By a slight abuse of notation, we denote by $\mu(G, b)$ the size of a maximum b-matching in G. We naturally extend Definition 3.8 to b-matchings:

Definition 3.11. Let $G = (V, E)$ be a graph with $b\colon V \to \mathbb{N}$ and let $P = \{M_1, \ldots, M_t\}$ be a modular partition of G. Let $n_i = \sum_{v \in M_i} b(v)$ and $\mu(G[M_i], b)$ be the size of a maximum b-matching in $G[M_i]$ for $i \in [t]$. We define an auxiliary graph $G^* = (V^*, E^*)$ together with capacity bounds $b^*\colon V \to \mathbb{N}$ as an instance (G^*, b^*) for the maximum b-matching problem as follows:

- For every module $M_i \in P$, with $i \in [t]$, we add three vertices v_i^1, v_i^2, v_i^3 to V^* and set $b^*(v_i^1) = b^*(v_i^2) = \mu(G[M_i], b)$ and $b^*(v_i^3) = n_i - 2\mu(G[M_i], b)$.
- We add the edge $\{v_i^1, v_i^2\}$ to E^* for $i \in [t]$.
- For each edge between vertices q_i and q_j in $G_{/P}$ that corresponds to modules M_i and M_j, we add the nine edges $\{v_i^c, v_j^d\}$ with $c, d \in \{1, 2, 3\}$ to E^*.

Lemma 3.12. *Let $G = (V, E)$ be a graph and $P = \{M_1, \ldots, M_t\}$ be a modular partition of G. Let (G^*, b^*) be the instance of a maximum b-matching problem as defined in Definition 3.11. Then, the size of a maximum b-matching in (G, b) is equal to the size of a maximum b-matching of (G^*, b^*).*

Proof. Consider a graph $G = (V, E)$ with a modular partition $P = \{M_1, \ldots, M_t\}$. For $M_i \in P$ let $n_i = \sum_{v \in M_i} b(v)$ and let μ_i be the size of a maximum b-matching in M_i. Note that one can solve b-matching by replacing every vertex v by $b(v)$ many isolated vertices that are connected

in the same way as v (i.e., substituting $I_{b(v)}$ into v). After considering this replacement and due to Lemma 3.7, we can replace $G[M_i]$, for $i \in [t]$, by a graph consisting of a complete bipartite graph K_{μ_i,μ_i} together with $n_i - 2\mu_i$ single vertices without changing the size of a maximum matching. We do this for every module M_i and denote the resulting graph by $\hat{G}$. As in the proof of Lemma 3.9, we can subdivide every module in three parts. This yields to the instance (G^*, b^*) as defined in Definition 3.11. Again, any maximum b-matching in (G^*, b^*) directly corresponds to a maximum b-matching in (G, b), which completes the proof. □

The running time can be bounded in the same way as before. However, to see that this algorithm is also adaptive for sparse graphs (at least for large b-values), we can modify the computation of the running time: Let again p be the number of nodes in $\mathrm{MD}(G)$, enumerate the nodes in $\mathrm{MD}(G)$, let t_i denote the number vertices in the quotient graph of the i-th node of $\mathrm{MD}(G)$, and let m_i the number of edges in the quotient graph of the i-th node of $\mathrm{MD}(G)$. Thus, we can compute a maximum b-matching of G^* subject to b^* in time $\mathcal{O}((t_i \log t_i) \cdot (m_i + t_i \log t_i))$ using the algorithm due to Gabow [Gab18]. Then, neglecting constant factors and assuming that $\mathrm{MD}(G)$ is already computed, we can solve MAXIMUM b-MATCHING in time:

$$\begin{aligned}\sum_{i=1}^{p}(t_i \log t_i) \cdot (m_i + t_i \log t_i) &= \sum_{i=1}^{p} m_i t_i \log t_i + \sum_{i=1}^{p} t_i^2 \log^2 t_i \\ &\leq \left(\sum_{i=1}^{p} m_i\right) \max_{i\in[p]}\{t_i \log t_i\} + \left(\sum_{i=1}^{p} t_i\right) \max_{i\in[p]}\{t_i \log^2 t_i\} \\ &\leq m \cdot \mathsf{h} \log \mathsf{h} + 2n \cdot (\mathsf{h} \log^2 \mathsf{h})\end{aligned}$$

Since constant factors propagate through the inequality, the total running time of the algorithm is $\mathcal{O}((m + n \log \mathsf{h}) \cdot (\mathsf{h} \log \mathsf{h}))$. Therefore, even for $\mathsf{h} \in \Theta(n)$ our algorithm is not worse than the (currently) best unparameterized algorithm with time complexity $\mathcal{O}((m + n \log n) \cdot (n \log n))$, assuming $b(V) \geq n \log n$.

3.3 General Running Time Theorem

For many parameterized algorithms that make use of the structure quantified by the parameter modular-width, the resulting running time is solely depending on the maximal time needed to process any node in the modular decomposition tree, as seen for example in the previous Section 3.2. Before we continue describing further algorithms, we will derive a general running time theorem that is applicable to many algorithms that utilize the modular decomposition tree, in particular, all algorithms in this chapter. Since we will focus on functions describing running times, we will restrict ourselves to functions $T\colon \mathbb{R}_{\geq 1} \to \mathbb{R}_{\geq 1}$ that are superhomogeneous.

Definition 3.13 ([BS05])**.** A function $T\colon \mathbb{R}_{\geq 1} \to \mathbb{R}_{\geq 1}$ is *superhomogeneous* if for all $\lambda \geq 1$ the following holds:

$$\lambda \cdot T(n) \leq T(\lambda \cdot n)$$

Lemma 3.14. *Let $T\colon \mathbb{R}^2_{\geq 1} \to \mathbb{R}_{\geq 1}$ be a function that is superhomogeneous in the first component and monotonically increasing in the second component. Then*

$$\max_{\substack{1\leq k\leq n\\ 1\leq \ell\leq m}} \frac{T(k,\ell)}{k} \leq \frac{T(n,m)}{n}.$$

Proof. Since T is monotonically increasing in the second component, the maximum is reached for $\ell = m$. Pick k with $1 \leq k \leq n$ arbitrarily and set $\lambda \geq 1$ so that $\lambda \cdot k = n$. It follows directly that

$$\frac{T(n,m)}{n} = \frac{T(\lambda k, m)}{\lambda k} \geq \frac{\lambda T(k,m)}{\lambda k} = \frac{T(k,m)}{k}.$$

This completes the proof. □

We can now state the running time framework.

Theorem 3.15. *Let G be a graph of modular-width equal to h, let* $\mathrm{MD}(G)$ be the modular decomposition tree of G, and let $T\colon \mathbb{R}^2_{\geq 1} \to \mathbb{R}_{\geq 1}$ be a function that is superhomogeneous in the first component and monotone increasing in the second component. If the running time of an algorithm for any prime node $v_i \in V(\mathrm{MD}(G))$ is upper bounded by $\mathcal{O}(T(n_i, m_i))$, where n_i and m_i denote the number of vertices and edges of the quotient graph corresponding to v_i, then the total running time can be upper bounded by

$$\mathcal{O}\left(\frac{n}{\mathsf{h}} \cdot T(\mathsf{h}, m) + n + m\right) \qquad \text{and} \qquad \mathcal{O}\left(\frac{n}{\mathsf{h}} \cdot T(\mathsf{h}, \mathsf{h}^2) + n + m\right).$$

If, additionally, T is also superhomogeneous in the second component then the running time can also be upper bounded by

$$\mathcal{O}\left(T(\mathsf{h}, m) + n + m\right).$$

Proof. In a first step we compute the modular decomposition tree of the input graph G in linear time [TCHP08]. First, we replace any serial resp. parallel node with t children by a sequence of $t-1$ 'pseudo-prime' nodes each with a corresponding quotient graph isomorphic to K_2 resp. I_2. Let p denote the number of prime nodes after this replacement. For any node v_i of $\mathrm{MD}(G)$ let n_i and m_i denote the number of vertices resp. edges in the quotient graph associated with v_i. Thus, it holds that $n_i \leq \mathsf{h}$ and $m_i \leq \mathsf{h}^2$. Then, the running time all nodes can be upper bounded by

$$\begin{aligned}\sum_{i=1}^{p} T(n_i, m_i) &= \sum_{i=1}^{p} n_i \frac{T(n_i, m_i)}{n_i} \\ &\leq \sum_{i=1}^{p} n_i \cdot \left(\max_{\substack{1 \leq n_i \leq \mathsf{h} \\ 1 \leq m_i \leq m}} \frac{T(n_i, m_i)}{n_i} \right) \\ &\leq 2n \cdot \frac{T(\mathsf{h}, m)}{\mathsf{h}}.\end{aligned} \tag{3.1}$$

The last inequality holds due to Lemma 3.14 and since $\sum_{i=1}^{t} n_i$ counts each node in the modular decomposition (except of the root) exactly once.

Since T is monotone increasing in the second component and each quotient graph has at most h^2 many edges, one can replace the value m_i in Equation (3.1) by the value h^2. Doing this, we can also bound the running time for processing all nodes in the modular decomposition tree by $\mathcal{O}(n \frac{T(\mathsf{h}, \mathsf{h}^2)}{\mathsf{h}})$.

If, additionally, T is also superhomogeneous in the second component then the running time can also be upper bounded by

$$\begin{aligned}\sum_{i=1}^{p} T(n_i, m_i) &= \sum_{i=1}^{p} m_i \frac{T(n_i, m_i)}{m_i} \\ &\leq \sum_{i=1}^{p} m_i \cdot \left(\max_{\substack{1 \leq n_i \leq \mathsf{h} \\ 1 \leq m_i \leq m}} \frac{T(n_i, m_i)}{m_i} \right) \\ &\leq m \cdot \frac{T(\mathsf{h}, m)}{m} \\ &= T(\mathsf{h}, m).\end{aligned}$$

The second to last inequality holds using the argumentation from Lemma 3.14 and since $\sum_{i=1}^{t} m_i$ counts every edge at most once. □

Example. In Section 3.2 it was shown how to solve MAXIMUM MATCHING and MAXIMUM b-MATCHING with a running time per prime node of $\mathcal{O}(m_i n_i \log n_i) + (n^2 \log^2 n)$, where n_i denotes the number of vertices and m_i denotes the number of edges in the quotient graph of node v_i in the modular decomposition tree. Thus, by using Theorem 3.15, one can bound the total running time by $\mathcal{O}(\min\{\mathsf{h}^2 \log \mathsf{h} \cdot n + m, (m + n \log \mathsf{h}) \cdot (\mathsf{h} \log \mathsf{h})\})$.

3.4 Vertex-Weighted All-Pairs Shortest Path

We turn now our focus on studying the VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS problem, in which the task is to compute $dist_G(u, v)$ for all $u, v \in V$ for a graph G with vertex weights $\omega \colon V \to \mathbb{R}_{\geq 0}$.

VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS

Input: An undirected graph $G = (V, E)$, vertex weights $\omega \colon V \to \mathbb{R}_{\geq 0}$.
Output: The pairwise distances $dist_G(u, v)$ for all $u, v \in V$.

In this chapter, we stick to undirected graphs and consider non-negative vertex weights. In Section 6.4.3, we also consider this problem with arbitrary vertex-weights on directed graphs without negative cycles. In this section, we obtain the following result.

Theorem 3.16. *For every graph $G = (V, E)$ with modular-width h and vertex weights $\omega \colon V \to \mathbb{R}_{\geq 0}$,* VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS *can be solved in time $\mathcal{O}(\mathsf{h}^{1.842} n + n^2)$ time using fast matrix multiplication or in combinatorial time $\mathcal{O}(\mathsf{h}^2 n + n^2)$.*

Let $\mathrm{MD}(G)$ be the modular decomposition tree of G. We will traverse $\mathrm{MD}(G)$ in a top-down manner. For a node v_M in the decomposition tree with children $v_{M_1}, \ldots, v_{M_t}$ let M resp. $M_1, \ldots, M_t$ be the corresponding modules in G for $t \geq 2$. We will compute for every pair of vertices $u, v \in M$ with $u \in M_i$ and $v \in M_j$ for $i \neq j$ the shortest path in the whole graph G. Since every vertex of G corresponds to a leaf in $\mathrm{MD}(G)$, we eventually consider every pair of vertices with this procedure. We start with some structural properties of shortest paths in a graph that can be partitioned into modules.

Lemma 3.17. *Let $P = \{M_1, \dots, M_t\}$ be a modular partition of a graph $G = (V, E)$ with vertex weights $\omega\colon V \to \mathbb{R}_{\geq 0}$. Let $x, y \in V$ be two vertices with $\{x, y\} \not\subseteq M_i$ for all $i \in [t]$. Then, there exists a shortest x-y path that visits each module at most once.*

Proof. Let $Q = (x = v_1, v_2, \dots, v_n = y)$ be a shortest x-y path in G regarding ω. Assume that there exist $i, j \in [n]$ with $i \neq j$ such that $\{v_i, v_j\} \subseteq M_k$ for some $k \in [t]$. Let i be minimal and j be maximal under this condition. We distinguish two cases: In the case $j \neq n$, consider the path $Q' = (v_1, \dots, v_{i-1}, v_i, v_{j+1}, v_{j+2}, \dots, v_n)$. Since j is maximal and $v_j \neq v_n$, it holds that $v_{j+1} \notin M_k$, but v_{j+1} is adjacent to all vertices of M_k. Thus, the edge $\{v_i, v_{j+1}\}$ exists and Q' is indeed a x-y path with $\omega(Q') \leq \omega(Q) = dist_G(x, y)$.

If $v_j = v_n$ then it holds that $v_i \neq v_1$ since otherwise $\{x, y\}$ would be in a same module. Consider the path $Q'' = (v_1, \dots, v_{i-1}, v_j)$. Since i is minimal, it holds that $v_{i-1} \notin M_k$, but v_{i-1} is adjacent to all vertices of M_k, in particular, to v_j. Thus, Q'' is an x-y path with $\omega(Q'') \leq \omega(Q) = dist_G(x, y)$. We iterate this procedure for every pair of vertices that are in a same module. Since the vertices in Q' resp. Q'' are a strict subset of the vertices in Q, the number of pairs that are in a same module strictly reduces each time. □

We will use this property to compute shortest paths between vertices in different modules. To do so, we extend the quotient graph by vertex weights.

Definition 3.18. Let $G = (V, E)$ be a graph with vertex weights $\omega_G\colon V \to \mathbb{R}_{\geq 0}$, and let $P = \{M_1, \dots, M_t\}$ be a modular partition of G. We define vertex weights for the quotient graph $G_{/P}$ by $\omega_{G_{/P}}(q_{M_i}) = \min_{v \in M_i} \omega_G(v)$.

Lemma 3.19. *Let $G = (V, E)$ be a graph with vertex weights $\omega_G\colon V \to \mathbb{R}_{\geq 0}$, and let $P = \{M_1, \dots, M_t\}$ be a modular partition of G. Let $G_{/P}$ be the quotient graph with vertex weights $\omega_{G_{/P}}$ as defined in Definition 3.18, and let $u, v \in V$ be two vertices with $u \in M_i$ and $v \in M_j$ for $i \neq j$. Then, $dist_G(u, v) = dist_{G_{/P}}(q_{M_i}, q_{M_j}) - \omega_{G_{/P}}(q_{M_i}) + \omega_G(u) - \omega_{G_{/P}}(q_{M_j}) + \omega_G(v)$.*

Proof. Let $u, v \in V$ be two vertices in G with $u \in M_i$ and $v \in M_j$ for $i \neq j$. Every shortest q_{M_i}-q_{M_j} path P^* in $G_{/P}$ corresponds to a u-v path P in G by first replacing each vertex in P^* by the minimum-weight vertex of the corresponding module, and afterwards, since M_i and M_j are modules, replacing the first vertex by u and the last vertex by v.

Conversely, let P be a shortest u-v path in G with $u \in M_i$ and $v \in M_j$ for $i \neq j$. Due to Lemma 3.17, we can assume that no two vertices of P are in the same module. Thus, due to the structure of modules, one can assume that each vertex of P, except of u and v, are vertices of minimum weight in their respective module. Hence, there is a q_{M_i}-q_{M_j} path in $G_{/P}$ of cost $dist_G(u, v) - \omega_G(v) + \omega_{G_{/P}}(q_{M_i}) - \omega_G(u) + \omega_{G_{/P}}(q_{M_j})$, which proves the claim. □

Due to Lemma 3.19, one can compute the shortest path length for vertices in different modules by solving the VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS problem on the quotient graph $G_{/P}$ with vertex weights $\omega_{G_{/P}}$. The next lemma shows that for vertices that are in a same module, either the entire shortest path between them is inside this module or it is a path of length two.

Corollary 3.20. *Let $G = (V, E)$ be a graph with vertex weights $\omega_G\colon V \to \mathbb{R}_{\geq 0}$. Let $M \subseteq V$ be a module in G and $P = \{M_1, \dots, M_t\}$ be a modular partition of $G[M]$. Let further $u, v \in M$ be two vertices with $\{u, v\} \subseteq M_i$ for an $i \in [t]$. Then, every shortest u-v path in G is either completely inside $G[M]$ or there exists a shortest u-v path with exactly two edges.*

Proof. Assume that there is a shortest u-v path P in G that is not completely inside $G[M]$, i.e., it uses at least one vertex in $p \in V \setminus M$. Since M is a module in G and $u, v \in M$, every vertex $x \in V \setminus M$ is either connected to both u and v or to neither u nor v. Thus, one can shortcut $P = (u, \ldots, p, \ldots, v)$ to $P' = (u, p, v)$ and since every vertex weight is non-negative, it holds that $\omega(P') \leq \omega(P)$. □

We define the slightly more general problem k-CAPPED VERTEX-WEIGHTED APSP that takes a vertex-weighted graph G as an input and asks for all pairs of vertices u, v for the value $d_k(u, v) = \min\{dist_G(u, v), k\}$. This generalizes VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS if we set k large enough, i.e., set $k = \sum_{v \in V} \omega(v)$. We can now describe the algorithm and prove Theorem 3.16.

Algorithm. For an input graph $G = (V, E)$ with vertex weights $\omega \colon V \to \mathbb{R}_{\geq 0}$, the algorithm first computes the modular decomposition tree $\mathrm{MD}(G)$ and then processes $\mathrm{MD}(G)$ in a top-down traversal, starting with the root node. For a node v_M in $\mathrm{MD}(G)$ with children $v_{M_1}, \ldots, v_{M_t}$, let M be the corresponding module and $P = \{M_1, \ldots, M_t\}$ be the corresponding modular partition of $G[M]$. We solve k-CAPPED VERTEX-WEIGHTED APSP in $G[M]$ as follows:

First, we construct the weighted quotient graph $G_{/P}$ with vertex weights $\omega_{G_{/P}}$ as defined in Definition 3.18, and solve VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS on $G_{/P}$. Next, we compute for all pairs of vertices in $G[M]$ that are in different modules M_i the shortest distance in G by utilizing Lemma 3.19. Afterwards, we compute for each module $M_i \in P$ the minimum weight of all vertices in adjacent modules using $G_{/P}$, i.e, $k_i = \min_{q_{M_j} \in N(q_{M_i})} \omega(q_{M_j})$. Finally, we use Corollary 3.20 and recurse by solving for each module $M_i \in P$ the Problem k-CAPPED APSP on $G[M_i]$ with $k = k_i$. For the root node, we set $k = \sum_{v \in V} \omega(v)$.

Running Time. For any prime node v_{M_i} in $\mathrm{MD}(G)$, VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS can be solved in time $\mathcal{O}(n_i^{2.842})$ with an algorithm due Yuster [Yus09], where n_i denotes the number of vertices in the corresponding quotient graph. With the standard combinatorial algorithm one can solve VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS in time $\mathcal{O}(n_i^3)$. Thus, by Theorem 3.15, the total running time for this step is $\mathcal{O}(n\,\mathsf{h}^{1.842} + m)$ if we use fast matrix multiplication or $\mathcal{O}(n\,\mathsf{h}^2 + m)$ for the combinatorial algorithm. After we have solved VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS on all nodes in the modular decomposition tree, we use Lemma 3.19 to compute for each pair of vertices in G that are in different modules the length of a shortest path in constant time. Since we do this for each pair of vertices in G exactly once, this sums up to a total running time of $\mathcal{O}(n^2)$. The computation of the values k_i can be done in time $\mathcal{O}(n_i^2)$, which is dominated by the time of solving VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS. In total, we obtain a combinatorial algorithm of time $\mathcal{O}(\mathsf{h}^2\, n + n^2)$ and an algorithm using fast matrix multiplication with time complexity $\mathcal{O}(\mathsf{h}^{1.842}\, n + n^2)$.

3.5 Triangle Counting

In this section, we consider the TRIANGLE COUNTING problem, in which one is interested in the number of triangles in the input graph, i.e., the number of different K_3 subgraphs.

TRIANGLE COUNTING

Input: An undirected graph $G = (V, E)$.
Output: The number of triangles in G.

The fastest known algorithm for TRIANGLE COUNTING in terms of the number of vertices n relies on fast matrix multiplication and runs in $\mathcal{O}(n^\omega)$ time [SW05]. We present an algorithm that runs in $\mathcal{O}(\mathsf{h}^{\omega-1} n + m)$ time. Again, our running time smoothly interpolates between linear time $\mathcal{O}(n+m)$ for $\mathsf{h} = \mathcal{O}(1)$ and the best unparameterized time for $\mathsf{h} \in \Theta(n)$, making it adaptive for sufficiently dense graphs; else, the $\mathcal{O}(m^{\frac{2\omega}{\omega+1}}) = \mathcal{O}(m^{1.41})$ time algorithm of Alon et al. [AYZ97] is faster. We will prove the following theorem.

Theorem 3.21. *For every graph $G = (V, E)$ with modular-width h,* TRIANGLE COUNTING *can be solved in time $\mathcal{O}(n \cdot \mathsf{h}^{\omega-1} + m)$.*

Algorithm First, we compute the modular decomposition tree $\mathrm{MD}(G)$ of a graph $G = (V, E)$. We will process $\mathrm{MD}(G)$ in a bottom-up manner. For each v_M in $\mathrm{MD}(G)$, with the corresponding module $M \subseteq V$, we will compute the following three values: the number of vertices $n_M = |V(G[M])|$, the number of edges $m_M = |E(G[M])|$, and the number of triangles $\triangle_M$ in $G[M]$. For any leaf node v_M in $\mathrm{MD}(G)$ we have $n_M = 1$ and $m_M = \triangle_M = 0$, because $G[M]$ consists of a single vertex. Let v_M be a prime node in $\mathrm{MD}(G)$ with children $\{v_{M_1}, \ldots, v_{M_t}\}$. This means that $P = \{M_1, \ldots, M_t\}$ is a modular partition of $G[M]$. Since we process $\mathrm{MD}(G)$ in a bottom-up manner, the values for $G[M_i]$ are already computed for $i \in [t]$. We can compute the value n_M by adding up the number of vertices of each M_i and we can compute m_M by traversing all edges in the quotient graph $G[M]_{/P}$:

$$n_M = \sum_{i=1}^{t} n_{M_i}$$

$$m_M = \sum_{i=1}^{t} m_{M_i} + \sum_{\{q_{M_i}, q_{M_j}\} \in E(G[M]_{/P})} n_{M_i} n_{M_j}$$

For computing $\triangle_M$, we count triangles in $G[M]$ of three types: Triangles using vertices in exactly one module, in two (adjacent) modules, or in three different modules of P. We call a triangle with vertices in three different modules a *separated* triangle. To compute the number of separated triangles, we use the following lemma:

Lemma 3.22. *Let $G = (V, E)$ be a graph with a modular partition $P = \{M_1, \ldots, M_t\}$ and quotient graph $G_{/P}$. Let $n_{M_i} := |M_i|$ and consider the weight function $w \colon E(G_{/P}) \to \mathbb{R}^+$ with $w(\{q_{M_i}, q_{M_j}\}) = \sqrt{n_{M_i} n_{M_j}}$. Let A be the weighted adjacency matrix of $G_{/P}$ with respect to w. Then, the number of separated triangles in G is:*

$$\frac{1}{6} \sum_{i,j=1}^{t} \left(A^2 \circ A\right)_{i,j},$$

where $A \circ B$ denotes the Hadamard product of the matrices A and B, i.e., $(A \circ B)_{i,j} = A_{i,j} B_{i,j}$.

Proof. To count all separated triangles in G we need to sum up the values $n_{M_i} n_{M_j} n_{M_k}$ for each triangle $(q_{M_i}, q_{M_k}, q_{M_j})$ in $G_{/P}$. We will show, that the value $(A^2 \circ A)_{i,j}$ exactly denotes to the number of separated triangles in G with one vertex in M_i and one vertex in M_j. We define a *wedge* as a path on three vertices (and a wedge $(q_{M_i}, q_{M_k}, q_{M_j})$ requires the presence of the

edges $\{q_{M_i}, q_{M_k}\}$ and $\{q_{M_k}, q_{M_j}\}$). It now holds the following:

$$\begin{aligned}\left(A^2\right)_{i,j} &= \sum_{k=1}^{t} A_{i,k}A_{k,j} \\ &= \sum_{\substack{k:(q_{M_i},q_{M_k},q_{M_j}) \\ \text{is a wedge in } G_{/P}}} \sqrt{n_{M_i}n_{M_k}}\sqrt{n_{M_k}n_{M_j}} \\ &= \sqrt{n_{M_i}n_{M_j}} \sum_{\substack{k:(q_{M_i},q_{M_k},q_{M_j}) \\ \text{is a wedge in } G_{/P}}} n_{M_k} \\ \Rightarrow \left(A^2 \circ A\right)_{i,j} &= \sum_{\substack{k:(q_{M_i},q_{M_k},q_{M_j}) \\ \text{is a triangle in } G_{/P}}} n_{M_i}n_{M_j}n_{M_k}\end{aligned}$$

Every separated triangle $(q_{M_i}, q_{M_k}, q_{M_j})$ in $G_{/P}$ is counted a total of six times: Once for each of the three edges $\{q_{M_i}, q_{M_k}\}$, $\{q_{M_k}, q_{M_j}\}$, and $\{q_{M_i}, q_{M_j}\}$; and for each edge $\{q_i, q_j\}$ of the triangle twice, i.e., by $(A^2 \circ A)_{i,j}$ and by $(A^2 \circ A)_{j,i}$. Thus, the claim follows. □

Using Lemma 3.22, we can now compute $\triangle_M$ by

$$\triangle_M = \sum_{i=1}^{t} \triangle_{M_i} + \sum_{\{q_i,q_j\}\in E(G_{/P})} \left(m_{M_i}n_{M_j} + n_{M_i}m_{M_j}\right) + \sum_{i,j=1}^{t} \frac{1}{6}\left(A^2 \circ A\right)_{i,j},$$

where the three terms refer to triangles with vertices from only one module, triangles using vertices of two adjacent modules, and separated triangles with vertices in three different (pairwise adjacent) modules.

If v_M is a parallel or serial node, we will use the same approach as in Section 3.2 and replace v_M by $t-1$ nodes with only two children each and quotient graphs isomorphic to I_2 (in the case of a parallel node) or K_2 (in the case of a serial node).

Running Time. Computing the modular decomposition tree $\mathrm{MD}(G)$ takes time $\mathcal{O}(n+m)$. The replacement of each parallel or serial node with $t \geq 3$ children by $t-1$ 'pseudo-prime' nodes can be done in $\mathcal{O}(n)$ time. Consider a node v_M in $\mathrm{MD}(G)$ with children $\{v_{M_1}, \ldots, v_{M_t}\}$ with $t \leq \mathsf{h}$. Recall, that $P = \{M_1, \ldots, M_t\}$ is a modular partition of $G[M]$. Computing n_M takes time $\mathcal{O}(t)$ and computing m_M takes time $\mathcal{O}(|E(G[M]_{/P})|) = \mathcal{O}(t^2)$. The running time for computing $\triangle_M$ is dominated by the computation of A^2, which takes time $\mathcal{O}(t^\omega)$. Note that $2 \leq t \leq \mathsf{h}$. By Theorem 3.15, we yield a total running time of $\mathcal{O}(n \cdot \mathsf{h}^{\omega-1} + m)$, which proves Theorem 3.21. Note that this algorithm is adaptive for dense graphs, meaning that even for $\mathsf{h} \in \Theta(n)$ our algorithm is not worse than $\mathcal{O}(n^\omega)$.

3.6 Edge-Disjoint Paths

In this section, we first address the problem EDGE-DISJOINT s-t PATHS, in which one is interested in the maximum number of edge-disjoint s-t paths (equivalently, finding an unweighted minimum s-t cut) in a given graph $G = (V, E)$ with $s, t \in V$. We denote the size of a maximum number of edge-disjoint s-t paths in a graph G by $\lambda_G(s, t)$.

MAXIMUM EDGE-DISJOINT s-t PATHS

Input: An undirected graph $G = (V, E)$, $s, t \in V$.
Output: The number $\lambda_G(s, t)$ of edge-disjoint s-t paths in G.

Later, we focus on finding a global unweighted minimum cut, i.e., $\max_{s,t \in V} \lambda_G(s, t)$. The weighted variants of these problems, in particular MAXIMUM s-t FLOW with edge capacities, are unlikely to admit faster algorithms when the modular-width is low, cf. the remarks in Section 3.8.

GLOBAL MINIMUM EDGE CUT

Input: An undirected graph $G = (V, E)$
Output: The number $\lambda(G) = \min\{\lambda_G(s, t) \mid s, t \in V\}$.

3.6.1 Maximum Edge-Disjoint s-t Paths

We first consider the MAXIMUM EDGE-DISJOINT s-t PATHS problem. Using a flow algorithm, one can determine the number of edge- or vertex-disjoint s-t paths in a graph. For the unweighted case there are specialized algorithms, e.g., computing the number of edge-disjoint s-t paths in an undirected graph can be done in time $\mathcal{O}(n^{\frac{3}{2}} m^{\frac{1}{2}})$ using an algorithm due to Goldberg and Rao [GR99a]. We will prove the following theorem.

Theorem 3.23. *For every graph $G = (V, E)$ with modular-width h and $s, t \in V$,* EDGE-DISJOINT s-t PATHS *can be solved in time $\mathcal{O}(\mathsf{h}^3 + n + m)$.*

Algorithm. Let $G = (V, E)$ be a graph with $s, t \in V$. We assume that G is connected, otherwise we consider the connected component with s and t as the new input graph or conclude that $\lambda_G(s, t) = 0$ if s and t are in different connected components of G. First, we compute the modular decomposition tree $\mathrm{MD}(G)$. Instead of traversing the decomposition tree completely, we will only consider one modular partition of G.

Lemma 3.24. *Let $G = (V, E)$ be a graph, let $s, t \in V$, and let P be a modular partition of G. If there exists a module $M \in P$ with $s, t \in M$ and a module $N \in P$ that is adjacent to M, then $\lambda_G(s, t) = \min\{\deg_G(s), \deg_G(t)\}$.*

Proof. Obviously, it holds that $\lambda_G(s, t) \leq \min\{\deg_G(s), \deg_G(t)\}$. For the converse direction assume, w.l.o.g., that $\deg_G(s) \leq \deg_G(t)$. For every vertex $v \in N_G(s) \setminus M$, we consider the path (s, v, t). We stress that v is also a direct neighbor of t and that all these paths are clearly edge-disjoint. Since we assume that $\deg_G(s) \leq \deg_G(t)$, it also holds that $\deg_{G[M]}(s) \leq \deg_{G[M]}(t)$. Hence, $|N_{G[M]}(s)| \leq |N_{G[M]}(t)|$ and we can assign for every vertex $v \in N_{G[M]}(s)$ a private vertex $v' \in N_{G[M]}(t)$. Let $w \in N$ be an arbitrary vertex in the neighboring module N. For all $v \in N_{G[M]}(s)$ we either choose the path (s, v, t), if $v = v'$, or the path (s, v, w, v', t), if $v \neq v'$. For $v = t \in N_M(s)$, if it exists, we use the path (s, t). Overall this results in $\deg_G(s)$ many edge-disjoint s-t paths. This implies that $\deg_G(s) = \min\{\deg_G(s), \deg_G(t)\} \leq \lambda_G(s, t)$. □

Corollary 3.25. *Let $G = (V, E)$ be a graph, let $s, t \in V$, and let P be a modular partition of G such that $G_{/P}$ is a complete graph. Then $\lambda_G(s, t) = \min\{\deg_G(s), \deg_G(t)\}$.*

Proof. Corollary 3.25 directly follows from Lemma 3.24 if $|G_{/P}| \geq 3$; in this case there always exists a modular partition P such that s and t are in a same module $M \in P$ with an adjacent module $N \in P$. If $|G_{/P}| = 2$, the quotient graph $G_{/P}$ is isomorphic to K_2 and the claim can be verified similar to the proof of Lemma 3.24. □

Consider the root vertex v_M of MD(G) and let $\{v_{M_1}, \ldots, v_{M_\ell}\}$ be the children of v_M, i.e., let $P = \{M_1, \ldots, M_\ell\}$ be a modular partition of $G[M] = G$. Since we assume that G is connected, v_M cannot be a parallel node. If v_M is a serial node, we can conclude that $\lambda_G(s,t) = \min\{\deg_G(s), \deg_G(t)\}$ by Corollary 3.25. Let v_M be a prime node. If s and t belong to the same module, we again conclude due to Lemma 3.24 that $\lambda_G(s,t) = \min\{\deg_G(s), \deg_G(t)\}$, since every quotient graph of a prime node is connected. It remains to solve the case that v_M is a prime node but s and t do not belong to the same module. We will reduce this case to a single computation of a maximum edge-capacitated flow. For readability, we will denote in the following the set of vertices of the quotient graph $G_{/P}$ by $\{q_1, \ldots, q_\ell\}$ while each vertex q_i corresponds to the module $M_i \in P$, for $i \in [\ell]$.

Definition 3.26. Let $G = (V, E)$ be a graph, let $s, t \in V$, and let $P = \{M_1, ..., M_\ell\}$ be a modular partition of G into $\ell \geq 2$ modules. Let $s \in M_1$, let $t \in M_\ell$, and let $G_{/P}$ be the quotient graph with vertex set $\{q_1, \ldots, q_\ell\}$. We define a flow network $N = (G', q_0, q_{\ell+1}, c)$ as follows:

- The graph G' is initiated as being equal to $G_{/P}$.
- We add vertices q_0 and $q_{\ell+1}$ to $V(G_{/P})$, each with the same neighbors as q_1 resp. q_ℓ.
- We add the edges $\{q_0, q_1\}$ and $\{q_{\ell+1}, q_\ell\}$.
- The $\ell + 2$ vertices of G' correspond to the sets of vertices in the partition $P' = \{M'_0, M'_1, M'_2, \ldots, M'_\ell, M'_{\ell+1}\}$ with $M'_0 = \{s\}$, $M'_1 = M_1 \setminus \{s\}, M'_\ell = M_\ell \setminus \{t\}$, $M'_{\ell+1} = \{t\}$, and $M'_i = M_i$ for $i \in \{2, 3, \ldots, \ell - 1\}$.
- The capacities on the edges of G' represent the number of edges between the corresponding vertex sets in G, i.e. $c(q_i, q_j) = |M'_i||M'_j|$ for $\{q_i, q_j\} \in E(G') \setminus \{\{q_0, q_1\}, \{q_\ell, q_{\ell+1}\}\}$ resp. $c(q_0, q_1) = deg_{G[M_1]}(s)$ and $c(q_\ell, q_{\ell+1}) = deg_{G[M_\ell]}(t)$.

Lemma 3.27. *Let $G = (V, E)$ be a graph, let $s, t \in V$, and let $P = \{M_1, ..., M_\ell\}$ be a modular partition of G. Let $s \in M_1$, $t \in M_\ell$ and $\ell \geq 2$. Let $N = (G', q_0, q_{\ell+1}, c)$ be the flow network as defined in Definition 3.26. Then, the maximum flow in N is equal to $\lambda_G(s,t)$.*

The graph G' together with the capacities c is a compact representation of G, but without the edges inside a module (except for incident edges to s or t). In order to prove Lemma 3.27, we first observe that those edges inside modules are not helpful to get edge-disjoint paths in G. To see this, we consider the following assignment problem.

Lemma 3.28. *Let $A = \{a_1, \ldots, a_\ell\}$, let $B = \{b_1, \ldots, b_r\}$, and let $X = \{x_1, \ldots, x_k\}$ be sets of vertices and G be a graph with vertex set $A \cup B \cup X$. Let $f \colon A \cup B \to \mathbb{N}$ be a function that denotes the demand of every vertex in $A \cup B$, with the constraints $f(a_i) \leq k = |X|$ and $f(b_j) \leq k$ for all $i \in [\ell]$ and $j \in [r]$, and $\sum_{i=1}^{\ell} f(a_i) = \sum_{j=1}^{r} f(b_j)$. Then there is a set of directed arcs $E \subseteq (A \times X) \cup (X \times B)$ in G such that $\deg^+_G(a_i) = f(a_i)$, $\deg^-_G(b_j) = f(b_j)$ and $\deg^+_G(x_d) = \deg^-_G(x_d)$ for all $i \in [\ell]$, $j \in [r]$ and $d \in [k]$.*

Proof. We can solve this assignment problem with a flow computation. To do so, we construct a directed graph G as follows: The vertex set consists of A, B, X and two vertices s and t. We add edges (s, a_i) of capacity $f(a_i)$ for each $i \in [\ell]$ and denote these edges with S. In almost the same manner we add edges (b_j, t) of capacity $f(b_j)$ for each $j \in [r]$ and denote these edges with T. At last, we add all edges $A \times X$ and $X \times B$ to the graph, each with capacity one. Denote the resulting network by $N = (G, s, t, c)$. To prove the lemma we only have to show that the maximum flow in N is equal $\sum_{i=1}^{\ell} f(a_i) = c(S) = c(T)$. To do so, we observe that the minimum weighted s-t cut in N is equal to $c(S) = c(T)$: Let $C \subseteq E(G)$ be an arbitrary minimum s-t cut in G. If $S \subseteq C$ then it holds that $c(S) \leq c(C)$ and since S is an s-t cut there is indeed equality. The same applies if $T \subseteq C$. Thus, assume that $S \setminus C \neq \emptyset$ and $T \setminus C \neq \emptyset$. Let w.l.o.g. $|S \setminus C| \leq |T \setminus C|$. Let $D = C \cap (S \cup T)$ and let $A' \subseteq A$, resp. $B' \subseteq B$, be the set of vertices of A, resp.B, that are not incident to an edge in D. Since $|S \setminus C| \leq |T \setminus C|$ it holds that $|A'| \leq |B'|$. It is easy to see that there are $k \cdot |A'|$ edge disjoint paths between A' and B'. Hence, to augment $S \cap C$ to an s-t cut without taking edges in $S \cup T$ one needs to take at least $k \cdot |A'| = k \cdot |S \setminus C|$ edges. Therefore, $c(C) \geq c(S \cap C) + k \cdot |S \setminus C| \geq c(S \cap C) + c(S \setminus C) = c(S)$. Again, since S is an s-t cut in N, there is indeed equality. □

Corollary 3.29. *Let $G = (V, E)$ be a graph, let $s, t \in V$, and let $P = \{M_1, ..., M_\ell\}$ be a modular partition of G into $\ell \geq 2$ modules with $s \in M_1$ and $t \in M_\ell$. Let further $P' = \{M'_0, M'_1, M'_2, \ldots, M'_\ell, M'_{\ell+1}\}$ be the partition of $V(G)$ as defined in Definition 3.26. Then, there exists a set of edge-disjoint s-t paths in G of size $\lambda_G(s, t)$ with the property that no path used edge inside a vertex set $M \in P'$.*

Proof. Consider a maximum set of edge-disjoint s-t paths in G. Assume that there is a path that uses an edge inside a vertex set $M \in P'$. Note that $M'_0 = \{s\}$ and $M_{\ell+1} = \{t\}$, implying $M \neq M'_0$ and $M \neq M'_{\ell+1}$. Thus, every path traversing nodes in M visits a vertex before and after M. Orient every path to a directed path from s to t (since the paths are edge-disjoint, this is possible). Denote the set of those directed edges by D. We can apply Lemma 3.28 to rearrange the paths, such that no edge inside M is used, by setting $X = M$, $A = \{v \in V \mid (v, m) \in D \wedge m \in M\}$ and $B = \{v \in V \mid (m, v) \in D \wedge m \in M\}$. Additionally, we set the demand $f(a) = |\{m \in M \mid (a, m) \in D\}|$ for $a \in A$ and $f(b) = |\{m \in M \mid (m, b) \in D\}|$ for $b \in B$. □

Proof of Lemma 3.27. Let $MF(N)$ denote the maximum flow value in $N = (G', q_0, q_{\ell+1}, c)$. Let $P' = \{M'_0, M'_1, M'_2, \ldots, M'_\ell, M'_{\ell+1}\}$ be the partition of V corresponding to the vertices in G'. Any flow in N corresponds to edge-disjoint paths in G not using edges inside a set of P', yielding $MF(N) \leq \lambda_G(s, t)$. Conversely, by Corollary 3.29 we can modify any maximum set of edge disjoint s-t paths to a set of edge-disjoint s-t paths that does not use edges inside a vertex set of P'. Again, any such set of edge-disjoint path corresponds to a flow in N, proving $MF(N) \geq \lambda_G(s, t)$. □

Running Time. Consider a connected graph $G = (V, E)$ with modular-width h and let $s, t \in V$. Computing the modular decomposition tree takes time $\mathcal{O}(n + m)$. Let v_M be the root module of the decomposition tree $MD(G)$ with children $\{v_{M_1}, \ldots, v_{M_\ell}\}$, i.e., $P = \{M_1, \ldots, M_\ell\}$ is a modular partition of $G[M] = G$. If v_M is a series node or s and t are in a same module in P, we can compute $\lambda_G(s, t)$ by Lemma 3.24 in time $\mathcal{O}(m)$. Otherwise, we use the network defined in Definition 3.26. Computing this network takes time $\mathcal{O}(|V(G')| + |E(G')|) = \mathcal{O}(\ell^2)$. Then, we can compute $\lambda_G(s, t)$ in time $\mathcal{O}(\ell^3)$ using the maximum flow algorithm by Orlin [Orl13]. Since $\ell \leq \mathsf{h}$ we have proven Theorem 3.23. Note that we apply the $\mathcal{O}(nm)$ time algorithm by Orlin only once on the quotient graph of the root node $v_M \in \mathrm{MD}(G)$, where h^2 is only an upper bound

on the number of edges in $G[M]_{/P}$. For sparse graphs, we can also bound the number of edges in $G[M]_{/P}$ by the number of edges in G, giving us the running time of $\mathcal{O}(\min\{\mathsf{h}\, m, \mathsf{h}^3\} + n + m)$.

Kernel. The algorithm with running time $\mathcal{O}(\mathsf{h}^3 + n + m)$ can be easily modified to compute a kernel in linear time: We can compute in time $\mathcal{O}(n+m)$ an equivalent instance of a maximum flow problem of size $\mathcal{O}(\mathsf{h}^2)$, which can be solved in time $\mathcal{O}(\mathsf{h}^3)$. Such results were also achieved by Coudert et al. [CDP19] for ECCENTRICITIES, HYPERBOLICITY, and BETWEENNESS CENTRALITY parameterized by modular-width. It is easy to see that this holds in general for *any* algorithm with running time $\mathcal{O}(f(k) + n + m)$: The running time is either dominated by $\mathcal{O}(f(k))$ or by $\mathcal{O}(n+m)$; we run the algorithm for $c \cdot (n+m)$ steps (for c large enough), either it terminates or we can conclude that $f(k) \geq c \cdot (n+m)$ and our input graph is already a kernel of size $\mathcal{O}(f(k))$.[7]

3.6.2 Global Minimum Cut

Now, we turn our focus onto computing the global minimum (edge) cut for an unweighted graph $G = (V, E)$, i.e. $\lambda(G) = \min\{\lambda_G(s,t) \mid s,t \in V\}$. We will reduce the computation of a global (unweighted) minimum cut of G to a single computation of a global weighted minimum cut in a graph closely related to the quotient graph of the root module. For this, we modify the algorithm for finding a minimum s-t cut for fixed $s, t \in V$ seen in Section 3.6.1. We will prove the following theorem.

Theorem 3.30. *For every graph $G = (V, E)$ with modular-width h,* GLOBAL MINIMUM EDGE CUT *can be solved in time $\mathcal{O}(\mathsf{h}^3 + n + m)$.*

Algorithm. Consider a graph $G = (V, E)$. We can assume that G is connected, otherwise it holds that $\lambda(G) = 0$. First, we compute the modular decomposition tree $\mathrm{MD}(G)$. Let v_M be the root node of $MD(G)$. If v_M is a serial node it follows from Corollary 3.25 that $\lambda_G(s,t) = \min\{\deg_G(s), \deg_G(t)\}$ for all pairs of vertices $s, t \in V$; therefore, if v_M is a serial node it holds that $\lambda(G) = \min_{v \in V} \deg_G(v)$. Assume that v_M is a prime node and let $\{v_{M_1}, \ldots, v_{M_\ell}\}$ be the children of v_M in $MD(G)$, i.e., $P = \{M_1, \ldots, M_\ell\}$ is modular partition of $G[M] = G$. Let $(s^*, t^*) = \arg\min\{\lambda_G(s,t) \mid s,t \in V\}$. Obviously, it holds that $\lambda(G) \leq \min_{v \in V} \deg_G(v)$. If s^* and t^* belong to the same module $M_i \in P$ then it holds that $\lambda(G) = \min_{v \in V} \deg_G(v)$ by Lemma 3.24. It is only possible that $\lambda(G) < \min_{v \in V} \deg_G(v)$, if s^* and t^* are in different modules. The following lemma shows that s^* and t^* are necessarily vertices of minimum degree in a module.

Lemma 3.31. *Let $G = (V, E)$ be a graph and $P = \{M_1, \ldots, M_\ell\}$ be a modular partition of G with $\ell \geq 2$. Let $(s^*, t^*) = \arg\min\{\lambda_G(s,t) \mid s \in M_i, t \in M_j, i \neq j\}$. Then, it is possible to pick s^* and t^* as vertices of minimum degree in their modules.*

Proof. Let s and t be two arbitrary vertices with $\{s,t\} \not\subseteq M_i$ for all $i \in [\ell]$. After a possible renaming of the modules, we can assume that $s \in M_1$ and $t \in M_\ell$. As shown in Section 3.6.1, one can compute $\lambda_G(s,t)$ by computing a maximum flow in the network $N = (G', q_0, q_{\ell+1}, c)$ as defined in Definition 3.26. The graph G' will be the same for all $s \in M_1$ and $t \in M_\ell$, but the capacities on the edges $\{q_0, q_1\}$ and $\{q_\ell, q_{\ell+1}\}$ do change. These capacities are equal to $deg_{G[M_1]}(s)$, resp. $deg_{G[M_\ell]}(t)$. Thus, they are minimal if we choose $s \in M_1$ and $t \in M_\ell$ such that s and t have minimum degree in M_1 resp. M_ℓ, which proves the claim. □

[7] This can be generalized in an obvious way to running times of type $\mathcal{O}(f(k) + g(N))$, where N denotes the input size.

Now, we can create an auxiliary graph that is similar to graph G' in Definition 3.26, in order to compute $\lambda(G)$.

Definition 3.32. Let $G = (V, E)$ be a graph and let $P = \{M_1, ..., M_\ell\}$ be a modular partition of G into $\ell \geq 2$ modules. Let $v_i \in M_i$ denote a vertex of minimum degree in M_i for $i \in [\ell]$. Let $G_{/P}$ be the quotient graph with vertex set $\{q_1, \ldots, q_\ell\}$. We define a edge-capacitated graph $G^* = (V^*, E^*)$ with capacities $c\colon E^* \to \mathbb{N}$ as follows:

- The graph G^* is initiated as being equal to $G_{/P}$.
- We add vertices $q_{\ell+i}$ to $V(G_{/P})$, each with the same neighbors as q_i for $i \in [\ell]$.
- We add the edges $\{q_i, q_{i+\ell}\}$ to E^* for $i \in [\ell]$.
- The 2ℓ vertices of G^* correspond to the sets of vertices in the partition $P' = \{M'_1, M'_{\ell+1}, M'_2, M'_{\ell+2}, \ldots, M'_\ell, M'_{2\ell}\}$ with $M'_i = \{v_i\}$ and $M'_{\ell+i} = M_i \setminus \{v_i\}$ for $i \in [\ell]$.
- The capacities on the edges of G^* represent the number of edges between the corresponding vertex sets in G, i.e.
  $c(q_i, q_j) = |M'_i||M'_j|$ for $\{q_i, q_j\} \in E(G') \setminus \{\{q_k, q_{k+\ell}\} \mid k \in [\ell]\}$ with $i, j \in [2\ell]$ and
  $c(q_i, q_{i+\ell}) = deg_{G[M_i]}(v_i)$ for $i \in [\ell]$.

Lemma 3.33. *Let $G = (V, E)$ be a graph and $P = \{M_1, \ldots, M_\ell\}$ be a modular partition of G into $\ell \geq 2$ modules. Let (G^*, c) be the weighted auxiliary graph defined in Definition 3.32. Then $\lambda(G) = \lambda((G^*, c))$.*

Proof. Let v_i be a vertex of minimum degree in M_i. We know that $\lambda(G) = \min\{\lambda_G(s, t) \mid s, t \in V\}$ is the lowest maximum s-t flow in G considering every pair of vertices $s, t \in V$. By Lemma 3.31, we only need to consider vertices of minimum degree in each module. First, we observe that a maximum v_i-v_j flow in G (with capacities all equal to one) has the same value as a maximum q_i-q_j flow in G^* (with the capacity function c as defined in Definition 3.32) by the same argument as in the proof of Lemma 3.27. Therefore, $\lambda(G) \geq \lambda((G^*, c))$. For the converse, we observe that $\lambda((G^*, c))$ corresponds to a maximum flow between two vertices q_i and q_j with $i, j \leq \ell$, because each pair $(q_i, q_{i+\ell})$ has the exact same neighborhood, but the capacities on incident edges of q_i are smaller than the capacities on incident edges of $q_{i+\ell}$. Since every q_i-q_j flow in G^* with $i, j \leq \ell$ has the same value as a v_i-v_j flow in G, we have proven the claim. □

Running Time. Consider a graph $G = (V, E)$ with modular-width h. Computing the modular decomposition tree $\mathrm{MD}(G)$ takes linear time $\mathcal{O}(n+m)$. Let v_M be the root node of $MD(G)$ with children $\{v_{M_1}, \ldots, v_{M_\ell}\}$, i.e, $P = \{M_1, \ldots, M_\ell\}$ is a modular partition of $G[M] = G$. If v_M is a parallel node or a serial node, we can compute $\lambda(G)$ in time $\mathcal{O}(m)$. Assume v_M to be a prime node. Generating the instance (G^*, c) according to Definition 3.32 takes time $\mathcal{O}(|V(G^*)|+|E(G^*)|) = \mathcal{O}(\ell^2)$. Note that $|V(G^*)| = 2\ell$. Computing a weighted global minimum edge cut in the undirected graph G^* can be done in time $\mathcal{O}(\ell^3)$ by using the algorithm of Stoer and Wagner [SW97]. Since $\ell \leq \mathsf{h}$ we have proven Theorem 3.30. Note that the running time of this algorithm is actual only depending on the size of the quotient graph of the root node, which is only upper bounded by the modular-width of the graph (but might be much smaller). Again, since the algorithm by Stoer and Wagner has a time complexity of $\mathcal{O}(nm + n^2 \log n)$ on graphs with n vertices and m edges, it is also possible to bound the running time of our algorithm by $\mathcal{O}(\mathsf{h}\, m + \mathsf{h}^2 \log \mathsf{h})$.

3.7 Vertex-Disjoint Paths

A connected graph $G = (V, E)$ is said to be *k-vertex-connected* if one can delete up to $k-1$ arbitrary vertices and G stays connected. The *vertex-connectivity* of G, denoted by $\kappa(G)$, is the largest k for which G is k-vertex-connected. In other words, $\kappa(G) = \min\{\kappa_G(s,t) | s,t \in V, \{s,t\} \notin E\}$ and $\kappa_G(s,t)$ denotes the minimum size of an s-t-vertex separator in G. By Menger's Theorem [Men27], for $s,t \in V$, the minimum size of an s-t-vertex separator is exactly the size of the maximum number of vertex-disjoint s-t paths, denoted by $\Pi_G(s,t)$. The latter one is as well of independent interest. One can compute $\Pi_G(s,t)$ by solving an s-t vertex capacitated flow with capacities equal to one for every vertex. Instead of focusing on $\Pi_G(s,t)$, we directly solve the more general problem of computing a maximum s-t vertex-capacitated flow with an arbitrary capacity function $c\colon V \setminus \{s,t\} \to \mathbb{R}^+$.

MAXIMUM s-t VERTEX FLOW

Input: A Network (G, s, t, c) with $G = (V, E)$, $s, t \in V$, $c\colon V \setminus \{s,t\} \to \mathbb{R}^+$.
Output: A maximum s-t vertex-capacitated flow f.

Computing the MAXIMUM s-t VERTEX FLOW in a vertex-capacitated graph will be the focus of Section 3.7.1. In Section 3.7.2 we will then focus on computing $\kappa(G)$, but again in the more general setting with vertex capacities.

GLOBAL MINIMUM VERTEX CUT

Input: An undirected graph $G = (V, E)$, capacity function $c\colon V \to \mathbb{R}^+$.
Output: A set $X \subseteq V$ of minimum capacity such that $G - X$ is disconnected.

3.7.1 Maximum s-t Vertex Flow

By a simple reduction from vertex-capacitated flow to directed edge-capacitated flow one can solve MAXIMUM S-T VERTEX FLOW in time $\mathcal{O}(nm)$ using an algorithm by Orlin [Orl13]. In this section, we will prove the following theorem.

Theorem 3.34. *Let $N = (G, s, t, c)$ be a network with $G = (V, E)$ be a graph with modular-width h, $s, t \in V$, and $c\colon V \to \mathbb{R}^+$. Then, we can compute a maximum s-t vertex-capacitated flow in N in time $\mathcal{O}(\mathsf{h}^3 + n + m)$.*

Algorithm. Consider a network $N = (G, s, t, c)$ consisting of a graph $G = (V, E)$ with $s, t \in V$ and a capacity function $c\colon V \setminus \{s,t\} \to \mathbb{R}^+$. We want to compute a maximum s-t flow in N. Assume that $\{s,t\} \notin E$, otherwise the maximum s-t flow is unbounded. Our algorithm will traverse the modular decomposition tree $\mathrm{MD}(G)$ in a bottom-up manner. Let v_M be the node in $\mathrm{MD}(G)$ that corresponds to the smallest module M with $s, t \in M$ (that is the lowest common ancestor of the two leaves in $\mathrm{MD}(G)$ corresponding to $\{s\}$ and $\{t\}$). We distinguish three cases to compute a maximum s-t flow in $G[M]$. If v_M is a parallel node, the maximum s-t flow in $G[M]$ is zero. If v_M is a serial node, s and t are adjacent (which we ruled out). For the case of v_M being a prime node in $\mathrm{MD}(G)$, let $\{v_{M_1}, .., v_{M_\ell}\}$ be the set of children of v_M. This means that $P = \{M_1, .., M_\ell\}$ is a modular partition of $G[M]$. Let $G[M]_{/P}$ be the quotient graph of $G[M]$ with the vertex set $\{q_1, \ldots, q_\ell\}$. We assume, w.l.o.g. , that $s \in M_1$ and $t \in M_\ell$. The following lemma shows that the maximum s-t flow in $(G[M], s, t, c)$ can be computed as

the maximum q_1-q_ℓ flow in $G[M]_{/P}$ where the capacity of q_i is simply the sum of capacities of the vertices of its corresponding module.

Lemma 3.35. *Let $N = (G, s, t, c)$ be a flow network with a graph $G = (V, E)$, $s, t \in V$, and $c\colon V \setminus \{s, t\} \to \mathbb{R}^+$. Let $P = \{M_1, \ldots, M_\ell\}$ be a partition of V into modules with $s \in M_1$ and $t \in M_\ell$. Then, the maximum flow in N is equal to the maximum flow in $N' = (G_{/P}, q_1, q_\ell, c')$ with the capacity function $c'\colon V(G_{/P}) \setminus \{q_1, q_\ell\} \to \mathbb{R}^+$ defined by $c'(q_i) = \sum_{v \in M_i} c(v)$.*

Proof. Recall that for an undirected graph $G = (V, E)$ we denote by $\overleftrightarrow{G} = (V, \overleftrightarrow{E})$ the directed graph with $\overleftrightarrow{E} = \{(u, v), (v, u) \mid \{u, v\} \in E\}$. Let $f'\colon E(\overleftrightarrow{G_{/P}}) \to \mathbb{R}^+$ be a maximum flow in N'. Then, f' corresponds to a flow f in N not using any edges inside the modules. Therefore, the size of maximum flow in N is at least the size of maximum flow in N'.

Conversely, consider a maximum flow $f\colon E(\overleftrightarrow{G}) \to \mathbb{R}^+$ of N. We show that there is always a maximum flow in N that does not use edges *inside* a module. Consider a maximum flow f in N that does use edges inside a module. It is well known that a flow in a graph can be decomposed into flows along paths in the graph. Thus, in any decomposition of f there must be an s-t path $Q = (s = v_1, v_2, \ldots, v_r = t)$ where some v_i and v_{i+1}, with $i \in [r-1]$, are contained in the same module M. We can replace flow along Q by sending the same amount of flow along any minimal subpath Q' of Q, which can be obtained by repeatedly shortcutting Q along edges between vertices that are not consecutive in Q. Clearly, this does not affect used capacity at vertices in Q' and only decreases used capacity at vertices that are in Q but not Q'. It is easy to see that the shortcutting operation keeps at most one of v_i and v_{i+1} in Q' (that is, at most one vertex from the subpath of Q in M is retained).

Applying the above method iteratively, we have modified the flow f such that the flow does not use edges inside a module. Such a flow f in N directly corresponds to a flow f' in N' with $|f'| = |f|$. □

After computing the maximum s-t flow in $G[M]$, we can directly compute the maximum s-t flow in G:

Lemma 3.36. *Let $N = (G, s, t, c)$ be a network with a graph $G = (V, E)$, vertices $s, t \in V$, and a capacity function $c\colon V \setminus \{s, t\} \to \mathbb{R}^+$ and let $M \subseteq V$ be a module of G with $s, t \in M$. Let f_M be a maximum s-t flow in $G[M]$. Then the maximum flow value in N, denoted by $MF(N)$, is equal to:*

$$MF(N) = |f_M| + \sum_{v \in N_G(s) \setminus M} c(v)$$

Proof. Since $s, t \in M$, it holds that every vertex $v \in N_G(s) \setminus M$ is also a neighbor of t. Hence, we can augment the flow f_M for every $v \in N_G(s) \setminus M$ by the augmenting path (s, v, t) with capacity $c(v)$. Since M is a module, all vertices adjacent to a vertex in M are used with full capacity, hence, there is no further augmenting path using vertices in $V \setminus M$. However, since f_M is already a maximum s-t flow in $G[M]$, there is also no further augmenting path completely in $G[M]$. □

Running Time. Consider a graph $G = (V, E)$ with modular-width h and a network $N = (G, s, t, c)$ with $s, t \in V$ and $c\colon V \setminus \{s, t\} \to \mathbb{R}^+$. Computing the modular decomposition tree $\mathrm{MD}(G)$ takes time $\mathcal{O}(n + m)$. Determining the node v_M in $\mathrm{MD}(G)$ that corresponds to the smallest module M with $s, t \in M$ (i.e. the lowest common ancestor of the two leaf nodes in $MD(G)$ that corresponds to the graph only consisting of vertex s resp. t) takes time $\mathcal{O}(n)$. We

can compute the size of a maximum s-t flow in $G[M]$ by either concluding that the flow is equal to zero (if v_M is parallel) or by using Lemma 3.35 (if v_M is prime). The latter can be done in time $\mathcal{O}(\ell^3)$ using the algorithm due to Orlin [Orl13] where $\ell \leq \mathsf{h}$ denotes the size of the quotient graph of M. Note that M cannot be series, since we assume $\{s,t\} \notin E$. Due to Lemma 3.36, we can compute the maximum flow in N in additional $\mathcal{O}(n)$ time, which proves Theorem 3.34. Again, one can also bound the computation of the maximum flow in $G_{/P}$ by $\mathcal{O}(\mathsf{h}\, m)$, giving us the adaptive running time of $\mathcal{O}(\min\{\mathsf{h}\, m, \mathsf{h}^3\} + n + m)$.

3.7.2 Global Minimum Vertex Cut

In the GLOBAL MINIMUM VERTEX CUT problem we are given an input graph $G = (V, E)$ and a capacity function $c\colon V \to \mathbb{R}^+$ and need to compute a set $X \subseteq V$ of minimum capacity such that $G - X$ is disconnected. This is equivalent to finding a pair of vertices $s, t \in V$ such that the maximum s-t vertex flow is minimized. We denote the minimum s-t vertex cut, respectively the maximum s-t flow in G, with capacity function c by $\Pi_{(G,c)}(s,t)$ and omit c if the capacity function is clear. We denote by $\Pi((G,c))$ the global minimum vertex cut, thus $\Pi((G,c)) = \min_{s,t \in V(G)} \Pi_{(G,c)}(s,t)$ and again omit c if the capacity function is clear. We set $\Pi_{(G,c)}(s,t) = \infty$, if s and t are adjacent. The naive approach to solve this problem is to calculate for each pair of vertices the maximum vertex-capacitated flow. Henzinger et al. [HRG00] showed how to solve this problem in time $\mathcal{O}(\kappa_1 nm \log(n^2/m))$, where κ_1 denotes the vertex connectivity if vertex capacities are ignored (or equivalently, if all capacities are set to one). We will prove the following theorem.

Theorem 3.37. *For every graph $G = (V, E)$ with modular-width at most h and vertex capacities $c\colon V \to \mathbb{R}^+$,* GLOBAL MINIMUM VERTEX CUT *can be solved in time $\mathcal{O}(\kappa_1\, \mathsf{h}^2\, n \log(\mathsf{h}) + m)$.*

Algorithm. Consider a graph $G = (V, E)$ and a capacity function $c\colon V \to \mathbb{R}^+$. First, we compute the modular decomposition tree $\mathrm{MD}(G)$. For every node v_M in $\mathrm{MD}(G)$, we will compute the total capacity of the corresponding module $c(M) := \sum_{v \in M} c(v)$ and the size of a minimum vertex cut $\Pi(G[M])$. Then, for the root node v_M in $\mathrm{MD}(G)$, we have eventually computed $\Pi(G[M]) = \Pi(G)$. We traverse the decomposition tree in a bottom-up manner. We set $\Pi(G[M]) = \infty$ for all leaf modules v_M. In the following, for computing $\Pi(G[M])$ for any module M corresponding to a node v_M in $MD(G)$, we can always assume that we have already computed the size of a minimum vertex cut for all child nodes. The next lemma shows that this information is enough to compute the minimum vertex cut in the parent node.

Lemma 3.38. *Let $G = (V, E)$ and M be a module of G. Let $s, t, u, v \in M$. If $\Pi_{G[M]}(s,t) \leq \Pi_{G[M]}(u,v)$ then also $\Pi_G(s,t) \leq \Pi_G(u,v)$.*

Proof. By Menger's Theorem, $\Pi_{G[M]}(s,t)$ and $\Pi_{G[M]}(u,v)$ both correspond to a maximum vertex flow in $G[M]$. Due to Lemma 3.36, the maximum flow in G increases by the same amount for both values. □

Let v_M be a node in $\mathrm{MD}(G)$ with children $\{v_{M_1}, \ldots, v_{M_\ell}\}$. This means $P = \{M_1, \ldots, M_\ell\}$ is a modular partition of $G[M]$. If v_M is a parallel node, it holds that $\Pi(G[M]) = 0$. Now, assume that v_M is a serial node. Again, we compute $c(M) = \sum_{i \in [\ell]} c(M_i)$. To compute $\Pi(G[M])$, we first observe that for all $s \in M_i$, $t \in M_j$ with $i \neq j$ we have $\Pi_G[M](s,t) = \infty$, because they are adjacent. Therefore, a minimum vertex cut in $G[M]$ has to be an s-t vertex cut with s and t being in the same module M_i. Hence, we can compute $\Pi(G[M])$ by extending every minimum

cut in a module by the summation of the capacities of the neighboring modules and taking the minimum:

$$\begin{aligned}\Pi(G[M]) &= \min_{i\in[\ell]} \left\{ \Pi(G[M_i]) + \sum_{k\in[\ell]\setminus\{i\}} c(M_k) \right\} \\ &= \min_{i\in[\ell]} \{\Pi(G[M_i]) + c(M) - c(M_i)\}\end{aligned}$$

Finally, assume that v_M is a prime node. There are two different types of vertex cuts in $G[M]$. The first type of vertex cut is an s-t cut for vertices s and t in a same module, in the following denoted by $\hat{\Pi}(G[M])$. In this case, every vertex cut in $G[M_i]$ has to be extended to a vertex cut in $G[M]$ by adding the capacities of neighboring modules, formally we have

$$\hat{\Pi}(G[M]) = \min_{i\in[t]} \left\{ \Pi(G[M_i]) + \sum_{j:\{q_i,q_j\}\in E\left(G[M]_{/P}\right)} c(M_j) \right\}.$$

The second type of vertex cuts in $G[M]$ is an s-t vertex cut with s and t being in different (and non-adjacent) modules. The following Lemma shows that in this case we can compute the maximum vertex cut in $G[M]$ by computing a maximum vertex cut in $G[M]_{/P}$ with a capacity function $c'\colon V(G[M]_{/P}) \to \mathbb{R}^+$ defined by $c'(q_i) = c(M_i)$.

Lemma 3.39. *Let $G = (V, E)$ be a graph with a modular partition P and let $s, t \in V$ with $\{s, t\} \notin E$. Let $M_s, M_t \in P$ with $M_s \neq M_t$, $s \in M_s$ and $t \in M_t$. Let $X \subseteq V$ be a minimum s-t vertex cut in G. Then, for every module $M \in P$, either $M \subseteq X$ or $M \cap X = \emptyset$.*

Proof. Assume for contradiction that there is a minimum s-t vertex cut X in G and that there is a module $M \in P$ with $M \cap X = X_M$ and $\emptyset \subsetneq X_M \subsetneq M$. We claim that in this case $X' = X \setminus X_M$ is an s-t vertex cut in G, contradicting the minimality of X.

Suppose X' is not an s-t vertex cut in G. Consider any s-t path Q that contains at most one vertex in each module. Such a path exists by elementary properties of modules. There must be a vertex $p \in X_M$ in Q as $X = X' \cup X_M$ is an s-t vertex cut. Clearly, replacing p by any vertex $q \in M \setminus X_M \neq \emptyset$ yields another s-t path that also avoids X, a contradiction. □

After computing the global minimum cut in the quotient graph with capacity function c', we simply need to compare this value with $\hat{\Pi}(G[M])$ and choose the smaller value as the minimum cut for $G[M]$.

Running Time. Let $G = (V, E)$ be a graph with capacity function $c\colon V \to \mathbb{R}^+$ that is an instance for the GLOBAL MINIMUM VERTEX CUT problem and let h be the modular-width of G. The modular decomposition tree $\mathrm{MD}(G)$ can be computed in linear time. For every node v_M in $\mathrm{MD}(G)$, computing $c(M)$ takes total time $\mathcal{O}(n)$, since it will be iteratively computed from the values of the child nodes and $\mathrm{MD}(G)$ has less than $2n$ nodes. By the same argument, the total running time for all serial nodes is $\mathcal{O}(n)$. Let v_M be a prime node in $\mathrm{MD}(G)$ with children $\{v_{M_1}, \ldots, v_{M_\ell}\}$, i.e., $P = \{M_1, \ldots, M_\ell\}$ is a modular partition of $G[M]$. Note that $\ell \leq \mathsf{h}$. The value $\hat{\Pi}(G[M])$ can be computed in time $\mathcal{O}(\ell^2)$. We can find a global minimum vertex capacitated cut in $G[M]_{/P}$ in time $\mathcal{O}(\kappa_1' n_i m_i \log(n_i^2/m_i))$ due to Henzinger et al. [HRG00] where κ_1' denotes the unweighted vertex connectivity in $G[M]_{/P}$ and $n_i \leq \mathsf{h}$ resp. $m_i \leq \mathsf{h}^2$ denotes the number of vertices resp. edges in $G[M]_{/P}$. Note that $\kappa_1' \leq \min\{\kappa_1, \mathsf{h}\}$. By Theorem 3.15, this results in a total running time of $\min\{\mathcal{O}(\kappa_1 \mathsf{h}\, m \log(\mathsf{h})), \mathcal{O}(\kappa_1 \mathsf{h}^2\, n \log(\mathsf{h}) + m)\}$ which proves Theorem 3.37. In both running times one can also replace κ_1 by h.

3.8 Conclusion

We have obtained efficient parameterized algorithms for the problems MAXIMUM MATCHING, MAXIMUM b-MATCHING, VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, TRIANGLE COUNTING, and several path- and flow-type problems with respect to the modular-width h of the input graph. All time bounds are of form $\mathcal{O}(f(\mathsf{h})n+m)$, $\mathcal{O}(n+f(\mathsf{h})m)$, $\mathcal{O}(f(\mathsf{h})+n+m)$, resp. $\mathcal{O}(f(\mathsf{h})n+n^2)$ for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, where running times of form $\mathcal{O}(f(\mathsf{h})+n+m)$ can be seen to imply linear-time preprocessing to size $\mathcal{O}(f(\mathsf{h}))$. Throughout, the dependence $f(\mathsf{h})$ is very low and several algorithms are adaptive in the sense that their time bound interpolates smoothly between the trivial lower bound of the problem when $\mathsf{h} = \mathcal{O}(1)$ and the best known unparameterized running time when $\mathsf{h} \in \Theta(n)$. Thus, even if typical inputs may have modular width $\Theta(n)$ (a caveat that all structural parameters face to some degree), using these algorithms costs only a constant-factor overhead and already $\mathsf{h} = o(n)$ yields an improvement over the unparameterized case. We stress that all running times in this chapter also include the time to compute the modular-decomposition tree; a quality that do not many structural parameters share.

As mentioned in the introduction, (low) modular-width seems useless in problems where edges are associated with weights and/or capacities. Intuitively, these numerical values distinguish edges between adjacent modules M and M', which could otherwise be treated as largely equivalent. For concreteness, consider an instance (G, s, t, w) of the SHORTEST s-t PATH problem where $G = (V, E)$ is a graph, $s, t \in V$, and $w\colon E(G) \to \mathbb{N}$ denotes the edge weights. Clearly, the distance from s to t is unaffected if we add the missing edges of G and let their weight exceed the sum of weights in w. However, the obtained graph is a clique and has constant modular-width. Similar arguments work for other edge-weighted/capacitated problems like MAXIMUM s-t FLOW using either huge or negligible weights. In each case, running times of form $\mathcal{O}(f(\mathsf{h})g(n,m))$ would imply time $\mathcal{O}(g(n,m))$ for the unparameterized case (without considering modular-width), so the best such running times cannot be outperformed even for low modular-width.

Apart from developing further efficient (and adaptive) parameterized algorithms relative to modular-width there are other directions of future work. Akin to conditional lower bounds via fine-grained analysis of algorithms it would be interesting to prove optimality of efficient parameterized algorithms for all regimes of the parameters (e.g., like Bringmann and Künnemann [BK18]). Which other graph parameters allow for adaptive parameterized running times so that even nontrivial upper bounds on the parameter imply faster algorithms than the unparameterized worst case? A way more general parameter than modular-width is the parameter clique-width. The clique-width is always at most the modular-width, actual, the clique-width of a graph G is the maximum clique-width of all quotient graphs of a prime node in a modular decomposition tree. The clique-width is also bounded on graph of bounded tree-width or bounded shrub-depth. For some of the considered problems in this chapter, we will give parameterized algorithm regarding the clique-width of the input in the next chapter.

Recent result for flow computation. Very recently, Chen et al. [CKL+22] claimed in a breakthrough result an $\mathcal{O}(m^{1+o(1)} \log U)$ time randomized algorithm for computing a maximum s-t flow in a graph G with integral capacities in $[U]$, where m resp. n denotes the number of vertices resp. edges in G. Via standard reduction between edge-capacitated and vertex-capacitated flows, this result also hold for vertex-capacitated flows. Li et al. [LP20] have proven that one can compute a global edge min cut in undirected weighted graphs using $polylog(n)$ calls to any maximum flow subroutine, which proves that one can solve GLOBAL MINIMUM EDGE CUT

in time $\mathcal{O}(m^{1+o(1)})$. In [LNP+21] it is shown that one can compute a global minimum vertex cut in time $\tilde{\mathcal{O}}(m^{\alpha})$ for any $\alpha \geq 1$ if there is a m^{α}-time algorithm for maximum flow, which results in a running time of $\mathcal{O}(m^{1+o(1)})$ for GLOBAL MINIMUM VERTEX CUT. This improves the running time of the best unparameterized algorithms for the problems in the four four lowermost rows in Table 3.1 to $\mathcal{O}(m^{1+o(1)})$, assuming polynomially bounded vertex capacities for the latter two.

Using those algorithms as black box routines in the parameterized algorithm, we can improve our results to $\mathcal{O}(\mathsf{h}^{2+o(1)} + n + m)$ for MAXIMUM EDGE-DISJOINT s-t PATHS, GLOBAL MINIMUM EDGE CUT, and MAXIMUM s-t VERTEX FLOW, resp. to $\mathcal{O}(m\,\mathsf{h}^{o(1)})$ for GLOBAL MINIMUM VERTEX CUT, where h denotes the modular-width of the input graph. For the former three problems, one can actually bound the running time by $\mathcal{O}(m_{G_{/P}}^{1+o(1)} + n + m)$ where $m_{G_{/P}} \leq \mathsf{h}^2$ denotes the number of edges in the quotient graph attached to the root node of the modular decomposition tree, resp. in the quotient graph attached to the lowest common ancestor of the two leaf nodes that corresponds to the graphs only consisting of vertex s resp. t.

Summarizing, we can still apply the linear-time preprocessing to get a kernel with at most $\mathcal{O}(\mathsf{h})$ many vertices, and then using the improved algorithm due to Chen et al. to get adaptive algorithms. However, since the computation of the modular decomposition does already take $\mathcal{O}(n + m)$, the speedup even for graph with constant modular-width is marginal to the almost linear time unparameterized algorithms.

4

Algorithms Parameterized by the Clique-Width of the Input

Clique-width is one of the most studied parameters in theoretical computer science. As modular-width (cf. Section 3.1), clique-width captures the closeness of a graph to a cograph, with cographs being exactly the graphs of clique-width at most two. However, clique-width is a strict generalization of modular-width. In fact, the clique-width of a graph G is equal to the maximum clique-width of any quotient graph of a prime node in the modular decomposition tree of G. On the other hand, modular-width cannot be bounded by a function of clique-width, e.g., any path with n vertices has modular-width n, but clique-width three. Every graph with bounded tree-width also has bounded clique-width as it holds that $\mathsf{cw}(G) \leq 3 \cdot 2^{\mathsf{tw}(G)-1}$ for each graph G [CR05]. The reverse implication does not hold since small clique-width does not imply the sparsity of the graph, e.g., each complete graph K_n has clique-width two, but tree-width $n-1$ for each $n \geq 2$. Also every graph of bounded shrub-depth does have bounded clique-width [GHN$^+$12]; but as the class of all cographs has unbounded shrub-depth, the shrub-depth cannot be bounded by any function of the clique-width.

A famous algorithmic metatheorem by Courcelle et al. [CMR00] states that on graphs of bounded clique-width one can solve any problem that can be expressed in MSO_1 logic in linear time. A caveat is that the running time depends non-elementarily on the clique-width of the input graph and the size of the MSO_1 formula, which is even provably unavoidable unless $\mathsf{P} = \mathsf{NP}$ [FG04]. Thus, this result is of less importance for our work in the framework of "FPT in P".

In this chapter, we will give efficient parameterized algorithms for tractable problems with a polynomial dependency on the clique-width cw of the input graph. As our main result, we present an $\mathcal{O}(\mathsf{cw}^2\, n^2)$-time algorithm for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, yielding a truly subcubic algorithm for $\mathsf{cw} \in \mathcal{O}(n^{0.5-\varepsilon})$. This immediately allows to solve the DIAMETER problem, in which one is interested in the longest shortest path in an unweighted graph,

in the same asymptotic time $\mathcal{O}(\mathsf{cw}^2\, n^2)$, even with vertex weights. This nicely complements the lower bound for DIAMETER that rules out algorithms with running time $\mathcal{O}(2^{o(\mathsf{cw})} \cdot n^{2-\varepsilon})$ for any $\varepsilon > 0$ [CDP19]. Further, we present an algorithm for TRIANGLE COUNTING with a time complexity of $\mathcal{O}(\mathsf{cw}^2\, n + \mathsf{cw}\, m)$, which is a slight improvement over the $\mathcal{O}(\mathsf{cw}^2(n+m))$-time algorithm by Coudert et al. [CDP19].

The clique-width of a graph denotes the minimum number k that is needed to construct the graph by a so-called k-expression. We refer to Section 4.1 for all formal definitions needed in this chapter. Trivially, it holds that $\mathsf{cw}(G) \leq n$, i.e., the clique-width is bounded by the number of vertices in an undirected graph. Actually, it holds that $\mathsf{cw}(G) \leq n - k$ as long as $2^k < n - kn$ [GWY16]. Also for directed graphs, one can slightly improve the trivial bound $\mathsf{dcw}(G) \leq n$ to $\mathsf{dcw}(G) \leq n - k$ for $4^k < n - k$, where $\mathsf{dcw}(G)$ denotes the directed clique-width of the graph G. Indeed, we will not use the parameter clique-width directly in this chapter. Instead, we will work with a closely related parameter NLC-width[1] introduced by Wanke [Wan94]. The NLC-width and the clique-width of a graph differ at most by a factor of two. Moreover, the corresponding NLC-width expression can be computed from a clique-width k-expression in linear time.

However, neither the clique-width nor the NLC-width of a graph can be computed in polynomial time, nor can it be approximated by a constant factor in linear time, unless $\mathsf{P} = \mathsf{NP}$ [FRRS09]. Thus, we need to assume that an instance is always given by a k-expression of a graph G, witnessing that the clique-width of G is at most k.

On the positive side, the graphs of clique-width at most three can be recognized in polynomial time [CHL$^+$12] and for some other graph classes of bounded clique-width, one can even compute a corresponding k-expression in linear time, e.g., for graphs of bounded modular-width [CO00] (this includes all cographs), graphs of bounded tree-width [CR05] or bounded split-width [Rao08b], or distance-hereditary graphs [GR00]. Thus, our result implies that on all these graph classes can solve VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS in optimal time $\mathcal{O}(n^2)$ and TRIANGLE COUNTING in optimal linear time $\mathcal{O}(n+m)$, even even if a k-expression is not a part of the input.[2]

Overview. We first define the clique-width and NLC-width of undirected and directed graphs in Section 4.1. In Section 4.2, we present the main result of this chapter, namely the algorithm for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS running in time $\mathcal{O}(\mathsf{cw}^2\, n^2)$ where cw denotes the clique-width of the graph. The algorithm for TRIANGLE COUNTING with a time complexity of $\mathcal{O}(\mathsf{cw}^2\, n + \mathsf{cw}\, m)$ can be found in Section 4.3. We conclude in Section 4.4.

4.1 Definition of Clique-Width and NLC-Width

A *k-labeled* graph is a graph in which each vertex is assigned one out of k labels. Formally, a vertex-labeled graph G is a triple (V, E, lab) where V is the vertex set, E denotes the set of edges, and $lab : V \to [k]$ is a function that assigns a label to each vertex. For a k-labeled graph $G = (V, E, lab)$, by $unlab(G) = (V, E)$ we denote the underlying unlabeled graph. We use this notation for both directed and undirected graphs.

Informally speaking, a graph G has clique-width at most k, if it is the underlying graph of some k-labeled graph that can be constructed by using four operations: (1) Introducing a

[1] The abbreviation NLC stands for *node label controlled*, a term originally defined for graph grammars.

[2] Since the output of VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS is an $n \times n$ distance matrix, every algorithm for this problem does take time $\Omega(n^2)$.

single labeled vertex, (2) redefining one label to another label, (3) taking the disjoint union of two already created k-labeled graphs, and (4) adding all edges between vertices of label i and vertices of label j. We first define clique-width of undirected graphs.

Definition 4.1 (undirected clique-width, [CO00])**.** Let $k \geq 1$. The class CW_k consists of all k-labeled graphs that can be constructed by the following operations:

- The nullary operation $\bullet_i$ that returns a graph consisting of a single vertex of label $i \in [k]$. The resulting graph is in CW_k.
- Let $G = (V, E, lab) \in \mathrm{CW}_k$ be a k-labeled graph, and let $a, b \in [k]$. Then

$$\rho_{a,b}(G) = (V, E, lab') \qquad \text{with } lab'(v) = \begin{cases} lab(v) & \text{if } lab(v) \neq a \\ b & \text{if } lab(v) = a \end{cases}$$

  is in CW_k.
- Let $G = (V_G, E_G, lab_G) \in \mathrm{CW}_k$ and $H = (V_H, E_H, lab_H) \in \mathrm{CW}_k$ be two k-labeled graphs in CW_k with $V_G \cap V_H = \emptyset$. Then the disjoint union, defined by

$$G \oplus H = (V_G \dot\cup V_H, E_G \dot\cup E_H, lab') \qquad \text{with } lab'(v) = \begin{cases} lab_G(v) & \text{if } v \in V_G \\ lab_H(v) & \text{if } v \in V_H \end{cases}$$

  is in CW_k.
- Let $G = (V, E, lab) \in \mathrm{CW}_k$ be a k-labeled graph, and let $a, b \in [k]$ with $a \neq b$. Then

$$\eta_{a,b}(G) = (V, E', lab) \qquad \text{with } E' = E \cup \left\{ \{u, v\} \in \binom{V}{2} \;\middle|\; lab(u) = a, lab(v) = b \right\}$$

  is in CW_k.

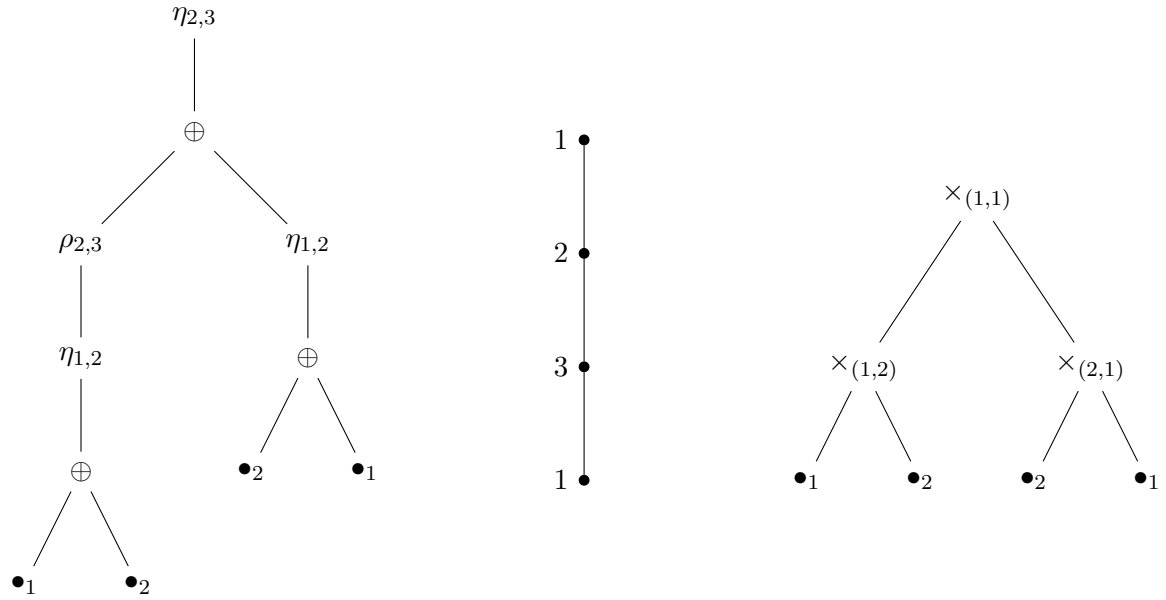
The clique-width of an undirected graph G, denoted by $\mathsf{cw}(G)$, is the smallest k such that there is a labeled graph $G' \in CW_k$ with $unlab(G') = G$. An example for a clique-width 3-expression for the path on four vertices as depicted in Figure 4.1b is

$$\eta_{2,3}((\rho_{2,3}(\eta_{1,2}(\bullet_1 \oplus \bullet_2)) \oplus \eta_{1,2}(\bullet_1 \oplus \bullet_2))).$$

By modifying the operation $\eta_{a,b}$ one can also define *directed* clique-width [CO00]. Instead of inserting every possible edge between vertices labeled by a and b, the binary operation $\alpha_{a,b}$ inserts all possible arcs from vertices labeled by a to vertices labeled by b. The operations $\rho_{a,b}$ and $\oplus$ can be defined for directed graphs analogously to the undirected case, cf. Definition 4.1.

Definition 4.2 (directed clique-width, [CO00])**.** Let $k \geq 1$. The class $d\mathrm{CW}_k$ consists of all k-labeled graphs that can be constructed by the following operations:

- The nullary operation $\bullet_i$ that returns a graph consisting of a single vertex of label $i \in [k]$. The resulting graph is in $d\mathrm{CW}_k$.
- Let $G = (V, E, lab) \in d\mathrm{CW}_k$ be a k-labeled graph, and let $a, b \in [k]$. Then $\rho_{a,b}(G)$ is in $d\mathrm{CW}_k$.



(a) A clique-width 3-expression tree of G.

(b) The resulting graph G.

(c) An NLC-width 2-expression tree of G.

Figure 4.1: An example of a clique-width 3-expression and an NLC-width 3-expression of a path G with four vertices.

- Let $G = (V_G, E_G, lab_G) \in d\text{CW}_k$ and $H = (V_H, E_H, lab_H) \in d\text{CW}_k$ be two k-labeled graphs in $d\text{CW}_k$ with $V_G \cap V_H = \emptyset$. Then the disjoint union $G \oplus H$ is in $d\text{CW}_k$.
- Let $G = (V, E, lab) \in d\text{CW}_k$ be a k-labeled graph, and let $a, b \in [k]$ with $a \neq b$. Then

$$\alpha_{a,b}(G) = (V, E', lab) \qquad \text{with } E' = E \cup \{(u, v) \in V \times V \mid lab(u) = a, lab(v) = b\}$$

  is in $d\text{CW}_k$.

The clique-width of a directed graph G, denoted by $\mathsf{dcw}(G)$, is the smallest k such that there is a labeled graph $G' \in d\text{CW}_k$ with $unlab(G') = G$. Very similar to the parameter clique-width, one can define the parameter NLC-width. The main differences are that the join operation η resp. α and the disjoint union operation $\oplus$ are somewhat combined and consecutive relabel operations are compressed into a single operation. We again first define NLC-width for undirected graphs.

Definition 4.3 (undirected NLC-width, [Wan94])**.** Let $k \geq 1$. The class NLC_k consists of all k-labeled graphs that can be constructed by the following operations:

- The nullary operation $\bullet_i$ that returns a graph consisting of a single vertex of label $i \in [k]$. The resulting graph is in NLC_k.
- Let $G = (V, E, lab) \in \text{NLC}_k$ and let $R\colon [k] \to [k]$. Then

$$\circ_R(G) = (V, E, lab') \qquad \text{with } lab'(v) = R(lab(v))$$

  is in NLC_k.

- Let $G = (V_G, E_G, lab_G) \in \mathrm{NLC}_k$ and $H = (V_H, E_H, lab_H) \in \mathrm{NLC}_k$ be two k-labeled graphs in NLC_k. Let $S \subseteq [k]^2$. Then $G \times_S H = (V_G \cup V_H, E', lab')$ with

$$lab'(v) = \begin{cases} lab_G(v) & \text{if } v \in V_G \\ lab_H(v) & \text{if } v \in V_H \end{cases} \text{ and}$$
$$E' = E_G \cup E_H \cup \left\{ \{u, v\} \in \binom{V_G \cup V_H}{2} \;\middle|\; u \in V_G, v \in V_H, \text{ and } (lab_G(u), lab_H(v)) \in S \right\}$$

  is in NLC_k.

The NLC-width of a graph G, denoted by $\mathsf{nlc}(G)$, is the smallest k such that there is a labeled graph $G' \in NLC_k$ with $unlab(G') = G$. To define *directed* NLC-width, we need to extend the operation $\times_S$. The new operation $\times_{\overrightarrow{S},\overleftarrow{S}}$ uses two relations $\overrightarrow{S}$ and $\overleftarrow{S}$ over $[k]$ to specify the direction of the directed joins between the respective sets of labeled vertices. The operation $\circ_R$ can be defined for directed graphs analogous to the undirected case, cf. Definition 4.3.

Definition 4.4 (directed NLC-width, [Wan94])**.** Let $k \geq 1$. The class $d\mathrm{NLC}_k$ consists of all k-labeled graphs that can be constructed by the following operations:

- The nullary operation $\bullet_i$ that returns a graph consisting of a single vertex of label $i \in [k]$. The resulting graph is in $d\mathrm{NLC}_k$.
- Let $G = (V, E, lab) \in \mathrm{NLC}_k$ and let $R\colon [k] \to [k]$. Then $\circ_R(G)$ is in NLC_k.
- Let $G = (V_G, E_G, lab_G) \in \mathrm{NLC}_k$ and $H = (V_H, E_H, lab_H) \in \mathrm{NLC}_k$ be two k-labeled graphs in NLC_k. Let $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$. Then $G \times_{\overrightarrow{S},\overleftarrow{S}} H = (V_G \cup V_H, E', lab')$ with

$$lab'(v) = \begin{cases} lab_G(v) & \text{if } v \in V_G \\ lab_H(v) & \text{if } v \in V_H \end{cases} \text{ and}$$
$$E' = E_G \cup E_H \cup \left\{ (u, v) \in V_G \times V_H \;\middle|\; (lab_G(u), lab_H(v)) \in \overrightarrow{S} \right\} \cup \left\{ (v, u) \in V_H \times V_G \;\middle|\; (lab_G(u), lab_H(v)) \in \overleftarrow{S} \right\}$$

  is in $d\mathrm{NLC}_k$.

The directed NLC-width of a graph G, denoted by $\mathsf{dnlc}(G)$, is the smallest k such that there is a labeled graph $G' \in d\mathrm{NLC}_k$ with $unlab(G') = G$.

An expression consisting of the operations defined in Definition 4.1 or Definition 4.2 is called a (directed) clique-width k-expression. An expression consisting of the operations defined in Definition 4.3 or Definition 4.4 is called a (directed) NLC-width k-expression. For any k-expression σ, we denote with $val(\sigma)$ the resulting labeled graph and by $tree(\sigma)$ the so-called k-expression tree of σ, i.e., a rooted tree T in which each leaf is marked with $\bullet_i$ for some $i \in [k]$ and each internal node is marked with an operation defined in Definitions 4.1 to 4.4 that is applied to the children resp. child of the node, see also Figure 4.1 for an example. In particular, in an expression tree T of an NLC-width k-expression, each leaf node of T is marked with $\bullet_i$ for some $i \in [k]$ and each internal node is either marked with $\circ_R$ for some $R\colon [k] \to [k]$ or with $\times_S$ for some $S \subseteq [k]^2$ (resp. $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \in [k]^2$ for directed NLC-width), according to the operations defined in Definition 4.3 (resp. Definition 4.4). Thus, a k-expression tree T of a graph G with n vertices does have exactly n leaves, one for each vertex in G. Since each

node marked with $\times$ resp. $\times_{\overrightarrow{S},\overleftarrow{S}}$ has exactly two children, there are $n-1$ nodes of this kind in any expression. Further, one can assume that there are no two consecutive relabel operations, i.e., a child of a node marked with $\circ_R$ in the expression tree $tree(\sigma)$ is either a leaf or a node marked with $\times_S$ resp. $\times_{\overrightarrow{S},\overleftarrow{S}}$. Thus, we may assume that the length of an NLC-expression is in $\mathcal{O}(n)$. Note that the length of a clique-width expression can be in $\Omega(n+m)$.

For a graph G and a positive integer k, the problem to decide whether G has (directed) clique-width at most k, as well as the problem to decide whether G has (directed) NLC-width at most k, is NP-hard. Johansson [Joh98] showed that the NLC-width of a graph with n vertices is at most $\lceil n/2 \rceil$: Partition the vertex set arbitrary into two sets of size at most $\lceil n/2 \rceil$, create the corresponding induced subgraphs by assigning each vertex a unique label, and combine the two subgraphs via an operation $\times_S$. The same holds for directed NLC-width.

In this chapter, we will solely focus on NLC-width k-expressions to design efficient parameterized algorithms. The following lemma shows that there is no drawback of doing so, since the NLC-width of a graph is bounded by the clique-width of the graph.

Lemma 4.5 ([Joh98])**.** *For any graph G it holds that* $\mathsf{nlc}(G) \leq \mathsf{cw}(G) \leq 2 \cdot \mathsf{nlc}(G)$.

Moreover, one can transform a clique-width k-expression into an equivalent NLC-width k-expression in linear time $\mathcal{O}(n+m)$ as described for example in [Joh98]. The same relations hold for the directed versions.

Lemma 4.6 ([GWY16])**.** *For any graph G it holds that* $\mathsf{dnlc}(G) \leq \mathsf{dcw}(G) \leq 2 \cdot \mathsf{dnlc}(G)$.

Recall that for a directed graph G, with $u(G)$ we denote the underlying undirected graph, and for an undirected graph $G = (V, E)$, with $\overleftrightarrow{G} = (V, A)$ we denote the corresponding bidirected graph (cf. Section 2.2 for the formal definitions). Then, for any directed graph G, it holds that $\mathsf{nlc}(u(G)) \leq \mathsf{dnlc}(G)$ and $\mathsf{cw}(u(G)) \leq \mathsf{dcw}(G)$. This can be seen by replacing each occurrence of $\times_{\overrightarrow{S},\overleftarrow{S}}$ in a directed NLC-width k-expression by $\times_{\overleftarrow{S \cup S}}$ resp. by replacing each occurrence of $\alpha_{a,b}$ in a directed clique-width k-expression by $\eta_{a,b}$ for $\overleftarrow{S}, \overrightarrow{S} \subseteq [k]^2$ and $a, b \in [k]$. By a similar observation, it also holds that $\mathsf{nlc}(G) = \mathsf{dnlc}(\overleftrightarrow{G})$ and $\mathsf{cw}(G) = \mathsf{dcw}(\overleftrightarrow{G})$.

4.2 All-Pairs Shortest Path Parameterized by Clique-Width

In this section, we present an algorithm parameterized by the clique-width of the input graph for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS. We consider the more general directed clique-width on a vertex-weighted graph with possibly negative vertex weight. We will either compute the shortest distance for each pair of vertices, or conclude that the graph contains a cycle of negative weight.

VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS

Input: A directed graph $G = (V, E)$, vertex weights $\omega \colon V \to \mathbb{R}$.
Output: The pairwise distances $dist_G(u, v)$ for all $u, v \in V$.

Since the output size of any VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS instance is in $\Omega(n^2)$, one cannot solve VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS in time $f(\mathsf{cw}) \cdot n^{2-\varepsilon}$ for any function f and $\varepsilon > 0$. We will show how to solve VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS in time $\mathcal{O}(\mathsf{cw}^2\, n^2)$.

Theorem 4.7. *For every directed graph $G = (V, E)$, given together with a directed clique-width k-expression and vertex weights $\omega\colon V \to \mathbb{R}$, we either can conclude that G contains a cycle of negative weight, or we can solve* VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS *in time $\mathcal{O}(k^2n^2)$.*

For a directed input graph $G = (V, E)$, given by a directed clique-width k-expression for some $k \geq 2$, in the first step we transform the directed clique-width k-expression to a directed NLC-width k-expression in linear time as described for example in [GWY16]. For the rest of this section, under a k-expression we always refer to a directed NLC-width k-expression instead of a directed clique-width k-expression. We interpret the k-expression σ as a k-expression tree $T = tree(\sigma)$, in which each node $x \in V(T)$ is marked with an operation of the k-expression that is applied to the children of x. Accordingly, T has exactly n leaves, each marked with an operation $\bullet_i$ for $i \in [k]$, and exactly $n-1$ nodes marked with an operation $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$. For the ease of presentation, we assume that there is always exactly one node marked with an operation $\circ_R$ for some $R\colon [k] \to [k]$ in between any two nodes marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ (using $R(i) = i$ when no actual relabeling is necessary). For a node $x \in V(T)$, we denote by G^x the labeled graph defined by the k-expression represented by the subtree of T rooted at x, and we define by $L_i^x = \{v \in V(G^x) \mid lab(v) = i\}$ the set of vertices in G^x with label $i \in [k]$. Note that $unlab(G^x)$ is an induced subgraph of G for any $x \in V(T)$. For the root node $r \in V(T)$ it holds that $G^r = G$. For a node $x \in V(T)$, we will use the shortcut $dist^x(u, v) := dist_{G^x}(u, v)$ to denote the distance between two vertices u and v in G^x.

The algorithm consists of three phases. In the first phase, we traverse the k-expression tree T in a bottom-up manner: For each node $x \in V(T)$, we partition the vertex set of G^x into the sets of same-labeled vertices and compute the shortest distance for every single vertex to (the closest vertex in) each set of same-labeled vertices. Additionally, we compute the distance between each pair of the same-labeled vertex sets, i.e., the shortest distance of any two vertices of the respective sets. Note, however, that in the first phase we only consider for each node $x \in V(T)$ the distances in the graph G^x. In the second phase, we perform a top-down traversal of the k-expression tree T and consider the whole graph G. Once we have computed the necessary values in phases one and two, we traverse T one last time and finally compute the shortest path distances between all pairs of vertices.

First Phase. We traverse T in a bottom-up manner and compute for each node $x \in V(T)$ and for all pairs $(i, j) \in [k]^2$ of labels the shortest distance in G^x between some vertex in L_i^x and some vertex in L_j^x. Additionally, we compute for every vertex $v \in V(G^x)$ and every label $i \in [k]$ the shortest distance in G^x from v to some vertex in L_i^x. To be precise, for a node $x \in V(T)$ we compute the following values; recall that for readability we shortcut $dist^x$ for $dist_{G^x}$:

$$c_{i,j}^x = dist^x(L_i^x, L_j^x) \qquad \text{for } i, j \in [k]$$
$$a_{v,i}^x = dist^x(v, L_i^x) \qquad \text{for } v \in V(G^x), i \in [k]$$

For nodes $x \in V(T)$ that are marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$ we need to compute some auxiliary values. Let y and z be the two children of x in T. This means that G^x consists of the disjoint union of G^y and G^z together with a directed full join between vertices from L_i^y to L_j^z for each $(i, j) \in \overrightarrow{S}$ and a directed full join between vertices from L_i^z to L_j^y for each $(i, j) \in \overleftarrow{S}$. Thus, one can partition the vertex set of G^x into the $2k$ sets $\{L_1^y, \ldots, L_k^y, L_1^z, \ldots, L_k^z\} =: \mathcal{L}^x$. For each pair $(A, B) \in \mathcal{L}^x \times \mathcal{L}^x$ of vertex sets, we compute the shortest distance between some vertex in A to some vertex of B. In addition, we compute the shortest distance between A and

B with the constraint that either the first edge, the last edge, or the first and the last edge of the shortest path is an edge of a newly inserted full join defined by $\overleftarrow{S}$ or $\overrightarrow{S}$. This achieves the effect that we additionally compute the shortest distance from (1) *all* vertices of A to *some* vertex of B, (2) from *some* vertex of A to *all* vertices of B, and (3) from *all* vertices of A to *all* vertices of B. Doing this, one can for example combine a path that ends at *some* vertex of A with a path that can start at *any* vertex of A. In the following, we will describe how to compute the required values for each of the three different types of nodes in the k-expression tree T.

For the base case, let $x \in V(T)$ be a leaf of the k-expression tree T. Thus, the node x is marked with $\bullet_\ell$ for some $\ell \in [k]$, i.e., G^x consists of a single vertex v with label ℓ. In this case the following holds:

$$c^x_{i,j} = \begin{cases} \omega(v) & \text{if } i = j = \ell \\ \infty & \text{otherwise} \end{cases} \qquad \text{for } i,j \in [k]$$

$$a^x_{v,i} = \begin{cases} \omega(v) & \text{if } i = \ell \\ \infty & \text{otherwise} \end{cases} \qquad \text{for } v \in V(G^x), i \in [k]$$

Now, let $x \in V(T)$ be an internal node of the k-expression tree T marked with $\circ_R$ for some $R\colon [k] \to [k]$. Let $y \in V(T)$ be the unique child of x in T. Since we traverse T in a bottom-up manner, we have already computed the values $a^y_{v,i}$ for all $v \in V(G^y)$ and $i \in [k]$ and the values $c^x_{i,j}$ for all $i,j \in [k]$. Note that $unlab(G^x) = unlab(G^y)$, which, in particular, implies that distances between vertices are identical in both graphs (this is not necessarily true for distances between label sets though as these sets may be different).

Lemma 4.8. *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\circ_R$ for some $R\colon [k] \to [k]$ and let $y \in V(T)$ be the child of x in T. Then $c^x_{i,j} = \min_{i' \in R^{-1}(i), j' \in R^{-1}(j)} c^y_{i',j'}$ for all $i,j \in [k]$.*

Proof. The vertex sets of G^x resp. G^y can be partitioned into the label sets $\{L^x_1, \ldots, L^x_k\}$ resp. $\{L^y_1, \ldots, L^y_k\}$. Note that some label sets L^x_i resp. L^y_i might be empty for some $i \in [k]$ and it holds that $L^y_i \subseteq L^x_{R(i)}$ and $L^x_i = \bigcup_{j \in R^{-1}(i)} L^y_j$ for all $i \in [k]$. It follows that

$$\begin{aligned} c^x_{i,j} &= dist^x\left(L^x_i, L^x_j\right) \\ &= dist^y\left(\bigcup_{i' \in R^{-1}(i)} L^y_{i'}, \bigcup_{j' \in R^{-1}(j)} L^y_{j'}\right) \\ &= \min_{\substack{i' \in R^{-1}(i),\\ j' \in R^{-1}(j)}} dist^y\left(L^y_{i'}, L^y_{j'}\right) \\ &= \min_{\substack{i' \in R^{-1}(i),\\ j' \in R^{-1}(j)}} c^y_{i',j'}. \end{aligned}$$

Here, it is crucial that the distances between vertices are the same in G^x and G^y, as noted above. □

Note that the computation of all $c^x_{i,j}$ can be realized in time $\mathcal{O}(k^2)$ by updating for every $c^y_{i,j}$ the corresponding value $c^x_{R(i),R(j)}$. The values $a^x_{v,i}$ can be computed similarly from the values at the child node.

Lemma 4.9. *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\circ_R$ for some $R\colon [k] \to [k]$ and let $y \in V(T)$ be the child of x in T. Then $a^x_{v,i} = \min_{j \in R^{-1}(i)} a^y_{v,j}$ for all $v \in V(G^x)$ and $i \in [k]$.*

Proof. Let $\{L^x_1, \ldots, L^x_k\}$ resp. $\{L^y_1, \ldots, L^y_k\}$ be the partition in G^x resp. G^y of the vertex set into sets of same-labeled vertices. Again, it holds that $L^y_i \subseteq L^x_{R(i)}$ and $L^x_i = \bigcup_{j \in R^{-1}(i)} L^y_j$ for all $i \in [k]$. Thus, the following holds:

$$\begin{aligned} a^x_{v,i} &= dist^x\left(v, L^x_i\right) \\ &= dist^y\left(v, \bigcup_{j \in R^{-1}(i)} L^y_j\right) \\ &= \min_{j \in R^{-1}(i)} dist^y\left(v, L^y_j\right) \\ &= \min_{j \in R^{-1}(i)} a^y_{v,j} \end{aligned}$$

□

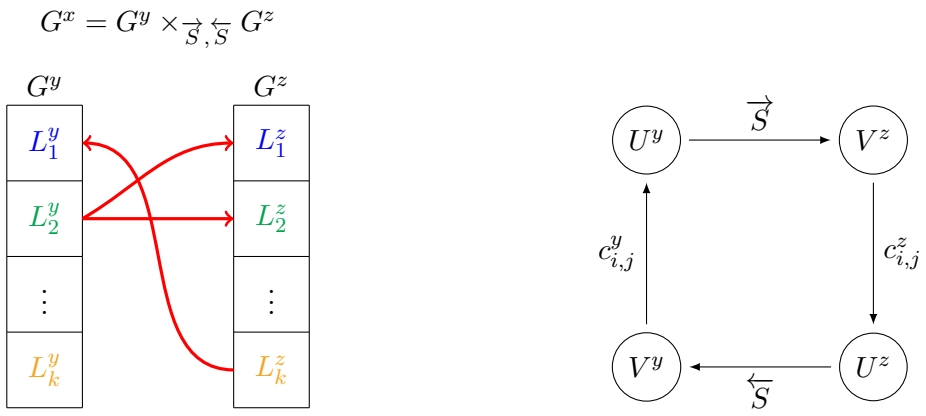
The running time for computing the values $a^x_{v,i}$ is $\mathcal{O}(nk)$ since we need to consider each value $a^y_{v,i}$ for any $v \in V(G^y)$ and $i \in [k]$ exactly once.

Finally, let $x \in V(T)$ be an internal node of the k-expression tree T marked with $\times_{\overrightarrow{S}, \overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$. Denote by $y \in V(T)$ and $z \in V(T)$ the two children of x in T, meaning that G^x combines the two labeled graphs G^y and G^z by introducing for each $(i,j) \in \overrightarrow{S}$ a directed full join from all vertices in L^y_i to all vertices in L^z_j and for each $(i,j) \in \overleftarrow{S}$ a directed full join from all vertices in L^z_i to all vertices in L^y_j. Thus, $V(G^x) = V(G^y) \dot\cup V(G^z)$ and one can partition the vertices of G^x into the $2k$ vertex sets $\{L^y_1, \ldots, L^y_k, L^z_1, \ldots, L^z_k\}$ and $L^x_i = L^y_i \dot\cup L^z_i$. See Figure 4.2 as an illustration. To compute the desired distances between the label sets $\{L^y_1, \ldots, L^y_k, L^z_1, \ldots, L^z_k\}$, we construct an edge-weighted directed graph H^x that represents all distances between the label sets in a graph with only $4k$ vertices.

For each label set L^a_i of G^x with $i \in [k]$ and $a \in \{y, z\}$ we create two vertices v^a_i and u^a_i. Let $V^a = \{v^a_i \mid i \in [k]\}$ resp. $U^a = \{u^a_i \mid i \in [k]\}$ for $a \in \{y, z\}$. We add a directed full join from V^y to U^y resp. from V^z to U^z with edge weights equal to the length of a shortest path between the two corresponding label sets. Finally, we connect vertices in U^y with vertices in V^z, if and only if the corresponding pair is contained in $\overrightarrow{S}$, resp. we connect vertices in U^z with vertices in V^y, if and only if the corresponding pair is contained in $\overleftarrow{S}$, i.e., if there is a directed full join in G^x between the two corresponding label sets. See also Figure 4.2 for an illustration. Formally, we define the directed, edge-weighted graph H^x as follows.

Definition 4.10. Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\times_{\overrightarrow{S}, \overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$ and let $y \in V(T)$ and $z \in V(T)$ be the children of x. We define H^x as a directed, edge-weighted graph on $4k$ vertices created as follows:

- For each label set $L^a_i \in \{L^y_1, \ldots, L^y_k, L^z_1, \ldots, L^z_k\}$ we create two vertices v^a_i and u^a_i for $i \in [k]$ and $a \in \{y, z\}$,
- add edge (v^y_i, u^y_j) with cost $c^y_{i,j}$ for every $i, j \in [k]$,
- add edge (v^z_i, u^z_j) with cost $c^z_{i,j}$ for every $i, j \in [k]$,
- add edge (u^y_i, v^z_j) with cost zero for every $(i,j) \in \overrightarrow{S}$,



(a) Example of $G^x = G^y \times_{\overrightarrow{S},\overleftarrow{S}} G^z$ with $\overrightarrow{S} = \{(2,1),(2,2)\}$ and $\overleftarrow{S} = \{(k,1)\}$.

(b) The auxiliary graph H^x as definied in Definition 4.10.

Figure 4.2: (a): Example of $G^x = G^y \times_{\overrightarrow{S},\overleftarrow{S}} G^z$. (b): Construction of the auxiliary graph H^x. Each large node consists of k isolated vertices corresponding to one of the k label sets in G^y resp. G^z. Between V^y and U^y there is a full join, each edge between the corresponding vertex of L_i^y and L_j^y is weighted by $c^y_{i,j}$, analogously for V^z and U^z. Vertices in U^y (resp. U^z) are only adjacent (with edge cost zero) to those vertices in V^z (resp. V^y) for which the corresponding label sets are connected via $\overrightarrow{S}$ (resp. $\overleftarrow{S}$).

- add edge (u_i^z, v_j^y) with cost zero for every $(i,j) \in \overleftarrow{S}$.

Note that some edges may have cost ∞, namely if there does not exist a path of the requested type. Also note that $c^y_{i,i}$ corresponds to the minimum weight of any vertex in L_i^y for $i \in [k]$. Next, we will see that H^x exhibits all the desired distances from G^x in a compact way.

Theorem 4.11. *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$ and let $y \in V(T)$ and $z \in V(T)$ be the children of x in T. Let H^x be the graph as defined in Definition 4.10. Then the following holds:*

(1) $dist_{H^x}(v_i^a, u_j^b) = dist^x(L_i^a, L_j^b)$ for all $a, b \in \{y, z\}$ and $i, j \in [k]$.

(2) $dist_{H^x}(u_i^a, u_j^b) = \min_{P \in \mathcal{P}} \omega(P) - \min_{v \in L_i^a} \omega(v)$ where $\mathcal{P}$ is the set of all paths in G^x starting in L_i^a, ending in L_j^b, and having the second vertex in $V(G^x) \setminus V(G^a)$.

(3) $dist_{H^x}(v_i^a, v_j^b) = \min_{P \in \mathcal{P}} \omega(P) - \min_{v \in L_j^b} \omega(v)$ where $\mathcal{P}$ is the set of all paths in G^x starting in L_i^a, ending in L_j^b, and having the penultimate vertex in $V(G^x) \setminus V(G^b)$.

(4) $dist_{H^x}(u_i^a, v_j^b) = \min_{P \in \mathcal{P}} \omega(P) - \min_{v \in L_i^a} \omega(v) - \min_{v \in L_j^b} \omega(v)$ where $\mathcal{P}$ is the set of all paths in G^x starting in L_i^a, ending in L_j^b, and having the second vertex in $V(G^x) \setminus V(G^a)$ and the penultimate vertex in $V(G^x) \setminus V(G^b)$.

We prove Theorem 4.11 in two steps. We first prove that every path in H^x corresponds to some path in G^x with the correct cost. Later, we prove that also each optimal path between two label sets in G^x corresponds to some shortest path in H^x. We start with statement (1) of Theorem 4.11.

Lemma 4.12. *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$ and let $y \in V(T)$ and $z \in V(T)$ be the children of x in T. Let H^x be the graph as defined in Definition 4.10 and let P^* be an arbitrary v_i^y-u_j^z path in H^x for some $i, j \in [k]$. Then, there exists an L_i^y-L_j^z path P in G^x with $\omega(P) = \omega_{H^x}(P^*)$.*

Proof. Due to the circular structure of H^x, each v_i^y-u_j^z path in H^x will repeat the sequence $(v_p^y, u_q^y, v_r^z, u_s^z)$ for some $p, q, r, s \in [k]$ until reaching u_j^z at the end of a sequence. Thus, each v_i^y-u_j^z path in H^x consists of 4ℓ vertices for $\ell \in \mathbb{N}^+$ and can be written as

$$P^* = \left(v_i^y = v_{p_1}^y, u_{q_1}^y, v_{r_1}^z, u_{s_1}^z, v_{p_2}^y, u_{q_2}^y, v_{r_2}^z, u_{s_2}^z, \ldots, v_{p_\ell}^y, u_{q_\ell}^y, v_{r_\ell}^z, u_{s_\ell}^z = u_j^z\right).$$

One can construct a path P in G^x from P^* as follows: For each edge $(v_{p_i}^y, u_{q_i}^y)$ in P^* of cost c_{p_i,q_i}^y pick a shortest path in G^y of total cost c_{p_i,q_i}^y and for each edge $(v_{r_i}^z, u_{s_i}^z)$ in P^* of cost c_{r_i,s_i}^y pick a shortest path in G^z of total cost c_{r_i,s_i}^z for each $i \in [\ell]$. Those paths always exist since c_{p_i,q_i}^y resp. c_{r_i,s_i}^y are defined as the cost of a shortest $L_{p_i}^y$-$L_{q_i}^y$ path in G^y resp. as the cost of a shortest $L_{r_i}^z$-$L_{s_i}^z$ path in G^z. Since each edge (u_{q_i}, v_{r_i}) exists if and only if there is a directed full join between the sets $L_{q_i}^y$ and $L_{r_i}^z$, one can connect the last vertex of the path corresponding to the previous edge in P^* (that ends in some vertex in $L_{q_i}^y$) to the first vertex of the path corresponding to the following edge in P^* (that starts at some vertex in $L_{r_i}^z$). In the same manner, one can argue that due to the edge $(u_{s_i}^z, v_{p_{i+1}}^y)$ one can connect the last vertex of the path corresponding to the edge $(v_{r_i}^z, u_{s_i}^z)$ with the first vertex of the path corresponding to the edge $(v_{p_{i+1}}^y, u_{q_{i+1}}^y)$. In both cases, the costs of the two endpoints of the connecting edge are already counted in the respective $c_{\cdot,\cdot}^{\cdot}$ value. Thus, each v_i^y-u_j^z path in H^x corresponds to an L_i^y-L_j^z path in G^x of the same cost. □

Next, we generalize this argumentation to the following corollary.

Corollary 4.13. *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$ and let $y \in V(T)$ and $z \in V(T)$ be the children of x in T. Let H^x be the graph as defined in Definition 4.10. Then for every $i, j \in [k]$ and $a, b \in \{y, z\}$ the following holds:*

(1) For any v_i^a-u_j^b path P^* in H^x there exists an L_i^a-L_j^b path P in G^x with $\omega_{H^x}(P^*) = \omega(P)$.

(2) For any u_i^a-u_j^b path P^* in H^x there exists an L_i^a-L_j^b path $P = (p_1, p_2, \ldots, p_\ell)$ in G^x with the property that $p_2 \in V(G^x) \setminus V(G^a)$ and $\omega_{H^x}(P^*) = \omega(P) - \omega(p_1)$.

(3) For any v_i^a-v_j^b path P^* in H^x there exists an L_i^a-L_j^b path $P = (p_1, \ldots, p_{\ell-1}, p_\ell)$ in G^x with the property that $p_{\ell-1} \in V(G^x) \setminus V(G^b)$ and $\omega_{H^x}(P^*) = \omega(P) - \omega(p_\ell)$.

(4) For any u_i^a-v_j^b path P^* in H^x there exists an L_i^a-L_j^b path $P = (p_1, p_2, \ldots, p_{\ell-1}, p_\ell)$ in G^x with the property that $p_2 \in V(G^x) \setminus V(G^a)$, $p_{\ell-1} \in V(G^x) \setminus V(G^b)$, and $\omega_{H^x}(P^*) = \omega(P) - \omega(p_1) - \omega(p_\ell)$.

Proof. With the same argumentation as in the proof of Lemma 4.12, one can prove that also every v_i^y-u_j^y path in H^x corresponds to an L_i^y-L_j^y path in G^x with the same cost. The two cases with $a = z$ and $b \in \{y, z\}$ now follow by swapping the roles of y and z. This proves (1). Similarly, one can argue that any path P^* in H^x that starts at a vertex u_i^y (resp. ends at a vertex v_i^z) for $i \in [k]$, corresponds to a path $P = (p_1, p_2, \ldots, p_{\ell-1}, p_\ell)$ in G^x that starts

in L_i^y (resp. ends in L_j^z) with the additional property (due to the directed edges in H^x) that $p_2 \in V(G^x) \setminus V(G^y)$ (resp. $p_{\ell-1} \in V(G^x) \setminus V(G^z)$). The cost of P^* in H^x is exactly the cost of P in G^x, minus the cost of the first (resp. last) vertex of P. In this regard, recall that all vertex costs in G^x are represented by the edge weights in H^x. The remaining cases of Corollary 4.13 with $a = z$ or $b = y$ again can be shown analogously. □

For any path in H^x that starts at some vertex u_i^a (resp. ends at some vertex v_j^b) one can find a corresponding path P in G^x with the property that the second vertex (resp. the penultimate vertex) is connected to *all* vertices of L_i^a (resp. L_j^b). Thus, one can extend any path that ends at *some* vertex in L_i^a by such a path (resp. one can prepend any path that starts in L_j^b by such a path). Hence, the cost of the first vertex (resp. last vertex) is neglected if the path starts in some vertex u_i^a or ends at some vertex v_j^b for $a, b \in \{y, z\}$. In general, every path that one can find in H^x corresponds to a path in G^x of essentially the same cost, possibly without the first or last vertex (which can be chosen as the minimum of the label set). This proves "$\leq$" in the equations of Theorem 4.11.

For the other direction, we will show that every optimal shortest path between two label sets in G^x is represented by a path in H^x.

Lemma 4.14. *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$ and let $y \in V(T)$ and $z \in V(T)$ be the children of x in T. Let H^x be the graph as defined in Definition 4.10. Let P be a shortest L_i^y-L_j^z path in G^x for some $i, j \in [k]$. Then there exist a v_i^y-u_j^z path P^* in H^x with $\omega_{H^x}(P^*) = \omega(P)$.*

Proof. Let P be a shortest L_i^y-L_j^z path in G^x for fixed $i, j \in [k]$ of length $\omega(P) = dist^x(L_i^y, L_j^z)$. Since $V(G^x)$ can be subdivided into the two vertex sets $V(G^y)$ and $V(G^z)$, we can split the path P into maximal subpaths consisting of vertices completely in $V(G^y)$ resp. completely in $V(G^z)$. Formally, let $P = (P_1, \ldots, P_d)$ with $V(P_{2q+1}) \subseteq V(G^y)$ and $V(P_{2q}) \subseteq V(G^z)$ for $0 \leq q \leq d/2$. Let $v_i, u_i \in V(G)$ such that $P_i = P_{[v_i,u_i]}$ for all $i \in [d]$. Thus,

$$P = (\underbrace{v_1, \ldots, u_1}_{P_1 \in V(G^y)}, \underbrace{v_2, \ldots, u_2}_{P_2 \in V(G^z)}, \ldots, \underbrace{v_{d-1}, \ldots, u_{d-1}}_{P_{d-1} \in V(G^y)}, \underbrace{v_d, \ldots, u_d}_{P_d \in V(G^z)}).$$

Note that a path P_ℓ could consist of a single vertex $v_\ell = u_\ell$ for $\ell \in [d]$. Define $E_{new}^x = E(G^x) \setminus (E(G^y) \cup E(G^z))$ as the set of newly created edges in G^x. I.e., E_{new}^x is the union of all directed full joins between the vertex sets L_i^y and L_j^z for each $(i, j) \in \overrightarrow{S}$ and L_i^z and L_j^y for each $(i, j) \in \overleftarrow{S}$. For the path P it holds by construction that $\{u_\ell, v_{\ell+1}\} \in E_{new}^x$ for $\ell \in [d-1]$ and, since P is a shortest path, each P_ℓ is a shortest v_ℓ-u_ℓ path in G^y (for ℓ odd) resp. in G^z (for ℓ even) for all $\ell \in [d]$.

To complete the proof, we will show that for each $\ell \in [d]$ it holds that $\omega(P_\ell) = c^y_{lab(v_\ell),lab(u_\ell)}$ for ℓ even and $\omega(P_\ell) = c^z_{lab(v_\ell),lab(u_\ell)}$ for ℓ odd, and thus that each path P_ℓ corresponds to an arc $(v^y_{lab(v_\ell)}, u^y_{lab(u_\ell)})$ for ℓ even and to an arc $(v^z_{lab(v_\ell)}, u^z_{lab(u_\ell)})$ for ℓ odd in H^x: Let w.l.o.g. ℓ be odd, hence $V(P_\ell) \subseteq V(G^y)$. Since P_ℓ is an $L^y_{lab(v_\ell)}$-$L^y_{lab(u_\ell)}$ path, it holds that $\omega(P_\ell) \geq c^y_{lab(v_\ell),lab(u_\ell)}$. Assume for contradiction that $\omega(P_\ell) > c^y_{lab(v_\ell),lab(u_\ell)}$ and let Q be an $L^y_{lab(v_\ell)}$-$L^y_{lab(u_\ell)}$ path in G^y with $\omega(Q) = c^y_{lab(v_\ell),lab(u_\ell)}$. Replace P_ℓ by Q in P to get $P' = (P_1, \ldots, P_{\ell-1}, Q, P_{\ell+1}, \ldots, P_d)$. Since $u_{\ell-1}$ is connected to all vertices in $L^y_{lab(v_\ell)}$, and $v_{\ell+1}$ is connected to all vertices in $L^y_{lab(u_\ell)}$, P' is an L_i^y-L_j^z path and since $\omega(P_\ell) > \omega(Q)$ the total cost of P' is smaller than the total cost of P, which is a contradiction. □

Again, one can generalize the argumentation of Lemma 4.14 to the following corollary.

Corollary 4.15. *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$ and let $y \in V(T)$ and $z \in V(T)$ be the children of x in T. Let H^x be the graph as defined in Definition 4.10. Then for every $i, j \in [k]$ and $a, b \in \{y, z\}$ the following holds:*

(1) For every shortest L_i^a-L_j^b path P in G^x, there exists a v_i^a-u_j^b path P^* in H^x of cost $\omega_{H^x}(P^*) = \omega(P)$.

(2) For every shortest L_i^a-L_j^b path P in G^x with the property that the second vertex is in $V(G^x) \setminus V(G^a)$, there exists a u_i^a-u_j^b path P^* in H^x of cost $\omega_{H^x}(P^*) = \omega(P) - \min_{v \in L_i^a} \omega(v)$.

(3) For every shortest L_i^a-L_j^b path P in G^x with the property that the penultimate vertex is in $V(G^x) \setminus V(G^b)$, there exists a v_i^a-v_j^b path P^* in H^x of cost $\omega_{H^x}(P^*) = \omega(P) - \min_{v \in L_j^b} \omega(v)$.

(4) For every shortest L_i^a-L_j^b path P in G^x with the property that the second vertex is in $V(G^x) \setminus V(G^a)$ and the penultimate vertex is in $V(G^x) \setminus V(G^b)$, there exists a u_i^a-v_j^b path in H^x of cost $\omega_{H^x}(P^*) = \omega(P) - \min_{v \in L_i^a} \omega(v) - \min_{v \in L_j^b} \omega(v)$.

Proof. With the same argumentation as done in Lemma 4.14 one can show that also every shortest L_i^y-L_j^y path in G^x corresponds to a shortest v_i^y-u_j^y path in H^x. The cases with $a = z$ follow analogously by renaming y and z. This proves (1). For the remaining cases we observe that for $i, j \in [k]$ and $a, b \in \{y, z\}$, every shortest L_i^a-L_j^b path P in G^x with the property that the second vertex is in $V(G^x) \setminus V(G^a)$ (resp. the penultimate vertex in $V(G^x) \setminus V(G^b)$), the first (resp. last) vertex of P is a minimum cost vertex in L_i^a (resp. in L_j^b) and that it is not covered in the cost of the corresponding path in H^x. □

Corollary 4.15 shows that every *shortest* L_i^a-L_j^b path in G^x is represented in H^x for $i, j \in [k]$ and $a, b \in \{y, z\}$. Together with Corollary 4.13, this proves Theorem 4.11.

After the construction of the auxiliary graph H^x as defined in Definition 4.10, we solve (edge-weighted) ALL-PAIRS SHORTEST PATHS on H^x and store the shortest distances for all pairs of vertices in H^x. Note that since we traverse the expression tree T from bottom to top, we can assume that G^y and G^z do not admit a cycle of negative length. Hence, a negative path in G^x would need to use vertices in G^y *and* G^z; and it follows from Theorem 4.11 that G^x contains a cycle of negative weight if and only if H^x contains a cycle of negative length. Since G^x is an induced subgraph of G, we either can conclude that G contains a cycle of negative weight, or we can compute all pairwise distances in H^x. With those values one can now compute the values $c_{i,j}^x$ and $a_{v,i}^x$ for $i, j \in [k]$ and $v \in V(G^x)$. Note that some of the values are only required in the second phase of the algorithm.

Lemma 4.16. *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$ and let $y \in V(T)$ and $z \in V(T)$ be the children of x in T. Let H^x be the graph as defined in Definition 4.10. Then, for all $i, j \in [k]$ it holds that*

$$c_{i,j}^x = \min \left\{ dist_{H^x}(v_i^y, u_j^y), dist_{H^x}(v_i^y, u_j^z), dist_{H^x}(v_i^z, u_j^y), dist_{H^x}(v_i^z, u_j^z) \right\}$$

Proof. Since $V(G^x)$ is the disjoint union of $V(G^y)$ and $V(G^z)$ it holds that $L_i^x = L_i^y \dot{\cup} L_i^z$ for any $i \in [k]$. Thus, the shortest path between the vertex sets L_i^x and L_j^x for $i, j \in [k]$ in G^x starts

in either L_i^y or L_i^z and ends in either L_j^y or L_j^z and the following holds:

$$\begin{aligned} c_{i,j}^x &= dist^x(L_i^x, L_j^x) \\ &= dist^x(L_i^y \dot\cup L_i^z, L_j^y \dot\cup L_j^z) \\ &= \min\left\{dist^x(L_i^y, L_j^y), dist^x(L_i^y, L_j^z), dist^x(L_i^z, L_j^y), dist^x(L_i^z, L_j^z)\right\} \\ &= \min\left\{dist_{H^x}(v_i^y, u_j^y), dist_{H^x}(v_i^y, u_j^z), dist_{H^x}(v_i^z, u_j^y), dist_{H^x}(v_i^z, u_j^z)\right\} \end{aligned}$$

The last equation follows from Theorem 4.11. □

Lemma 4.17. *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$ and let $y \in V(T)$ and $z \in V(T)$ be the children of x in T. Let H^x be the graph as defined in Definition 4.10. Then, for any $v \in V(G^y)$ and $i \in [k]$ it holds that*

$$a_{v,i}^x = \min_{\substack{j\in[k],\\ a\in\{y,z\}}} \left\{a_{v,j}^y + dist_{H^x}(u_j^y, u_i^a)\right\}$$

and for any $v \in V(G^z)$ and $i \in [k]$ it holds that

$$a_{v,i}^x = \min_{\substack{j\in[k],\\ a\in\{y,z\}}} \left\{a_{v,j}^z + dist_{H^x}(u_j^z, u_i^a)\right\}.$$

Proof. Assume $v \in V(G^y)$. Let $P = (v = w_1, w_2, \ldots, w_\ell)$ be a shortest v-L_i^x path in G^x with $v \in V(G^y)$. Subdivide P into two parts $P = P_1, P_2$ such that P_1 is the maximal subpath of P with $V(P_1) \subseteq V(G^y)$. Let u_1 be the last vertex of P_1. Then, $\omega(P_1) \geq a^y_{v,lab(u_1)}$ and $\omega(P_2) \geq dist_{H^x}(u^y_{lab(u_1)}, u_i^a)$, thus $a^x_{v,i} = \omega(P_1) + \omega(P_2) \geq \min_{j\in[k],a\in\{y,z\}}\{a^y_{v,j} + dist_{H^x}(u_j^y, u_i^a)\}$.

On the other hand, for each $j \in [k]$ and $a \in \{y, z\}$ there is a path in G^x of length $a^y_{v,j} + dist_{H^x}(u_j^y, u_i^a)$: Per definition there is a v-L_j^y path P_1 in G^x with $\omega(P_1) = a^y_{v,j}$ and due to Theorem 4.11, there is a L_j^y-L_i^z path P_2 of cost $\omega(P_2) = dist_{H^x}(u_j^y, u_i^a) - \min_{v\in L_j^y}\omega(v)$ with the property that the second vertex is connected to all vertices of L_j^y. Thus, one can combine the path P_1 with the path P_2 except of the first vertex of P_2. The resulting path is a v-L_i^a path of the desired cost. The equation for $v \in V(G^z)$ follows analogously. □

Second Phase. In this phase, we process the k-expression tree T in a top-down manner and use the local values computed in the first phase to determine the distances in the whole graph G.

Consider an internal node $x \in V(T)$ of the k-expression tree T marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$ and let $y \in V(T)$ and $z \in V(T)$ be the children of x in T. Let again L_i^y resp. L_i^z denote the set of vertices with label i in G^y resp. G^z for $i \in [k]$. For an internal node x with children y and z we will compute for every vertex set L_i^y resp. L_i^z and every vertex $v \in V(G^x)$ the minimum cost of all paths in G that start in v and end in L_i^y resp. L_i^z with the property that the penultimate vertex is in $V(G) \setminus V(G^y)$ resp. in $V(G) \setminus V(G^z)$. Thus, the penultimate vertex is connected to *all* vertices of the vertex set L_i^y resp. L_i^z. It will therefore be convenient not to include the cost of the final vertex in these values (as done in the definition below). Note that we consider the whole graph G in this step instead of just G^x.

Formally, for a node $x \in V(T)$ marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$ with children y and z we compute for every $v \in V(G^x)$ and $i \in [k]$ the following values:

- $d^x_{v,i,y} = \min_{P\in\mathcal{P}} \omega(P) - \min_{u\in L^y_i} \omega(u)$ where $\mathcal{P}$ is the set of all paths in G starting in v, ending in L^y_i, and having the penultimate vertex in $V(G) \setminus V(G^y)$.
- $d^x_{v,i,z} = \min_{P\in\mathcal{P}} \omega(P) - \min_{u\in L^z_i} \omega(u)$ where $\mathcal{P}$ is the set of all paths in G starting in v, ending in L^z_i, and having the penultimate vertex in $V(G) \setminus V(G^z)$.

For a node $x \in V(T)$ marked with $\circ_R$ for some $R\colon [k] \to [k]$ and the child y, we only compute $d^x_{v,i,y}$. We start by computing those values for the root node. We can assume, w.l.o.g., that the root node has label $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$.

Lemma 4.18. *Let $r \in V(T)$ be the root node of the k-expression tree T marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$ and let $y \in V(T)$ and $z \in V(T)$ be the children of r in T. Let H^r be the graph defined in Definition 4.10. Then, for every $v \in V(G^y)$ and for every $i \in [k]$ it holds that*

$$d^r_{v,i,y} = \min_{j\in[k]} \left\{ a^y_{v,j} + dist_{H^r}(u^y_j, v^y_i) \right\} \quad \text{and} \quad d^r_{v,i,z} = \min_{j\in[k]} \left\{ a^y_{v,j} + dist_{H^r}(u^y_j, v^z_i) \right\}.$$

Analogously, for every $v \in V(G^z)$ and for every $i \in [k]$ it holds that

$$d^r_{v,i,y} = \min_{j\in[k]} \left\{ a^z_{v,j} + dist_{H^r}(u^z_j, v^y_i) \right\} \quad \text{and} \quad d^r_{v,i,z} = \min_{j\in[k]} \left\{ a^z_{v,j} + dist_{H^r}(u^z_j, v^z_i) \right\}.$$

Proof. We prove the first equation for the case that $v \in V(G^y)$. The second equation as well as both equations for $v \in V(G^z)$ can be shown analogously.

We first show that $d^r_{v,i,y} \geq \min_{j\in[k]}\{a^y_{v,j} + dist_{H^r}(u^y_j, v^y_i)\}$. Consider a shortest path $P = (p_1, \ldots, p_{n-1}, p_n)$ with $p_1 = v$, $p_n \in L^y_i$, and $p_{n-1} \in V(G) \setminus V(G^y) = V(G^z)$. Let ℓ be the maximal index such that $p_r \in V(G^y)$ for all $1 \leq r \leq \ell$ and let $j \in [k]$ such that $p_\ell \in L^y_j$. It holds that $1 \leq \ell \leq n-2$ since $p_1 \in V(G^y)$ and $p_{n-1} \notin V(G^y)$. Consider the subpaths $P_1 = (p_1, \ldots, p_\ell)$ and $P_2 = (p_{\ell+1}, \ldots, p_n)$. By construction, P_1 is a v-L^y_j path in G^y and hence $\omega(P_1) \geq a^y_{v,j}$. Since $p_{\ell+1}$ is connected to p_ℓ and $p_{\ell+1} \in G^z$, the vertex $p_{\ell+1}$ is connected to every vertex in L^y_j. We prepend P_2 by $p' = \arg\min_{u\in L^y_j} \omega(u)$ and denote the resulting path by P'_2. Since P is a shortest v-L^y_i path and $p_{n-1} \in V(G^z)$, it holds that $p_n = \arg\min_{u\in L^y_i} \omega(u)$. The path P'_2 is an L^y_j-L^y_i path in $G^r = G$ with the property that the second vertex and the penultimate vertex are in $V(G^r) \setminus V(G^y)$ and hence, due to Theorem 4.11, $\omega(P'_2) \geq dist_{H^r}(u^y_j, v^y_i) + \omega(p') + \omega(p_n)$. Note that $\omega(P) = \omega(P_1) + \omega(P'_2) - \omega(p')$. Thus,

$$\begin{aligned}
d^r_{v,i,y} &= \omega(P) - \min_{u\in L^y_i} \omega(u) \\
&= \omega(P_1) + \omega(P'_2) - \omega(p') - \min_{u\in L^y_i} \omega(u) \\
&\geq a^y_{v,j} + dist_{H^r}(u^y_j, v^y_i) + \omega(p_n) - \min_{u\in L^y_i} \omega(u) \\
&= a^y_{v,j} + dist_{H^r}(u^y_j, v^y_i) \\
&\geq \min_{j\in[k]} \left\{ a^y_{v,j} + dist_{H^r}(u^y_j, v^y_i) \right\}.
\end{aligned}$$

For the other direction, we observe that for each $j \in [k]$ there is always a path in G of cost $a^y_{v,j} + dist_{H^r}(u^y_j, v^y_i) + \min_{u\in L^y_i} \omega(u)$ that starts in v, ends in L^y_i, and that has its penultimate vertex in $V(G) \setminus V(G^y)$: For fixed $j \in [k]$ let P_1 be a shortest v-L^y_j path in G^y. By definition, it holds that $\omega(P_1) = a^y_{v,j}$. Let P_2 be a shortest L^y_j-L^y_i path in $G^r = G$ with the property

that the second vertex and the penultimate vertex are in $V(G^r) \setminus V(G^y)$. By Theorem 4.11 it holds that $\omega(P_2) = dist_{H^r}(u_j^y, v_i^y) + \min_{u \in L_j^y} \omega(u) + \min_{u \in L_i^y} \omega(u)$. Since the second vertex of P_2 is connected to *all* vertices of L_j^y we can combine P_1 and P_2 by removing the first vertex of P_2 to get a v-L_i^y path P in G with the property that the penultimate vertex is in $V(G) \setminus V(G^y)$ of cost $a_{v,j}^y + dist_{H^r}(u_j^y, v_i^y) + \min_{u \in L_i^y} \omega(u)$. By definition of $d_{v,i,y}^r$, it follows that $a_{v,j}^y + dist_{H^r}(u_j^y, v_i^y) \geq d_{v,i,y}^r$. Because the argument works for all $j \in [k]$, it follows directly that $d_{v,i,y}^r \leq \min_{j \in [k]} \{a_{v,j}^y + dist_{H^r}(u_j^y, v_i^y)\}$, which completes the proof. □

Next, we show how to propagate those values downwards in the k-expression tree, starting with a node marked with $\circ_R$ for some $R : [k] \to [k]$.

Lemma 4.19. *Let $x \in V(T)$ be an internal node of the k-expression tree T marked with $\circ_R$ for some $R\colon [k] \to [k]$. Let $y \in V(T)$ be the unique child of x and $w \in V(T)$ be the unique ancestor of x in T. Then, for any $v \in V(G^x)$ and $i \in [k]$ it holds that $d_{v,i,y}^x = d_{v,R(i),x}^w$.*

Proof. Since $x \in V(T)$ is marked with $\circ_R$ for some $R\colon [k] \to [k]$ it holds that $unlab(G^y) = unlab(G^x)$ and that $L_i^x = \bigcup_{j:R(j)=i} L_j^y$. Thus, all vertices in $V(G) \setminus V(G^x)$ are either connected to all vertices of L_i^x or to none. Consider a shortest v-L_i^y path P in G with the property that the penultimate vertex is in $V(G) \setminus V(G^y)$. Since $V(G^y) = V(G^x)$, P is also a v-$L_{R(i)}^x$ path with the property that the penultimate vertex is in $V(G) \setminus V(G^x)$. Hence, $d_{v,i,y}^x \geq d_{v,R(i),x}^w$. On the other hand, every shortest v-$L_{R(i)}^x$ path in G with the property that the penultimate vertex is in $V(G) \setminus V(G^x)$ can be changed to a v-L_i^y path by possibly replacing the final vertex with $\arg\min_{u \in L_i^y} \omega(u)$. Since the cost of this vertex is not included in d-values, it follows that $d_{v,R(i),x}^w \geq d_{v,i,y}^x$. □

We now show the propagation of the values for nodes $x \in V(T)$ that are marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$. We start with one specific case and then conclude the general case as a corollary.

Lemma 4.20. *Let $x \in V(T)$ be an internal node of the k-expression tree T marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$. Let $y \in V(T)$ and $z \in V(T)$ be the two children of x in T, let $w \in V(T)$ be the unique ancestor of x in T. Then, for any $v \in V(G^y)$ and $i \in [k]$ it holds that $d_{v,i,y}^x$ is the minimum of the following three values:*

- $d_{v,i,x}^w$,
- $\min\limits_{j \in [k]} \left\{ a_{v,j}^y + dist_{H^x}(u_j^y, v_i^y) \right\}$,
- $\min\limits_{\substack{j \in [k],\\ c \in \{y,z\}}} \left\{ d_{v,j,x}^w + dist_{H^x}(v_j^c, v_i^y) \right\}$.

Proof. We can assume, that w is marked with $\circ_R$ for some $R\colon [k] \to [k]$ and that x is the only child of w.

Let $P = (p_1, \ldots, p_{n-1}, p_n)$ be a shortest v-L_i^y path in G with penultimate vertex in $V(G) \setminus V(G^y)$, i.e., with $p_1 = v$, $p_n \in L_i^y$, and $p_{n-1} \in V(G) \setminus V(G^y)$; thus, $\omega(P) - \omega(p_n) = d_{v,i,y}^x$. We distinguish three cases:

Case 1: $p_{n-1} \in V(G) \setminus V(G^x)$. In this case, P is also a v-L_i^x path such that the penultimate vertex is in $V(G) \setminus V(G^x)$; thus, $d_{v,i,y}^x \geq d_{v,i,x}^w$.

Case 2: $p_{n-1} \in V(G^z)$ and all vertices of P are in G^x. In this case, we can compute the value in the same way as done in Lemma 4.18 for the root node and it holds that $d_{v,i,y}^x =$

$\min_{j\in[k]}\{a^y_{v,j} + dist_{H^x}(u^y_j, v^y_i)\}$.

Case 3: $p_{n-1} \in V(G^z)$ and at least one vertex in P is in $V(G)\setminus V(G^x)$. Let p_ℓ be the last vertex of P that is in $V(G)\setminus V(G^x)$; clearly, $p_{\ell+1} \in V(G^x)$. We split the path P into the two subpaths $P_1 = (p_1, \ldots, p_\ell)$ and $P_2 = (p_{\ell+1}, \ldots, p_n)$. Let $j \in [k]$ such that $p_{\ell+1} \in L^x_j$. Since p_ℓ is connected to $p_{\ell+1}$, the vertex p_ℓ is connected to every vertex in L^x_j. We extend P_1 by $p' = \arg\min_{u\in L^x_j} \omega(u)$ and denote the resulting path by P'_1. Now it holds by definition that $\omega(P'_1) - \omega(p') \geq d^w_{v,j,x}$, as the penultimate vertex p_ℓ of P'_1 is in $V \setminus V(G^x)$. Let further $c \in \{y, z\}$ such that $p_{\ell+1} \in L^c_j$ (note that $L^x_j = L^y_j \dot\cup L^z_j$). Then, $\omega(P_2) - \omega(p_n) \geq dist_{H^x}(v^c_j, v^y_i)$ by Theorem 4.11, as P_2 is a path in G^x. Note that $\omega(P) = \omega(P'_1) + \omega(P_2) - \omega(p')$. Thus, in this case it holds that

$$\begin{aligned}
d^x_{v,i,y} &= \omega(P) - \min_{u\in L^y_i} \omega(u) \\
&= \omega(P'_1) - \omega(p') + \omega(P_2) - \min_{u\in L^y_i} \omega(u) \\
&\geq d^w_{v,j,x} + dist_{H^x}(v^c_j, v^y_i) + \omega(p_n) - \min_{u\in L^y_i} \omega(u) \\
&\geq d^w_{v,j,x} + dist_{H^x}(v^c_j, v^y_i) \\
&\geq \min_{\substack{j\in[k],\\ c\in\{y,z\}}} \left\{ d^w_{v,j,x} + dist_{H^x}(v^c_j, v^y_i) \right\}
\end{aligned}$$

We have seen in the case analysis above that in each case $d^x_{v,i,y}$ is at least the value considered in the case; in particular, it is at least equal to their collective minimum value. On the other hand, for each case there is a path P fulfilling the definition of $d^x_{v,i,y}$ such that $\omega(P) - \min_{u\in L^y_i} \omega(u)$ equals the value of the considered case. Thus, $d^x_{v,i,y}$ is also at most equal to the minimum taken over all three cases. This completes the proof. □

Lemma 4.20 shows how to compute the value $d^x_{v,i,y}$ for any $v \in V(G^y)$. By a similar argumentation one can also compute the value $d^x_{v,i,\beta}$ for any $v \in V(G^\alpha)$ for $\alpha, \beta \in \{y, z\}$.

Corollary 4.21. *Let $x \in V(T)$ be an internal node of the k-expression tree T marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$. Let $y \in V(T)$ and $z \in V(T)$ be the two children of x in T, let $w \in V(T)$ be the unique ancestor of x in T, and let $\alpha, \beta \in \{y, z\}$. Then, for any $v \in V(G^\alpha)$ and $i \in [k]$ it holds that $d^x_{v,i,\beta}$ is the minimum of the following three values:*

- $d^w_{v,i,x}$
- $\min_{j\in[k]} \left\{ a^\alpha_{v,j} + dist_{H^x}(u^\alpha_j, v^\beta_i) \right\}$
- $\min_{\substack{j\in[k],\\ c\in\{y,z\}}} \left\{ d^w_{v,j,x} + dist_{H^x}(v^c_j, v^\beta_i) \right\}$

Third Phase. In the third phase, we traverse the k-expression tree T one final time; The order in which the nodes are visited is immaterial. We go over all nodes x with label $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$ and compute for each pair of vertices (u, v) with $u \in V(G^y)$ and $v \in V(G^z)$ the shortest u-v path in G, with $y \in V(T)$ and $z \in V(T)$ denoting the two children of x in T. Since the leaves of T are in one-to-one correspondence with single-vertex graphs, one for each vertex of G, this procedure will consider every pair of vertices in G at some node $x \in V(T)$. In the second phase we have extended the local values $a^x_{v,i}$ to the values $d^x_{v,i,z}$ that can now be

used to determine the cost of a shortest u-L_i^z path in the whole graph G. To get a u-v path in G, we need to extend this path by a L_i^z-v path in G^z. That is exactly the value $a_{v,i}^z$ if we flip the direction of all edges in G. Formally, for each node $x \in V(T)$ we define the value $a_{v,i}^{x,\leftarrow}$ for all $i \in [k]$ and $v \in V(G)$ as

$$a_{v,i}^{x,\leftarrow} = dist^x(L_i^x, v).$$

We can compute those values as done in the first phase by considering the edge-flipped graph $G^{\leftarrow}$ (with the same vertex weights). Eventually, we can compute the costs of a shortest u-v path in G for $u \in V(G^y)$ and $v \in V(G^z)$.

Lemma 4.22. *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$ and let $y \in V(T)$ and $z \in V(T)$ be the children of x in T. Then, for $u \in V(G^y)$ and $v \in V(G^z)$ it holds that*

$$dist_G(u,v) = \min_{i \in [k]} \left\{ d_{u,i,z}^x + a_{v,i}^{z,\leftarrow} \right\}$$

.

Proof. Let $P = (u = p_1, \ldots, p_n = v)$ be a shortest u-v path in G. Let ℓ be the largest index such that $p_\ell \in V(G) \setminus V(G^z)$. Since $p_1 \in V(G^y)$ and $p_n \in V(G^z)$ this index must exist and it holds that $1 \leq \ell \leq n-1$. Split P into two subpaths $P_1 = (p_1, \ldots, p_\ell)$ and $P_2 = (p_{\ell+1}, \ldots, p_n)$. Let $i \in [k]$ such that $p_{\ell+1} \in L_i^z$. Since p_ℓ is connected to $p_{\ell+1}$, the vertex p_ℓ is connected to every vertex in L_i^z. We extend P_1 by $p' = \arg\min_{u \in L_i^z} \omega(u)$ and denote the resulting path by P_1'. Now, P_1' is a u-L_i^z path with penultimate vertex in $V(G) \setminus V(G^z)$ and, therefore $\omega(P_1') - \omega(p') \geq d_{u,i,z}^x$. Similarly, P_2 is a L_i^z-v path in G^z, implying that $\omega(P_2) \geq a_{v,i}^{z,\leftarrow}$. Thus

$$\begin{aligned} dist_G(u,v) &= \omega(P) \\ &= \omega(P_1') - \omega(p') + \omega(P_2) \\ &\geq d_{u,i,z}^x + a_{v,i}^{z,\leftarrow} \\ &\geq \min_{i \in [k]} \left\{ d_{u,i,z}^x + a_{v,i}^{z,\leftarrow} \right\}. \end{aligned}$$

Conversely, we show that for each $i \in [k]$ there is a u-v path in G of cost $d_{u,i,z}^x + a_{v,i}^{z,\leftarrow}$: For fixed $i \in [k]$ let P' be a shortest u-L_i^z path in G with the property that the penultimate vertex is in $V(G) \setminus V(G^z)$. By definition it holds that $\omega(P') = d_{u,i,z}^x + \min_{w \in L_i^z} \omega(w)$. Let P_1 be obtained from P' by removing the last vertex. Now, $\omega(P_1) = d_{u,i,z}^x$ and P_1 has the property that it starts in u and that its last vertex is adjacent to all vertices of L_i^z. Let P_2 be a shortest L_i^z-v path in G^z. By definition, it holds that $\omega(P_2) = a_{v,i}^{z,\leftarrow}$. Now, we can extend P_1 by P_2 to get a u-v path in G of cost $d_{u,i,z}^x + a_{v,i}^{z,\leftarrow}$. This implies that $dist_G(u,v) \leq d_{u,i,z}^x + a_{v,i}^{z,\leftarrow}$ and, using that the argument works for all $i \in [k]$, that $dist_G(u,v) \leq \min_{i \in [k]} \{ d_{u,i,z}^x + a_{v,i}^{z,\leftarrow} \}$. □

Running time. First, we need to transform the directed clique-width k-expression into a directed NLC-width k-expression tree T, which can be done in linear time $\mathcal{O}(n+m)$ [GWY16].

In the first traversal, we compute for every node $x \in V(T)$ marked with $\circ_R$ for some $R\colon [k] \to [k]$, the values $a_{v,i}^x$ and $c_{i,j}^x$ for all $v \in V(G^x)$ and $i, j \in [k]$ using Lemma 4.9 and Lemma 4.8 in time $\mathcal{O}(k^2)$ resp. $\mathcal{O}(nk)$ per node x. For a node $x \in V(T)$ marked with $\times_{\overrightarrow{S},\overleftarrow{S}}$ for some $\overrightarrow{S}, \overleftarrow{S} \subseteq [k]^2$, we first compute the auxiliary graph H^x as defined in Definition 4.10 in time

$\mathcal{O}(|V(H)| + |E(H)|) = \mathcal{O}(k^2)$ and solve (edge-weighted) ALL-PAIRS SHORTEST PATHS on H^x in time $\mathcal{O}(k^3)$. After this, using Lemma 4.16 resp. Lemma 4.17, we compute each of the k^2 many values $c^x_{i,j}$ in constant time and each of the $n \cdot k$ many values $a^x_{v,i}$ in time $\mathcal{O}(k)$, resulting in a running time of $\mathcal{O}(k^3 + k^2 \cdot n)$ per node $x \in V(T)$. The values $a^{x,\leftarrow}_{v,i}$ can be computed in the same time considering the edge-flipped graph.

In the second phase, we perform a top-down traversal of T to compute for each node $x \in V(T)$ the values $d^x_{v,i,y}$ and $d^x_{v,i,z}$ for all $v \in G^x$ and $i \in [k]$ using Lemma 4.18, Lemma 4.19, and Corollary 4.21. For each node x, we compute at most $2n \cdot k$ values, each in time $\mathcal{O}(k)$, which results in a running time of $\mathcal{O}(nk^2)$ per node of T. Since there are $\mathcal{O}(n)$ nodes in the k-expression tree T, the total running time of the first and second phase is $\mathcal{O}(nk^3 + n^2k^2) = \mathcal{O}(n^2k^2)$.

In the last phase, we consider each pair of vertices $u, v \in V(G)$ exactly once and compute for each pair the shortest distance in time $\mathcal{O}(k)$ using Lemma 4.22. Thus, the running time of the last phase is bounded by $\mathcal{O}(n^2k)$. In total, we obtain the claimed bound of $\mathcal{O}(k^2n^2)$ and have proven Theorem 4.7.

4.3 Triangle Counting Parameterized by Clique-Width

Coudert et al. [CDP19] showed how to solve TRIANGLE COUNTING parameterized by the clique-width cw of a graph G with n vertices and m edges in time $\mathcal{O}(\mathsf{cw}^2(n + m))$. In this section, using an equivalent NLC-width k-expression, we show how to solve TRIANGLE COUNTING in time $\mathcal{O}(k^2n + km)$.

TRIANGLE COUNTING

Input: An undirected graph $G = (V, E)$.
Output: The number of triangles in G.

Note that the length of an NLC-width expression is $\mathcal{O}(n)$, whereas the length of a clique-width expression can be $\Omega(n + m)$ and that one can transform any clique-width k-expression into an equivalent NLC-width k-expression in linear time, cf. [Joh98]. In this section, we will prove the following theorem.

Theorem 4.23. *For every undirected graph $G = (V, E)$, given together with a clique-width k-expression, we can solve* TRIANGLE COUNTING in time $\mathcal{O}(k^2n + km)$.

Algorithm. First, we transform the given clique-width k-expression into an equivalent NLC-width k-expression. For the NLC-width k-expression σ, we consider the expression tree $T = \mathit{tree}(\sigma)$, i.e., each leaf in T is marked with $\bullet_i$ for some $i \in [k]$ and each internal node in T is either marked with $\circ_R$ for some $R\colon [k] \to [k]$ or by $\times_S$ for some $S \subseteq [k]^2$. As done in the previous section, we denote for a node $x \in V(T)$ by G^x the labeled graph defined by the k-expression represented by the subtree of T rooted in x and we define by $L^x_i = \{v \in V(G^x) \mid \mathit{lab}(v) = i\}$ the set of vertices in G^x with label $i \in [k]$. For the root node $r \in V(T)$ it holds that $G^r = G$. We will compute for each vertex $x \in V(T)$ the number of triangles in G^x that we denote by $\triangle^x$. Additionally, we will compute for each $x \in V(T)$ and $i, j \in [k]$ the following auxiliary values $m^x_{i,j}$ and n^x_i:

$$n^x_i = |L^x_i|$$
$$m^x_{i,j} = \left|\left\{\{u, v\} \in E \;\middle|\; u \in L^x_i, v \in L^x_j\right\}\right|$$

We traverse the k-expression tree from bottom to top. First, consider a leaf $x \in V(T)$ marked with $\bullet_\ell$ for some $\ell \in [k]$, i.e., G^x consists of a single vertex v of label ℓ. For this node x it holds for all $i \in [k]$ that either $n_i^x = 1$ if $i = \ell$ or $n_i^x = 0$ if $i \neq \ell$. Moreover, $m_{i,j}^x = 0$ for all $i, j \in [k]$ and $\triangle^x = 0$. Now, consider an internal node $x \in V(T)$ marked with $\circ_R$ for some $R\colon [k] \to [k]$ and let $y \in V(T)$ be the child of x in T. Since $unlab(G^y) = unlab(G^x)$ it holds that $\triangle^x = \triangle^y$. The values n_i^x and $m_{i,j}^x$ for $i, j \in [k]$ can be computed in time $\mathcal{O}(k^2)$ by the following lemma.

Lemma 4.24. *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\circ_R$ for some $R\colon [k] \to [k]$ and let $y \in V(T)$ be the child of x in T. If the values n_i^y and $m_{i,j}^y$ are known for $i, j \in [k]$ then one can compute the values n_i^x and $m_{i,j}^x$ in time $\mathcal{O}(k^2)$.*

Proof. Since it holds that $L_i^x = \cup_{i' \in R^{-1}(i)} L_{i'}^y$, we can compute the values n_i^x for each $i \in [k]$ as follows

$$n_i^x = \sum_{i' \in R^{-1}(i)} n_{i'}^y$$

Similarly, we can compute the values $m_{i,j}^x$ for $i, j \in [k]$ as follows.

$$\begin{aligned} m_{i,j}^x &= \left|\left\{\{u,v\} \in E \;\middle|\; u \in L_i^x, v \in L_j^x\right\}\right| \\ &= \left|\left\{\{u,v\} \in E \;\middle|\; u \in \bigcup_{i' \in R^{-1}(i)} L_{i'}^y, v \in \bigcup_{j' \in R^{-1}(j)} L_{j'}^y\right\}\right| \\ &= \sum_{i' \in R^{-1}(i)} \sum_{j' \in R^{-1}(j)} \left|\left\{\{u,v\} \;\middle|\; u \in L_{i'}^y, v \in L_{j'}^y\right\}\right| \\ &= \sum_{i' \in R^{-1}(i)} \sum_{j' \in R^{-1}(j)} m_{i',j'}^y \end{aligned} \tag{4.1}$$

Since R is a function and can be interpreted as a right-unique and left-total relation, each value $m_{i,j}^y$ appears in Equation (4.1) exactly once, namely in the computation of $m_{R(i),R(j)}$). Thus, to compute the values $m_{i,j}^x$ for all $i, j \in [k]$, we can first initialize the values $m_{i,j}^x$ by zero and later add the value $m_{i,j}^y$ to the (current) value of $m_{R(i),R(j)}^x$ for each $i, j \in [k]$. Hence, we can compute all values $m_{i,j}^x$ in time $\mathcal{O}(k^2)$. □

Next, we consider a node $x \in V(T)$ that is marked with $\times_S$ for some $S \subseteq [k]^2$. Again, we can compute the auxiliary values n_i^x and $m_{i,j}^x$ for $i, j \in [k]$ in time $\mathcal{O}(k^2)$.

Lemma 4.25. *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$ and let $y \in V(T)$ and $z \in V(T)$ be the children of x in T. Then, for any $i, j \in [k]$ it holds that $n_i^x = n_i^y + n_i^z$ and*

$$m_{i,j}^x = \begin{cases} m_{i,j}^y + m_{i,j}^z + n_i^y n_j^z + n_j^y n_i^y & \text{if } (i,j) \in S \text{ and } (j,i) \in S \\ m_{i,j}^y + m_{i,j}^z + n_i^y n_j^z & \text{if } (i,j) \in S \text{ and } (j,i) \notin S \\ m_{i,j}^y + m_{i,j}^z + n_j^y n_i^y & \text{if } (i,j) \notin S \text{ and } (j,i) \in S \\ m_{i,j}^y + m_{i,j}^z & \text{if } (i,j) \notin S \text{ and } (j,i) \notin S \end{cases}$$

Proof. Since $L_i^x = L_i^y \dot\cup L_i^z$ it holds that $|L_i^x| = |L_i^y| + |L_i^z|$. Further, it holds that

$$m_{i,j}^x = \sum_{a,b \in \{y,z\}} \left|\left\{\{u,v\} \in E(G^x) \;\middle|\; u \in L_i^a, v \in L_j^b\right\}\right|$$

while the two summands for $a \neq b$ are equal to the number of edges in respective full join (if present). This proves the claim. □

Using the auxiliary values n_i^x and $m_{i,j}^x$, we can finally compute the number of triangles in G^x as follows.

Lemma 4.26. *Let $x \in V(T)$ be an internal node of a k-expression tree T marked with $\times_S$ for some $S \subseteq [k]^2$ and let $y \in V(T)$ and $z \in V(T)$ be the children of x in T. Then, it holds that*

$$\triangle^x = \triangle^y + \triangle^z \tag{4.2}$$

$$+ \sum_{(i,j)\in S} m_{i,i}^y \cdot n_j^z + m_{j,j}^z \cdot n_i^y \tag{4.3}$$

$$+ \sum_{\substack{(i,j)\in S \\ (i,k)\in S}} m_{j,k}^z \cdot n_i^y + \sum_{\substack{(i,j)\in S \\ (\ell,j)\in S}} m_{i,\ell}^y \cdot n_j^z. \tag{4.4}$$

Proof. We define $E_{new}^x = E(G^x) \setminus (E(G^y) \cup E(G^z))$ as the set of newly created edges in G^x, i.e., E_{new}^x is the union of all full joins between the vertex sets L_i^y and L_j^z for each $(i,j) \in S$. First, we show that each triangle in G^x is counted in the sum above. Any triangle not using an edge in E_{new} is either completely in G^y or G^z and thus, counted in Line 4.2. Since each edge in E_{new} connects a vertex from G^y to a vertex in G^z, there is no triangle using exactly one edge in E_{new}, and since the edge set E_{new} induces a bipartite graph, there is no triangle in G^x using three edges in E_{new}. Hence, all remaining triangles consist of two edges in E_{new} and one edge in $E(G^y)$ or $E(G^z)$. If both edges in E_{new} belong to a full join generated by one pair $(i,j) \in S$, this triangle is counted in Line 4.3. If the two edges in E_{new} belong to full joins generated by different pairs in S, it is counted in Line 4.4. Hence, the above sum is at least the number of triangles in G^x. For the other direction, each summand in the above sum corresponds to a set of unique triangles in G^x, thus, the sum is also at most the number of triangles in G^x. □

Running time. Transforming the clique-width k-expression into an equivalent NLC-width k-expression can be done in linear time. Denote by $T = \mathit{tree}(\sigma)$ the expression tree of the NLC-width k-expression σ. Initializing the values n_i^x, $m_{i,j}^x$, and $\triangle^x$ for all leaves $x \in V(T)$ takes total time $\mathcal{O}(n)$. For an internal node $x \in V(T)$ marked with $\circ_R$ for some $R\colon [k] \to [k]$, we can update those values in time $\mathcal{O}(k^2)$ due to Lemma 4.24. For any node $x \in V(T)$ marked with $\times_S$ for some $S \subseteq [k]$, we can update the values n_i^x and $m_{i,j}^x$ for all $i,j \in [k]$ per node in $\mathcal{O}(k^2)$ due to Lemma 4.25. One can easily bound the computation of $\triangle^x$ due to Lemma 4.26 by $\mathcal{O}(k^3)$, however, we can analyze the total running time more carefully: For a node $x \in V(T)$ that is marked with $\times_{S_x}$ for $S_x \subseteq [k]^2$, we first expand S_x into a $k \times k$ matrix in time $\mathcal{O}(k^2)$; thus, for arbitrary $i,j \in [k]$ we can check in constant time if $(i,j) \in S_x$. Line 4.2 and Line 4.3 of Lemma 4.26 can surely be computed in time $\mathcal{O}(k)$; for Line 4.4 we iterate over the set S_x and check for each pair $(i,j) \in S_x$ and each $\ell \in [k]$ if $(i,\ell) \in S_x$ resp. $(\ell,i) \in S_x$ in time $\mathcal{O}(|S_x|k)$. Now, the total running time of the whole algorithm can be bounded by

$$\sum_{x\in V(T)} |S_x|\, k + k^2 = \left(\sum_{x\in V(T)} |S_x|\right) k + \sum_{x\in V(T)} k^2 \leq mk + 3nk^2$$

Note that it holds that $\sum_{x\in V(T)} |S_x| \leq m$ and note that there are at most $3n$ nodes in T, namely n leaves, $n-1$ nodes marked with $\times_S$ for some $S \subseteq [k]^2$, each potentially followed by a node marked with $\circ_R$ for some $R\colon [k] \to [k]$. As constant factors propagate through the inequality, we have proven Theorem 4.23.

4.4 Conclusion

We extended the studies in the "FPT in P" framework and obtained efficient parameterized algorithms with respect to the parameter clique-width for the problems VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS and TRIANGLE COUNTING. The parameter clique-width is a generalization of the parameter modular-width, for which we got an adaptive algorithm for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS in Section 3.4. The algorithm parameterized by the (much) stronger parameter clique-width cw of time $\mathcal{O}(\mathsf{cw}^2\, n^2)$ is truly subcubic for $\mathsf{cw} \in \mathcal{O}(n^{0.5-\varepsilon})$ for any $\varepsilon > 0$. Note that it also permits us to solve DIAMETER in the same time, complementing the lower bound ruling out any $\mathcal{O}(2^{o(\mathsf{cw})} \cdot n^{2-\varepsilon})$-time algorithm for DIAMETER for any $\varepsilon > 0$, by Coudert et al. [CDP19]. Our algorithm applies to arbitrary vertex-weighted directed graphs without negative cycles. Note that, like all efficient algorithms parameterized by the clique-width, we necessarily assume to be given a suitable expression in general.[3]

We remark that it is important that small clique-width (as well as small modular-width) does not imply the sparsity of the graph. For other parameters that do imply the sparsity of the graph like tree-depth or tree-width (meaning that for parameter value k, the number of edges is bounded by $\mathcal{O}(kn)$, where n denotes the number of vertices in the graph), the algorithm of Pettie and Ramachandran [PR05] would directly yield a running time of $\mathcal{O}(kn^2 + n^2 \log\log n)$, which is nearly optimal. For the same reason, as already mentioned for the parameter modular-width in Section 3.8, it is hopeless to consider edge-weighted graphs with (low) clique-width for most problems, as one could modify an arbitrary input graph by adding all the missing edges with sufficiently large or small weights, depending on the specific problem. E.g., for edge-weighted ALL-PAIRS SHORTEST PATHS, after adding all missing edges with sufficiently large weights, the shortest-path lengths do not change, but the resulting graph is a complete graph of constant clique-width.

As a second result, we slightly improved the running time for the problem TRIANGLE COUNTING from $\mathcal{O}(\mathsf{cw}^2(n+m))$ to $\mathcal{O}(\mathsf{cw}^2\, n + m)$ mainly due to the use of an NLC-width k-expression instead of a clique-width k-expression. Thus, we achieved an adaptive algorithm for the problem TRIANGLE COUNTING for dense graphs if one rules out the use of fast matrix multiplication, i.e., even in the worst-case of $\mathsf{cw} \in \Theta(n)$ the running time of this algorithm is not worst than the best combinatorial unparameterized algorithms on dense graphs, and even $\mathsf{cw} \in o(n)$ yields an improvement. A natural further-reaching question regarding TRIANGLE COUNTING is whether it is possible to develop a parameterized algorithm that is also adaptive towards non-combinatorial or sparse graphs, e.g., a parameterized algorithm utilizing fast matrix multiplication to count the number of triangles.

[3] As mentioned in the beginning of this chapter, for some graph classes of bounded clique-width one can compute the corresponding k-expression in linear time, e.g., distance-hereditary graphs, or graphs of bounded modular-width, split-width, or tree-width.

5

Triangle Counting Parameterized by the Twin-Width of the Input

Recently, the parameter twin-width was introduced by Bonnet et al. [BKTW20] over a series of papers. The *twin-width* of a graph G, denoted by $\mathsf{tww}(G)$, also measures the distance of a graph from being a cograph (alike modular-width and clique-width). As seen in Section 3.1, a graph is a cograph if and only if the modular decomposition of G does not admit any prime node. Cographs can also be characterized as those graphs that arise from an isolated vertex by a sequence of augmentations to the graph in form of adding a true or false twin to any vertex [MT16]. In other words, cographs are exactly those graph that can be contracted to a single vertex by iterating contractions of twins [BKTW20]. Intuitively, a graph has bounded twin-width if one can contract the graph into a single vertex by a sequence of contraction of *near*-twins that are pairs of vertices whose neighborhood only differs by a bounded number of elements. The key point for twin-width is now to keep track of the error by another type of edges, called red edges; and one requires that the maximum number of incident red edges of each vertex stays bounded, or, respectively, is at most the twin-width. See Section 5.1 for the precise definition of twin-width.

Bonnet et al. [BKTW20] showed that every graph of boolean-width k has twin-width at most $2^{k+1} - 1$, i.e., it holds that $\mathsf{tww}(G) \leq 2^{\mathsf{boolw}(G)+1} - 1$ for every graph G. This implies that the class of bounded twin-width graphs contains all graph that have bounded parameter value for all introduced parameters in Section 2.3, e.g., bounded boolean-width, bounded rank-width, bounded clique-width, bounded tree-width, or bounded branch-width.[1] On the other hand, for every positive integers d and n, the d-dimensional n-grid has twin-width at most $3d$ [BKTW20]. Since all above mentioned parameters are unbounded on 2-dimensional $n \times n$-grids, one cannot bound any of those parameters by a function depending on the twin-width. Furthermore, the class of bounded twin-width contains several more graph classes such as unit interval, proper-

[1] We refer to Section 2.3 for a short definition of the mentioned parameters.

minor closed, or H-minor free graphs [BGK$^+$21b]. For some of those classes, e.g., proper minor closed classes and bounded rank-width graph, it is possible to compute in polynomial time a sequence of d-contractions, witnessing that the twin-width is at most d [BGK$^+$21a]. In general, determining the twin-width of a given graph can not be done efficiently, as even deciding if the twin-width of a graph is at most 4 is NP-hard [BBD22]. Thus, we will assume that a graph is given by a d-contraction sequence.

While bounded twin-width generalizes many other graph classes, Bonnet et al. [BKTW20] showed that one can decide first-order (FO) formulas in fixed-parameter time (FPT) on graphs of bounded twin-width, if a corresponding witness is given. I.e., given a d-contraction sequence of a graph G with n nodes and a first-order sentences φ, one can decide $G \models \varphi$ in time $f(|\varphi|, d) \cdot n$ for some computable non-elementary function f. Moreover, various intractable problems like INDEPENDENT SET, DOMINATING SET, and CLIQUE can be solved in time $2^{\mathcal{O}(d)}n$-time [BGK$^+$21b] on graphs with n vertices and twin-width at most d.

In this chapter, we show how to solve the TRIANGLE COUNTING problem on graphs with n vertices and m edges in time $\mathcal{O}(d^2n + m)$, with d denoting the twin-width of the graph, assuming that a d-contraction sequence is given in a compact way (as a naive representation of a d-contraction sequence can be of length $\Omega(n(n + m))$).

We stress that even though every graph of bounded clique-width also has bounded twin-width, the twin-width of the graph can only be exponentially bounded, i.e., it holds that $\mathsf{boolw}(G) \leq 2^{\mathsf{tww}(G)+1} - 1$ with $\mathsf{boolw}(G)$ denotes the more general parameter boolean-width and it holds that $\mathsf{cw}(G) \leq 2^{\mathsf{boolw}(G)}$ [BTV11]. Thus, although the $\mathcal{O}(d^2n+m)$ algorithm also implies a linear time algorithm for solving TRIANGLE COUNTING on bounded clique-width graph, this algorithm does not generalize the algorithm for TRIANGLE COUNTING parameterized by the clique-width of the input graph (cf. Section 4.3), since the dependency on the clique-width of the presented algorithm in this chapter might be much worse.

Overview. In Section 5.1, we formally define the twin-width of a graph G and define the d-contraction sequence of a graph G. We will also discuss how to encode a d-contraction sequence efficiently in linear space. The description of the algorithm for TRIANGLE COUNTING running in time $\mathcal{O}(d^2n + m)$ follows in Section 5.2. We conclude in Section 5.3.

5.1 Definition of Twin-Width.

A *trigraph* G is a triple $G = (V, E, R)$ where $E \subseteq \binom{V}{2}$ and $R \subseteq \binom{V}{2}$ are two disjoint sets of undirected edges. We refer to an edge in E as a black edge and to an edge in R as a red edge. By setting $R = \emptyset$, one can interpret any graph (V, E) as a trigraph $(V, E, \emptyset)$. In that sense, we will sometimes call a graph $G = (V, E)$ a trigraph G and mean $G = (V, E, \emptyset)$ by a slight abuse of notation. For any trigraph $G = (V, E, R)$ and any vertex $v \in V$, we denote by $N_R(v)$ the vertices that are adjacent to v via a red edge, i.e., $N_R(v) = \{u \in V \mid \{u, v\} \in R\}$. Analogously, we define $N_E(v) = \{u \in V \mid \{u, v\} \in E\}$. For a vertex $v \in V$, we denote by $|N_R(v)|$ the *red degree* of vertex v. A trigraph $G = (V, E, R)$ with maximum red degree at most d is called a d-trigraph. For a trigraph $G = (V, E, R)$ and two vertices $u, v \in V$, we define $G/u, v = (V', E', R')$ as the trigraph obtained from G by contracting u and v into a new vertex w and after updating the edge sets in the following way: A vertex x is adjacent to the new vertex w in $G/u, v$ by a black edge if and only if x is adjacent to u *and* to v in G by a black edge in G. Moreover, x is non-adjacent to w in $G/u, v$, if x is neither adjacent to u nor to v in G (via any edge). In all other cases, x is adjacent to w by a red edge in $G/u, v$. Formally, $V' = (V \setminus \{u, v\} \cup \{w\})$ with

the following edges being incident to w:

- $\{w, x\} \in E'$ if and only if $\{u, x\} \in E$ and $\{v, x\} \in E$;
- $\{w, x\} \notin E' \cup R'$ if and only if $\{u, x\} \notin E \cup R$;
- $\{w, x\} \in R'$ otherwise.

All edges that are not incident to u nor to v in G remain unchanged in $G/u, v$. We stress that u and v do not need to be adjacent. We also say that the graph $G/u, v$ is a *contraction of* the graph G. For any integer $d \geq 0$, if both G and $G/u, v$ are d-trigraphs, we call $G/u, v$ a *d-contraction* of G. A trigraph G is *d-collapsible* if there exists a sequence of d-contractions which contracts G to a single vertex. The twin-width of a a graph G denotes the smallest d such that there exists a d-contraction sequence of G into a single vertex.

Definition 5.1 (twin-width, [BKTW20])**.** Let $G = (V, E)$ be a graph. The *twin-width* of G, denoted by $\mathsf{tww}(G)$, the the minimum integer $d \geq 0$, such that $(V, E, \emptyset)$ is d-collapsible.

In other words, for any graph G with $\mathsf{tww}(G) = d$, there exists a sequence of trigraphs $G_n, G_{n-1}, \ldots, G_2, G_1$ with $G_n = G$, $G_1 = K_1$ and G_k is a d-contraction of G_{k+1} for $k \in [n-1]$. See Figure 5.1 for an example of a d-contraction sequence. To represent a d-contraction sequence efficiently using only $\mathcal{O}(n)$ space, we just specify the vertices that get contracted in each step. For this, we enumerate the set of vertices, while the new arising vertex after each contraction is identified by the next greater index of this enumeration.

Definition 5.2 (Compact representation of a d-contraction)**.** Let $G = G_n, G_{n-1}, \ldots, G_1 = K_1$ be a d-contraction sequence of an n-vertex graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$. Then, we call $(v_{i_k}, v_{j_k})_{n \geq k \geq 2}$ with $G_{k-1} = G_k / v_{i_k}, v_{j_k}$ a *compact representation* of a d-contraction sequence. The graph G_{k-1} results from G_k by contracting the two vertices v_{i_k} and v_{j_k} into a new vertex v_{2n-k+1} for $k \in [2, n]$.

As additional notation, for a d-contraction sequence $G_n, G_{n-1}, \ldots, G_2, G_1$ and a vertex $v \in V(G_k)$ with $k \in [n]$, we denote by $v(G)$ the subset of vertices in G eventually contracted into v in G_k and we denote $G^v = G[v(G)]$.

5.2 Algorithm

In this section, we will show how to solve TRIANGLE COUNTING parameterized by the twin-width of the given graph. We denote the number of triangles in a graph by $\#T(G)$, i.e., $\#T(G) = |\{\{x, y, z\} \subseteq \binom{V}{3} \mid \{x, y\}, \{y, z\}, \{z, x\} \in E\}|$.

TRIANGLE COUNTING

Input: An undirected graph $G = (V, E)$.
Output: The number of triangles $\#T(G)$ in G.

Theorem 5.3. *Let $G = (V, E)$ be an undirected graph with $\mathsf{tww}(G) = d$, and let a compact representation of a d-contraction sequence be given. Then, one can solve* TRIANGLE COUNTING *in time $\mathcal{O}(d^2 n + m)$.*

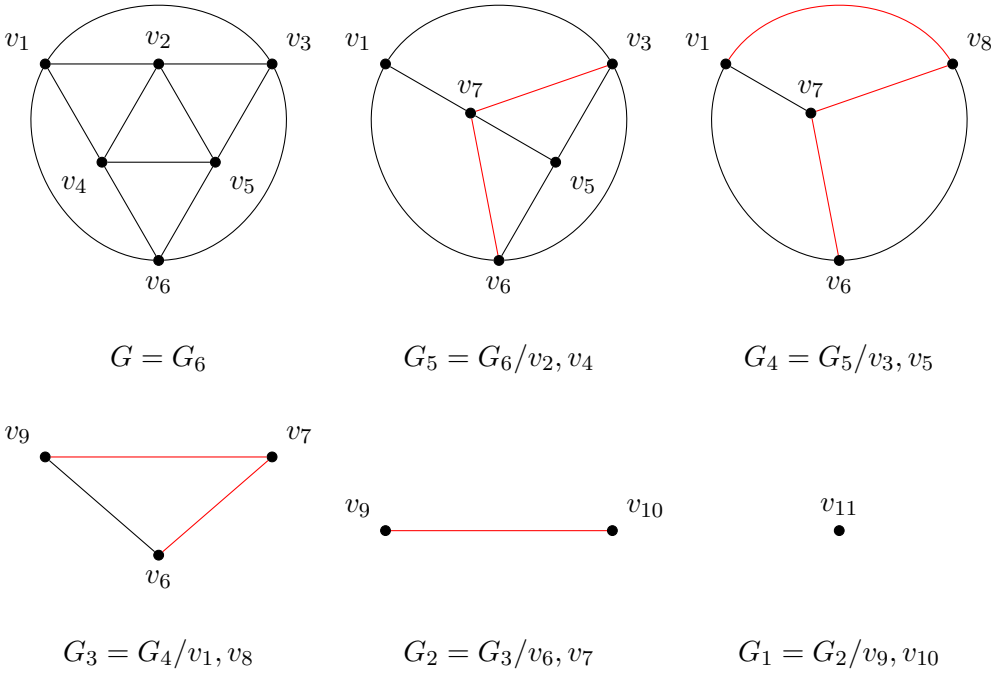


Figure 5.1: Example of a 2-contraction sequence for the octahedron graph G with the compact representation $((v_2, v_4), (v_3, v_5), (v_1, v_8), (v_6, v_7), (v_9, v_{10}))$.

Using the compact representation of the d-contraction sequence as defined in Definition 5.2, we gradually construct the graphs $G = G_n, G_{n-1}, \ldots, G_1 = K_1$. Consider a trigraph $G_k = (V_k, E_k, R_k)$ of the contraction sequence of G for $k \in [n]$ and a fixed triangle $\{a, b, c\}$ in G with $a, b, c \in V(G)$. The vertices of the triangle can be in subgraphs corresponding to one, two, or three vertices of V_k. More formally, we observe the following:

Observation 5.4. *Let $G = G_n, \ldots, G_1 = K_1$ be a contraction sequence of a graph G, let $G_k = (V_k, E_k, R_k)$ be a trigraph of the contraction sequence, and let $\{a, b, c\}$ be a triangle in G for $a, b, c \in V(G)$. Then, exactly one of the following statements is true for $x, y, z \in V_k$ (after possibly reordering a, b, and c):*

(i) $a \in x(G)$, $b \in y(G)$, $c \in z(G)$ *with* $\{x, y\}, \{y, z\}, \{z, x\} \in E_k$

(ii) $a \in x(G)$, $b \in y(G)$, $c \in z(G)$ *with* $\{x, y\}, \{y, z\} \in E_k$, $\{z, x\} \in R_k$

(iii) $a \in x(G)$, $b \in y(G)$, $c \in z(G)$ *with* $\{x, y\} \in E_k$, $\{y, z\}, \{z, x\} \in R_k$ $(\star)$

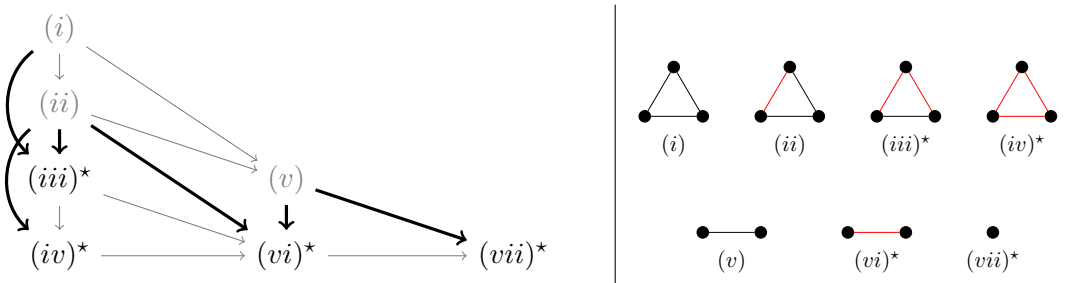
(iv) $a \in x(G)$, $b \in y(G)$, $c \in z(G)$ *with* $\{x, y\}, \{y, z\}, \{z, x\} \in R_k$ $(\star)$

(v) $a \in x(G)$ *and* $b, c \in y(G)$ *with* $\{x, y\} \in E_k$

(vi) $a \in x(G)$ *and* $b, c \in y(G)$ *with* $\{x, y\} \in R_k$ $(\star)$

(vii) $a, b, c \in x(G)$ $(\star)$

Let $G = G_n, G_{n-1}, \ldots, G_1 = K_1$ be a d-contraction sequence of a graph G with n vertices and let $G_i = (V_i, E_i, R_i)$ for $i \in [n]$. Consider a fixed trigraph G_k in this contraction sequence that will be contracted into $G_{k-1} = G_k/v_{i_k}, v_{j_k}$ according to the contraction sequence for



(a) Diagram of case transitions.

(b) Illustration of the different cases.

Figure 5.2: Possible case transitions of a triangle in G from G_k to G_{k-1} for $k \in [2, n]$ regarding the cases described in Observation 5.4. The number of all triangles that correspond to a case marked by a star will be stored throughout the algorithm. Self-loops are omitted in the diagram. Transitions from an unmarked case to a marked case are represented by thick arrows.

$v_{i_k}, v_{j,k} \in V(G_k)$. For simplicity, we define $u := v_{i_k}$ and $v := v_{j_k}$ as the two vertices in $V(G_k)$ that get contracted into the new vertex $w := v_{2n-k+1}$ of G_{k-1}. A fixed triangle T of G might transition from a case in G_k to a different case in G_{k-1} if T consists of vertices in $u(G)$ or $v(G)$. If T consists of vertices in both $u(G)$ and $v(G)$, one vertex less is needed in G_{k-1} to specify T. If T does admit a non-empty cut with only one of the two sets $u(G)$ or $v(G)$, the incident edges of u (resp. v) in G_{k-1} might turn red. See Figure 5.2 for a full diagram of all possible case transitions for a triangle in G from G_k to G_{k-1}.

Initially, each triangle of G is of Case (i), and eventually all triangles will be of Case (vii). Over the course of the algorithm, we explicitly store the number of all triangles of G that appear in G_k as a case marked by a star $(\star)$ in Observation 5.4. Since Case (vii) is a marked case, we will eventually count all triangles in G. For algorithmic reasons, also Case (iii), (iv), and (vi) are marked by a star and we will count a triangle once it *first* becomes a triangle of a marked case. We point out that once a triangle in G is a triangle of a marked case in G_k, it will remain a triangle of a marked case in all G_i for $1 \leq i \leq k$.

Additionally to the number of triangles in G corresponding to a marked case, we keep track of the number of vertices and the number of edges of those subgraphs of G that get contracted into a vertex x in G_k, i.e., we compute the values $n_x = |V(G^x)|$ and $m_x = |E(G^x)|$ for each vertex $x \in V(G_k)$. Further, we store for each red edge $\{x, y\} \in R_k$ the number of edges between G^x and G^y, i.e., $\mu_{x,y} = |\{\{a, b\} \in E(G) \mid a \in x(G), b \in y(G)\}|$. Note that each black edge corresponds to a full join between the respective subgraphs in G.

For the vertex $w \in V(G_{k-1})$ that is the contraction of the two vertices $u, v \in V_k$, the number of vertices in G^w is the sum of these numbers in G^u and G^v. For the number of edges, we also need to add the number of edges between G^u and G^v. Thus, it holds that $n_w = n_u + n_v$ and

$$m_w = \begin{cases} m_u + m_v + n_u \cdot n_v & \text{if } \{u, v\} \in E_k, \\ m_u + m_v + \mu_{u,v} & \text{if } \{u, v\} \in R_k, \\ m_u + m_v & \text{otherwise.} \end{cases}$$

For every other vertex $x \in V(G_{k-1})$, $x \neq w$, these values remain unchanged. Finally, for any vertex $x \in V(G_{k-1})$, such that $\{w, x\} \in R(G_{k-1})$ the number of edges between G^w and G^x can

Algorithm 1: TriangleCounting(G)

Input: A graph $G = (V, E)$ and a compact representation of a contraction sequence $(v_{i_k}, v_{j_k})_{n \geq k \geq 2}$
Output: The number of triangles t in the graph G

```
1  t := 0
2  R = ∅
3  for every vertex x ∈ V do
4      n_x := 1                                  // number of vertices in G^x
5      m_x := 0                                  // number of edges in G^x
6  for k = n down to 2 do
7      u := v_{i_k}
8      v := v_{j_k}
9      w := v_{2n-k+1}
10     UpdateAuxiliaryValues(G, u, v, w)
11     if {u, v} ∈ E then
12         t = t + n_u · m_v + n_v · m_u         // Case (v) → Case (vii)
13     for each x ∈ N_R(w) do
14         TriCountOneNeighbor(G, u, v, w, x)
15         TriCountTwoNeighbors(G, u, v, w, x)
16     UpdateGraph(G, u, v, w)
17 return t
```

be computed as follows:

$$\mu_{w,x} = \begin{cases} n_u \cdot n_x & \text{if } \{u,x\} \in E_k \text{ and } \{v,x\} \notin (E_k \cup R_k), \\ n_v \cdot n_x & \text{if } \{u,x\} \notin (E_k \cup R_k) \text{ and } \{v,x\} \in E_k, \\ \mu_{u,x} & \text{if } \{u,x\} \in R_k \text{ and } \{v,x\} \notin (E_k \cup R_k), \\ \mu_{v,x} & \text{if } \{u,x\} \notin (E_k \cup R_k) \text{ and } \{v,x\} \in R_k, \\ \mu_{u,x} + n_v \cdot n_x & \text{if } \{u,x\} \in R_k \text{ and } \{v,x\} \in E_k, \\ \mu_{v,x} + n_u \cdot n_x & \text{if } \{u,x\} \in E_k \text{ and } \{v,x\} \in R_k, \\ \mu_{u,x} + \mu_{v,x} & \text{if } \{u,v\} \in R_k \text{ and } \{v,x\} \in R_k. \end{cases}$$

For every other vertex $y \in V(G_{k-1})$, $y \neq w$, such that $\{x, y\} \in R(G_{k-1})$, the value $\mu_{x,y}$ remains unchanged.

We can now describe the algorithm that takes a compact representation of a d-contraction sequence as an input. In each iteration k of the algorithm, the two vertices v_{i_k} and v_{j_k}, given in the compact representation of the contraction sequence, are contracted to form the trigraph G_{k-1}. We then update the auxiliary values n_x, m_x, and $\mu_{x,y}$ that have changed for $x, y \in V(G_{k-1})$. After this, we count those triangles that correspond to a marked case in G_{k-1} but to an unmarked case in G_k. More precisely, the transition from Case (v) to Case (vii) is dealt by the main Algorithm. In addition, the procedure TriCountOneNeighbor focuses on the transitions from Case (ii) and Case (v) to Case (vi) whereas the procedure TriCountTwoNeighbors handles the transitions from Case (i) and Case (ii) to Case (iii) and from Case (ii) to Case (iv). See Algorithm 1 for the pseudocode of the main algorithm. For algorithmic purposes, we assume that we are given a graph $G = (V, E)$ that will, over the course of the algorithm, be updated into

Algorithm 2: TriCountOneNeighbor(G, u, v, w, x)

```
1 if {u,x} ∈ E then
2     t = t + n_u · m_x + n_x · m_u                    // Case (v) → Case (vi)
3     if {u,v} ∈ E and {v,x} ∈ R then
4         t = t + μ_{v,x} · n_u                        // Case (ii) → Case (vi)
5 else if {v,x} ∈ E then
6     t = t + n_v · m_x + n_x · m_v                    // Case (v) → Case (vi)
7     if {u,v} ∈ E and {u,x} ∈ R then
8         t = t + μ_{u,x} · n_v                        // Case (ii) → Case (vi)
```

the successive trigraphs defined by the contraction sequence. The variable t in the algorithm represents the number of triangles in G computed so far.

The procedure UpdateAuxiliaryValues is not given but will update the values n_x, m_w, and $\mu_{w,x}$ as explained previously whereas the pseudocode of the procedures TriCountOneNeighbor and TriCountTwoNeighbors are depicted in Algorithm 2 and Algorithm 3. Finally, UpdateGraph performs the actual contraction of the graph G. The pseudocode of this procedure is omitted. Note that although the set $N_R(w)$ in Line 13 is the set all those vertices that will be adjacent to w via a red edge in $G_{k-1} = G_k/u, v$, the graph itself is not updated until Line 16. In the procedure TriCountOneNeighbor, we consider a single incident red edges of the newly introduced vertex w to an adjacent vertex $x \in N_R(w)$. We then investigate the different edges between u, v, and x to detect triangles in G that transition from Case (ii) or Case (v) to Case (vi), see Algorithm 2.

Finally, in the procedure TriCountTwoNeighbors we focus on the edges between the newly introduced vertex w and two of its neighbors $x, y \in N_R(w)$. We then consider the different edges between u, v, x, and y to detect triangles in G that transitions from Case (i) or Case (ii) to Case (iii) and from Case (ii) to Case (iv). See Algorithm 3 for the pseudocode of this procedure. We have now described the algorithm and can prove Theorem 5.3.

Proof of Theorem 5.3. We first prove the correctness of our algorithm. Given the compact representation of the d-contraction sequence $(v_{i_k}, v_{j_k})_{n \geq k \geq 2}$, the algorithm generates iteratively the contraction sequence $G = G_n, G_{n-1}, \ldots, G_1 = K_1$ with $G_{k-1} = G_k/v_{i_k}, v_{j_k}$ using the procedure UpdateGraph at the end of each iteration. The values n_x, m_x, and $\mu_{x,y}$ are updated by the procedure UpdateAuxiliaryValues in each iteration as described previously.

To prove that the final value of t is equal to the number of triangles in G, we will prove that the following invariant is true at the beginning of each iteration, i.e., for each graph $G_k = (V_k, E_k, R_k)$ in the contraction sequence for $k \in [n]$. Recall that we denote by $\#T(G)$ the number of all triangles in G.

$$\#T(G) = t_k + \underbrace{\sum_{\substack{\{x,y\},\\ \{y,z\},\{x,z\} \in E_k}} n_x n_y n_z}_{\text{Case } (i)} + \underbrace{\sum_{\substack{\{x,z\} \in R_k\\ \{x,y\},\{y,z\} \in E_k}} \mu_{x,z} n_y}_{\text{Case } (ii)} + \underbrace{\sum_{\{x,y\} \in E_k} (n_x m_y + m_x n_y)}_{\text{Case } (v)}$$

We denote by t_k the current value of t at the start of iteration k (and t_1 the final value after iteration $k = 2$). For $k = n$, the value of t_n is initialized to zero, $R_n = \emptyset$, $m_x = 0$, and $n_x = 1$

Algorithm 3: TriCountTwoNeighbors(G, u, v, w, x)

```
1  for each y ∈ V such that {x,y} ∈ R and {w,y} will be black do
2      if {u,x} ∈ E then
3          t = t + μ_{x,y} · n_u                          // Case (ii) → Case (iii)
4      else if {v,x} ∈ E then
5          t = t + μ_{x,y} · n_v                          // Case (ii) → Case (iii)
6  for each y ∈ V such that {w,y} will be red do
7      if {x,y} ∈ E then
8          if {u,x} ∈ E and {u,y} ∈ E then
9              t = t + n_u · n_x · n_y                    // Case (i) → Case (iii)
10         else if {v,x} ∈ E and {v,y} ∈ E then
11             t = t + n_v · n_x · n_y                    // Case (i) → Case (iii)
12         if {u,x} ∈ R and {u,y} ∈ E then
13             t = t + μ_{u,x} · n_y                      // Case (ii) → Case (iii)
14         else if {v,x} ∈ R and {v,y} ∈ E then
15             t = t + μ_{v,x} · n_y                      // Case (ii) → Case (iii)
16         if {u,x} ∈ E and {u,x} ∈ R then
17             t = t + μ_{u,x} · n_x                      // Case (ii) → Case (iii)
18         else if {v,x} ∈ E and {v,y} ∈ R then
19             t = t + μ_{v,x} · n_x                      // Case (ii) → Case (iii)
20     else if {x,y} ∈ R then
21         if {u,x} ∈ E and {u,y} ∈ E then
22             t = t + μ_{x,y} · n_u                      // Case (ii) → Case (iv)
23         else if {v,x} ∈ E and {v,y} ∈ E then
24             t = t + μ_{x,y} · n_v                      // Case (ii) → Case (iv)
```

for all $x \in V_n$, cf. Line 1 to 5 of Algorithm 1. Therefore, the invariant simplifies to the second summand only, which is indeed the desired number of all triangles in G. We will show that the value of the invariant will never change. Thus, for $i = 1$, it will hold that $\#T(G) = t_1 + 0 + 0 + 0$ and the correctness of Algorithm 1 follows.

As described in Observation 5.4, we distinguish seven cases of a possible occurrence of a triangle of G in G_k. In the beginning, all triangles in G are of Case (i) but some may change from one case to another one whenever we contract v_{i_k} and v_{j_k} in G_k to get G_{k-1}. For a fixed triangle, all possible case transitions are depicted in Figure 5.2. Note that the triangles of G of Case (i), (ii), or (v), are counted directly by the corresponding sums in the invariant. We are left to show that the (current) value of t_k is indeed the count of all triangles of G that appear in G_k as Case (iii), (iv), (vi), or (vii). Recall that once a fixed triangle is of one of the latter cases, this triangle can never transition back to an unmarked case.

By induction, we can assume that the invariant is true for G_k. To prove the invariant for $k-1$, we keep track of all triangles whose case changes from G_k to G_{k-1} regarding Observation 5.4. Note that we only need to consider the triangles that are of a case that is not marked by a star in G_k, but in a case that is marked in G_{k-1}. Let $G_{k-1} = G_k/u, v$ and let w be the new vertex of

G_{k-1} in which u and v got contracted. In the following, we assume $\{a, b, c\}$ with $a, b, c \in V(G)$ the be a triangle that is of an unmarked case in G_k but of a marked case in G_{k-1}. We consider all possible transition from an unmarked case to a marked case, cf. Figure 5.2:

Case (i) to Case (iii): This implies that (after possibly reordering a, b, and c) there exists $x, y \in V_{k-1}$ with $a \in w(G)$, $b \in x(G)$, $c \in y(G)$, $\{w, x\}, \{w, y\} \in R_k$, and $\{x, y\} \in E_k$. Since w is the contraction of u and v and $\{a, b, c\}$ was a triangle of Case (i) in G_k, it holds that either $a \in u(G)$ with $\{u, x\}, \{u, y\} \in E_k$ or $a \in v(G)$ with $\{v, x\}, \{v, y\} \in E_k$. In the former case, it is counted in the procedure TriCountTwoNeighbors, Line 9. In the latter case it is counted in the same procedure in Line 11.

Case (ii) to Case (iii): This implies that (after possibly reordering a, b, and c) there exists $x, y \in V_{k-1}$ with $a \in w(G)$, $b \in x(G)$, and $c \in y(G)$. Let us first assume that $\{x, y\} \in R_{k-1}$. Since $\{a, b, c\}$ is a triangle of Case (ii) in G_k, it holds that either $\{u, x\}, \{u, y\} \in E_k$ or $\{v, x\}, \{v, y\} \in E_k$. In the former case, it is counted in the procedure TriCountTwoNeighbors, Line 3, and in the latter case, in Line 5. Note that in Line 3 it necessarily holds that also $\{u, y\} \in E_k$ since $\{w, y\} \in E_{k-1}$, resp. that in Line 5 it necessarily holds that also $\{v, y\} \in E_k$. Now assume that $\{x, y\} \in E_{k-1}$, i.e., $\{w, x\}, \{w, y\} \in R_{k-1}$. Since $\{a, b, c\}$ is a triangle of Case (ii) in G_k, it now holds that either $\{u, x\} \in R_k$ and $\{u, y\} \in E_k$ (counted in the procedure TriCountTwoNeighbors, Line 13); $\{v, x\} \in R_k$ and $\{v, y\} \in E_k$ (Line 15); $\{u, x\} \in E_k$ and $\{u, y\} \in R_k$ (Line 17); or $\{v, x\} \in E_k$ and $\{v, y\} \in R_k$ (Line 19). Note that since $\{w, x\}$, $\{w, y\} \in R_{k-1}$, it is not possible that the first two or the last two cases occur simultaneously.

Case (ii) to Case (iv): This implies that (after possibly reordering a, b, and c) there exists $x, y \in V_{k-1}$ with $a \in w(G)$, $b \in x(G)$, $c \in y(G)$ and $\{w, x\}, \{w, y\}, \{x, y\} \in R_{k-1}$. Note that $\{x, y\} \in R_{k-1}$ implies that $\{x, y\} \in R_k$. Since $\{a, b, c\}$ is of Case (ii) in G_k, it needs to hold that either $\{u, x\}, \{u, y\} \in E_k$ or $\{v, x\}, \{v, y\} \in E_k$. The former is counted in the procedure TriCountTwoNeighbors, Line 22 and the latter in Line 24.

Case (ii) to Case (vi): This implies that (after possibly reordering a, b, and c) there exists $x \in V_{k-1}$ with $\{w, x\} \in R_{k-1}$, $a, b \in w(G)$, and $c \in x(G)$. Since $\{a, b, c\}$ is of Case (ii) in G_k, it holds that $\{u, v\} \in E_k$ (otherwise the edge $\{w, x\}$ would not be in R_{k-1}) and that either $\{u, x\} \in R_k$ and $\{v, x\} \in E_k$ (counted in the procedure TriCountOneNeighbor, Line 8), or $\{v, x\} \in R_k$ and $\{u, x\} \in E_k$ (Line 4).

Case (v) to Case (vi): If $\{a, b, c\}$ is a triangle of Case (v) in G_k, this black edge is either incident to u (counted in the procedure TriCountOneNeighbor, Line 2) or to v (Line 6).

Case (v) to Case (vii): Finally, if a triangle $\{a, b, c\}$ is of Case (v) in G_k and of Case (vii) in G_{k-1} it now holds that $a, b, c \in w(G)$ and such a triangle is counted in Algorithm 1, Line 12.

Thus, the number of all the triangles of G that are of Case (iii), (iv), (vi), and (vii) in G_{k-1} is indeed computed and stored in the variable t after the iteration k. Since the algorithm only increases t whenever a triangle transitions from an unmarked case to a marked case after contraction, the value t is exactly the desired value.

We are left to show the desired running time. We store the graph in sorted adjacency lists, which can be initially realized in time $\mathcal{O}(n + m)$ using a linear-time sorting algorithm to sort the vertices $v_1, \ldots, v_n$. To contract the two vertices u and v in each iteration, we can scan the sorted adjacency lists of u and v to identify the red neighborhood and black neighborhood of w. Since w has at most d incident red edges and since we decrease the number of total edges by one for each neighbor that is connected to u and v via a black edge, the total running time for every call of the procedure UpdateGraph, sums up to $\mathcal{O}(dn + m)$. Since the auxiliary values only needs to be computed for the new vertex w and for the incident red edges of w, they can be updated in time $\mathcal{O}(d)$ per iteration. Eventually, it takes $\mathcal{O}(dn)$ for every call of the procedure UpdateAuxiliaryValues. Finally, the procedures TriCountOneNeighbor and

TriCountTwoNeighbors are called at most d times per iteration, taking respectively $\mathcal{O}(1)$ and $\mathcal{O}(d)$ time. Thus, the overall running time of Algorithm 1 is bounded by $\mathcal{O}(d^2n + m)$ and we have proven Theorem 5.3. □

5.3 Conclusion

In this section, we have obtained an efficient parameterized algorithm for triangle counting parameterized by the twin-width of the input graph, assuming that a d-contraction sequence is given in a compact representation. Our algorithm is based on dynamic programming and stores a few values that need to be updated at each contraction step. The presented algorithm is adaptive on dense inputs regarding combinatorial algorithms as it runs in time $\mathcal{O}(d^2n + m)$ whereas the best unparameterized combinatorial algorithm for triangle counting on dense inputs takes time $\mathcal{O}(n^3)$. In particular, our algorithm runs in linear time on graphs of bounded twin-width, which contains several other graph classes (given a compact d-contraction sequence).

Bonnet et al. [BGK+21b] gave an algorithm for solving unweighted single source shortest paths in time $\mathcal{O}(dn \log n)$ and hence unweighted all-pairs shortest paths in time $\mathcal{O}(dn^2 \log n)$ on graphs of twin-width at most d assuming an ordered union tree of a d-contraction sequence is given, which is also a compact way of representing a d-contraction sequence. To show this result, they proved that every graph of twin-width at most d admit a so-called interval biclique partition of size $\mathcal{O}(dn)$ that can be computed in time $\mathcal{O}(dn)$. Thus, one can solve unweighted APSP in time $\mathcal{O}(n^2 \log n)$ on bounded twin-width graph, assume a $\mathcal{O}(1)$-contraction sequence is given. In contrast, the related problem diameter in which one is interested in the longest shortest path in an unweighted and undirected graph, cannot be solved, or even $3/2-\varepsilon'$-approximate, in time $\mathcal{O}(n^{2-\varepsilon})$ on bounded twin-width graphs, even if a $\mathcal{O}(1)$-contraction sequence is given, for any $\varepsilon', \varepsilon > 0$.

Apart from this, there are no further efficient parameterized algorithms known for tractable problems when parameterized by the twin-width. In the realm of NP-hard problems, there exists algorithms for independent set, dominating set, and clique, all running in time $\mathcal{O}(2^{c \cdot d}n)$ for some constant c [BKTW20].

A natural possible direction of further research is to extend the set of problems that can be solved faster utilizing low twin-width of a graph, such as maximum matching, or vertex-weighted all-pairs shortest paths. Also for triangle counting, for which we presented an algorithm that is only adaptive when compared to the best combinatorial algorithms, an interesting open question is whether one can utilize fast matrix multiplication to lower the dependency on the twin-width in the running time, e.g., is it possible to solve triangle counting in time $\mathcal{O}(\mathsf{tww}^{\omega-1} n+m)$, which would be an optimal adaptive algorithm, matching the running time for triangle counting of $\mathcal{O}(n^\omega)$ by Alon et al. [AYZ97].

Finally, due to the local and simple structure of a triangle, the problem triangle counting seems to be a good toy problem to investigate if a given structure in a graph can be helpful to obtain faster (at best even adaptive) algorithms. Since we have seen a (combinatorially) adaptive algorithm for triangle counting parameterized by the twin-width of a graph, what is a most general parameter for which an adaptive algorithm for triangle counting exists?

Part III

Heterogeneous Graph Classes

6

Heterogeneous Structure: Combining Tree-Depth and Modular-Width

Many computational problems admit fast algorithms on special inputs, e.g., many graph problems can be solved much faster on interval graphs, planar graph, bipartite graphs, cographs, or on graphs where a specific parameter is small. In particular, parameterized algorithms are a great success, with examples presented in the previous chapters of this thesis, but there are also numerous examples in the literature for several parameters and different problems, see e.g. [FLS+18, MNN16, BN18, FKM+19, IOO18, CDP19, DP21b, Duc22b] for a small fraction of algorithms in the realm of "FPT in P" that leverage some kind of structure to obtain faster algorithm. This holds for both, tractable and intractable problems (while the research done for NP-hard problems is even more manifested and elaborated).

As a downside, the required structure is often not very general as well as many graph parameters are incomparable to one another, e.g., the class all subdivided stars is of bounded tree-depth though of unbounded modular-width, in contrast to the class of all cliques K_n that is of unbounded tree-depth but bounded modular-with; while for both parameters there are efficient parameterized algorithms for many problems. Thus, for a specific problem, often there just is no *most general* parameter for which this problem can be solved more efficiently, or there is one, but the dependency on the parameter is much worse than on smaller and incomparable parameters.

Considering the wealth of such algorithmic results, our motivation is to define graph classes generalizing two or more parameters while still being able to use the beneficial structure to obtain efficient algorithms. In this chapter, building on measures of the graph structure tree-depth and modular-width, we show how to robustly define *heterogeneous combinations* of such measures and how to (often optimally) use the corresponding graph structure for faster and more general algorithms. To this end, we adopt definitions based on graph operations and algebraic expressions such as are common for clique-width (cf. Chapter 4) and as were used by

Table 6.1: Overview of our algorithmic results, where n and m denote the number of vertices and edges of the input graph. NEGATIVE CYCLE DETECTION and VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS are on directed, vertex-weighted graphs with arbitrary vertex weights. The result for TRIANGLE COUNTING on MW_h was shown in Section 3.5, the result for NEGATIVE CYCLE DETECTION on TD_k was shown by Iwata et al. [IOO18].

Graph class	TRIANGLE COUNTING	NEGATIVE CYCLE DETECTION	V.-WEIGHT. APSP
MW_h	$\mathcal{O}(h^{\omega-1}n + m)$ [Ch. 3]	$\mathcal{O}(h^2n + n^2)$	$\mathcal{O}(h^2n + n^2)$
TD_k	$\mathcal{O}(km)$	$\mathcal{O}(k(m + n\log n))$ [IOO18]	$\mathcal{O}(kn^2)$
MTD_ℓ	$\mathcal{O}(\ell m)$	$\mathcal{O}(\ell(m + n\log n) + n^2)$	$\mathcal{O}(\ell n^2)$
TD_kMW_h	$\mathcal{O}(h^{\omega-1}n + km)$	$\mathcal{O}(h^2n + k(m + n\log n) + n^2)$	$\mathcal{O}(h^2n + kn^2)$
$\text{TD}_k\text{MTD}_\ell$	$\mathcal{O}((k + \ell)m)$	$\mathcal{O}((k + \ell)(m + n\log n) + n^2)$	$\mathcal{O}((k + \ell)n^2)$
$\text{MW}_h\text{MTD}_\ell$	$\mathcal{O}(h^{\omega-1}n + \ell m)$	$\mathcal{O}(h^2n + \ell(m + n\log n) + n^2)$	$\mathcal{O}(h^2n + \ell n^2)$
$\text{TD}_k\text{MW}_h\text{MTD}_\ell$	$\mathcal{O}(h^{\omega-1}n + (k + \ell)m)$	$\mathcal{O}(h^2n + (k + \ell)(m + n\log n) + n^2)$	$\mathcal{O}(h^2n + (k + \ell)n^2)$

Iwata et al. [IOO18] for tree-depth alone. By allowing richer sets of operations, stemming from different types of beneficial structure, we robustly define a large range of heterogeneous graph structure that admits faster algorithms than the general case.

Apart from algebraic expressions for clique-width and tree-depth, we also find motivation through modular tree-width, introduced by Paulusma et al. [PSS16], which is the tree-width of the graph after contracting all twin classes. In a similar (but more general) style we define modular tree-depth to extend modular-width by allowing substitutions into (possibly large) graphs of small tree-depth rather than graphs of small size.

Our conceptual contribution is a clean and robust way of formalizing classes of graphs with (possibly beneficial) heterogeneous structure by using an operations-based perspective on forms of homogeneous structure along with algebraic expressions. In the present work, these build on operations used for constructing graphs of small tree-depth, modular-width, or modular tree-depth, but the potential for encompassing a much greater variety of operations (and hence greater variety structure) is evident. We show formally, how the arising graph classes relate to one another, establishing for example that the combination of tree-depth and modular-width is incomparable to modular tree-depth. Similarly, already for bounded parameter values, the new forms of heterogeneous structure are incomparable to underlying forms of homogeneous structure.

On the algorithmic side, we extend the framework of Iwata et al. [IOO18], which applies to tree-depth via the operations union and addition of a vertex, to work for all required operations. Building on the work of the running-time framework defined in Section 3.3 for modular-width alone, we give a general running-time framework that simplifies the task of obtaining running times that match the known bounds for the included homogeneous cases. To show its applicability, we apply our framework to three example problems, namely TRIANGLE COUNTING, NEGATIVE CYCLE DETECTION, and VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS. For each problem, using our framework, we give algorithmic results relative to heterogeneous measures that throughout match the best running time known for the homogeneous case. See Table 6.1 for an overview of our results.

Overview. In Section 6.1 we recall operations-based definitions for the parameters tree-depth and modular-width. In Section 6.2 we introduce the parameter modular tree-depth and use algebraic expressions to define graph classes with heterogeneous structure. The general running-time framework for these heterogeneous classes is presented in Section 6.3 and the applications to TRIANGLE COUNTING, NEGATIVE CYCLE DETECTION, and VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS can be found in Section 6.4. The relations between the obtained graph classes and to other known parameters are explored in Section 6.5. We conclude in Section 6.6.

6.1 Graph Operations and Algebraic Expressions

Recall that for two graphs H and G, by $H[v \leftarrow G]$ we denote the substitution of G into $v \in V(H)$, i.e., the graph obtained by replacing v in H with the graph G and giving each of its vertices the same neighborhood as v had. This extends up to complete substitution into a t-vertex graph H, denoted $H[v_1 \leftarrow G_1, \ldots, v_t \leftarrow G_t]$.

Graph operations. These graph operations that we will define now will be used in algebraic expressions. The nullary operations $\circ$ and $\bullet$ return the empty graph respectively the graph with a single vertex. For $t \geq 2$, the t-ary operations $\textsc{Union}_t$ and $\textsc{Join}_t$ are defined by $\textsc{Union}_t(G_1, \ldots, G_t) := G_1 \dot\cup \ldots \dot\cup G_t$ and $\textsc{Join}_t(G_1, \ldots, G_t) := G_1 \bowtie \ldots \bowtie G_t$; we will usually omit the subscript t and allow these operations for all $t \geq 2$. For a vertex x and a set $E_x \subseteq \{\{x, v\} \mid v \in V\}$, the unary operation $\textsc{Inc}_{x,E_x}$ is defined by $\textsc{Inc}_{x,E_x}(G) = (V \cup \{x\}, E \cup E_x)$ for all graphs $G = (V, E)$ and a new vertex $x \notin V$. For a graph H with $V(H) = \{v_1, \ldots, v_t\}$, the t-ary operation $\textsc{Subst}_H$ is defined by $\textsc{Subst}_H(G_1, \ldots, G_t) := H[v_1 \leftarrow G_1, \ldots, v_t \leftarrow G_t]$, with G_i being graphs for $i \in [t]$.

Algebraic expressions. As is common especially for clique-width, we use algebraic expressions over certain sets of operations to describe the structure or the construction of graphs. We say that G has an algebraic expression if G is the result of evaluating the expression (possibly followed by a single renaming of vertices). For an algebraic expression σ, we denote by $val(\sigma)$ the resulting graph and by $tree(\sigma) = T^\sigma$ we denote the corresponding expression tree, i.e., a rooted tree in which each node corresponds to an operation of the expression that is applied to its children. We define the *nesting depth* of an operation in an expression tree as the maximum number of nodes in the expression tree corresponding to this operation that are on a root-to-leaf path. We denote the nesting depth of an operation in an algebraic expression as the nesting depth of this operation in the corresponding expression tree. We let the empty graph correspond to the empty expression, irrespective of the set of operations.

Tree-depth. There are many equivalent definitions for the tree-depth of a graph, e.g., as the minimum height of a rooted forest whose closure contains G as a subgraph or by a recursive definition based on connected components and vertex deletion, cf. Section 2.3 for a detailed definition. Here, we give an equivalent folklore definition via algebraic expressions.

Definition 6.1. The *tree-depth* of a graph G, denoted $\mathsf{td}(G)$, is the smallest $k \in \mathbb{N}$ such that G has an algebraic expression over $\{\circ, \textsc{Union}\} \cup \{\textsc{Inc}_{x,E_x}\}$ whose $\textsc{Inc}_{x,E_x}$ operations have nesting depth at most k. The class $\textsc{Td}_k$ contains all graphs of tree-depth at most k.

Modular-width. Like tree-depth, the modular-width has several concrete definitions, which unfortunately do not all agree on what graphs have modular-width zero, one, or two; they are equivalent for modular-width h, for all $h \geq 3$, so there is no asymptotic difference between them.[1] See also Section 3.1 for a detailed definition of modular-width. For the purpose of this chapter, we give a definition via algebraic expressions, which is a alight adaptation of a definition due to Gajarský et al. [GLO13]. Note that this definition is equivalent to the one in Section 3.1, e.g., MW_0 is the class of all cographs.

Definition 6.2 (adapted from [GLO13])**.** The *modular-width* of a graph G, denoted $\mathsf{mw}(G)$, is the smallest $h \in \mathbb{N}$ such that G has an algebraic expression over $\{\bullet, \textsc{Union}, \textsc{Join}\} \cup \{\textsc{Subst}_H \mid |H| \leq h\}$. The class MW_h contains all graphs of modular-width at most h.

An expression as defined in Definition 6.2 can be easily derived from a modular decomposition tree by traversing the tree from bottom to top. On the other side, the vertex set of a graph resulting from an operation $\textsc{Subst}_H$ with $|H| \leq h$ can be surely partitioned resp. decomposed back into at most h modules, i.e., one can also easily derive a modular decomposition tree of width at most h from any expression as defined in Definition 6.2. Thus, Definition 6.2 is indeed equivalent to the definition presented in Chapter 3.

6.2 Heterogeneous Structure

Here we use the introduced graph operations to define classes of graphs with heterogeneous structure that generalize both TD_k and MW_h. We will also use them to encompass and generalize modular tree-depth, which we will introduce in a moment. In the following section, we then show how to use such structure algorithmically (and optimally).

Definition 6.3. For $k, h \in \mathbb{N}$, the class TD_kMW_h contains all graphs G that have an algebraic expression over $\{\circ, \bullet, \textsc{Union}, \textsc{Join}\} \cup \{\textsc{Inc}_{x,E_x}\} \cup \{\textsc{Subst}_H \mid |H| \leq h\}$ whose $\textsc{Inc}_{x,E_x}$ operations have nesting depth at most k.

The following propositions follow directly from the definition of TD_kMW_h; similar relations are true for the other classes and we do not list all of them explicitly. The inequality in Proposition 6.5 holds since all cliques are contained in the class MW_0 and therefore in TD_kMW_0, but not in TD_k. The converse non-relations, even for bounded values of k and h, are showed in Section 6.5.

Proposition 6.4. $\text{TD}_k \subseteq \text{TD}_k\text{MW}_h$ *and* $\text{MW}_h \subseteq \text{TD}_k\text{MW}_h$.

Proposition 6.5. $\text{TD}_0\text{MW}_h = \text{MW}_h$ *but* $\text{TD}_k\text{MW}_0 \neq \text{TD}_k$

Modular tree-depth. Modular tree-width was introduced and studied in several recent papers [PSS16, Men16, LM17, Lam20].[2] In these works, the modular tree-width of a graph G is the smallest $\ell \in \mathbb{N}$ such that G can be constructed from a graph of tree-width at most ℓ by

[1] The only minor semantic difference in all definition is whether the class of cographs have modular-width zero, one, or two. Note that since there is no prime graph with less than four vertices, the class of graphs of modular-width at most three is equal to the class of graphs of modular-width at most two.

[2] The modular tree-width of a graph was first considered as a parameter for CNF formulas in several works [PSS16, Men16, LM17] and was defined as the tree-width after the contraction of modules of the incidence graph of the formula. On general graphs the modular tree-width was considered by Lampis [Lam20], defined as the tree-width of the graph obtained from a graph if one collapses each twin class into a single vertex.

substituting each vertex with an independent set or a clique of arbitrary size. In other words, the modular tree-width is the width of the graph after collapsing each *twin class* to a single vertex. Recall that two vertices are *twins* if they have the same sets of neighbors; clearly, this is an equivalence relation. The twin classes are the equivalence classes of the twin relation.

An analogous definition for modular tree-depth would entail substituting cliques and independent sets into a graph of tree-depth at most ℓ. In our definition of modular tree-depth we deviate from this style, by instead extending modular-width to allow substitution into pattern graphs H of arbitrary size but tree-depth at most ℓ. (We would similarly define (generalized) modular tree-width but we do not study it in this work.) To avoid confusion, we will use *restricted modular tree-depth* to refer to modular tree-depth defined in the above style. Note that in many of the mentioned applications, restricted and generalized modular tree-width coincide because the considered graphs have a modular partition whose modules are cliques and independent sets.

Definition 6.6 (modular tree-depth)**.** The *modular tree-depth* of a graph G, denoted $\mathsf{mtd}(G)$, is the smallest $\ell \in \mathbb{N}$ such that G has an algebraic expression over $\{\bullet, \textsc{Union}, \textsc{Join}\} \cup \{\textsc{Subst}_H \mid \mathsf{td}(H) \leq \ell\}$. The class $\textsc{Mtd}_\ell$ contains all graphs of modular tree-depth at most ℓ.

Further classes with heterogeneous structure. Now, we can define three natural combinations of $\textsc{Mtd}_\ell$ with the classes considered so far. Note that $\textsc{Mtd}_\ell$ subsumes $\textsc{Td}_k$ and $\textsc{Mw}_h$ for $\ell \geq k, h$ but, surprisingly perhaps, there is no $\ell \in \mathbb{N}$ such that it fully contains $\textsc{Td}_1\textsc{Mw}_0$. This also means that, e.g., $\textsc{Mw}_h\textsc{Mtd}_\ell \subseteq \textsc{Mtd}_\ell$ when $h \leq \ell$ but in general this is not the case. Intuitively, substitution into a pattern H of small tree-depth is algorithmically more costly than substitution into a small pattern H, hence the case $h > \ell$ is sensible. The relations between these graph classes are explored in Section 6.5.

Definition 6.7. For $k, h, \ell \in \mathbb{N}$ we define the three graph classes $\textsc{Td}_k\textsc{Mtd}_\ell$, $\textsc{Mw}_h\textsc{Mtd}_\ell$, and $\textsc{Td}_k\textsc{Mw}_h\textsc{Mtd}_\ell$ to contain all graphs that have an algebraic expression of the following type:

- $\textsc{Td}_k\textsc{Mtd}_\ell$: algebraic expressions over the set $\{\circ, \bullet, \textsc{Union}, \textsc{Join}\} \cup \{\textsc{Inc}_{x,E_x}\} \cup \{\textsc{Subst}_H \mid \mathsf{td}(H) \leq \ell\}$ of operations whose $\textsc{Inc}_{x,E_x}$ operations have nesting depth at most k
- $\textsc{Mw}_h\textsc{Mtd}_\ell$: algebraic expressions over the set $\{\bullet, \textsc{Union}, \textsc{Join}\} \cup \{\textsc{Subst}_H \mid |H| \leq h\} \cup \{\textsc{Subst}_H \mid \mathsf{td}(H) \leq \ell\}$ of operations
- $\textsc{Td}_k\textsc{Mw}_h\textsc{Mtd}_\ell$: algebraic expressions over the set $\{\circ, \bullet, \textsc{Union}, \textsc{Join}\} \cup \{\textsc{Inc}_{x,E_x}\} \cup \{\textsc{Subst}_H \mid |H| \leq h\} \cup \{\textsc{Subst}_H \mid \mathsf{td}(H) \leq \ell\}$ of operations whose $\textsc{Inc}_{x,E_x}$ operations have nesting depth at most k

Remark 6.8. *The sets of allowed operations are simply the unions of what is allowed in the homogeneous case, while* $\textsc{Inc}_{x,E_x}$ *operations keep their restriction on nesting depth.*

Remark 6.9. *In this work, we always implicitly assume that for the operations* $\{\textsc{Subst}_H \mid \mathsf{td}(H) \leq \ell\}$ *a tree-depth expression for* H *is given.*

This concludes the introduction of new graph classes with heterogeneous structure. We will now turn to showing how useful they are for designing more general algorithms for well-structured graphs. Fortunately, this turns out to be just as robust as the simpler case of the class $\textsc{Td}_k$, studied by Iwata et al. [IOO18], and comes down to designing a separate routine for each allowed operation.

6.3 Running Time Framework

In this section, we combine the running time framework provided in Section 3.3 for graphs parameterized by the modular-width with the divide-and-conquer framework proved by Iwata et al. [IOO18] for graphs parameterized by the tree-depth. Since we will focus on functions describing running times, we will restrict ourselves to functions $T\colon \mathbb{R}_{\geq 1} \to \mathbb{R}_{\geq 1}$ that are *superhomogeneous* [BS05], i.e., for all $\lambda \geq 1$ it holds that $\lambda \cdot T(n) \leq T(\lambda \cdot n)$, see also Definition 3.13. As shown in Lemma 3.14, it holds for all functions f that are superhomogeneous in the first component and monotonically increasing in the second component that

$$\max_{\substack{1\leq x\leq n\\ 1\leq y\leq m}} \frac{T(x,y)}{x} \leq \frac{T(n,m)}{n}. \tag{6.1}$$

Inspired by the functional way of the divide-and-conquer framework by Iwata et al. [IOO18], we extend this approach to cope with the operations defined in the previous sections.

Theorem 6.10. *Let G be a graph such that $G \in \textsc{Td}_k\textsc{Mw}_h\textsc{Mtd}_\ell$ with a given expression for some integers $k, h, \ell \in \mathbb{N}$ and let f be a function defined on subgraphs of G such that $f(\circ)$ and $f(\bullet)$ can be computed in constant time. Let further T_{Inc}, T_{Sub}, and T_{SubTd} be functions that are superhomogeneous in each component that bound the running times of the following algorithms A_{Inc}, A_{Sub}, and A_{SubTd}:*

- *$A_{\text{Inc}}(G', f(G'), x, E_x) \mapsto f(\textsc{Inc}_{x,E_x}(G'))$. Given a graph G', its value $f(G')$, a vertex $x \notin V(G')$, and $E_x \subseteq \{\{x,u\} \mid u \in V(G')\}$, this algorithm computes the value $f(\textsc{Inc}_{x,E_x}(G'))$ in time $T_{\text{Inc}}(|V(G') \cup \{x\}|, |E(G') \cup E_x|)$.*
- *$A_{\text{Sub}}(H, (G_1, f(G_1)), \ldots, (G_t, f(G_t))) \mapsto f(\textsc{Subst}_H(G_1, \ldots, G_t))$. Given a t-vertex pattern graph H and graphs G_i with their values $f(G_i)$ for $i \in [t]$, this algorithm computes the value $f(\textsc{Subst}_H(G_1, \ldots, G_t))$ in time $T_{\text{Sub}}(|V(H)|, |E(H)|) = T_{\text{Sub}}(t, |E(H)|)$.*
- *$A_{\text{SubTd}}(H, (G_1, f(G_1)), \ldots, (G_t, f(G_t))) \mapsto f(\textsc{Subst}_H(G_1, \ldots, G_t))$. Given a t-vertex pattern graph H and graphs G_i with their values $f(G_i)$, this algorithm computes the value $f(\textsc{Subst}_H(G_1, \ldots, G_t))$ in time $T_{\text{SubTd}}(|V(H)|, |E(H)|, \mathsf{td}(H))$.*

Then, one can compute $f(G)$ in total time $\mathcal{O}(kT_{\text{Inc}}(n,m) + \frac{n}{h}T_{\text{Sub}}(h,m) + T_{\text{SubTd}}(n,m,\ell))$.

Proof. Let σ be a corresponding algebraic expression of $G \in \textsc{Td}_k\textsc{Mw}_h\textsc{Mtd}_\ell$, i.e., $val(\sigma) = G$. Let $T^\sigma = tree(\sigma)$ be the corresponding expression tree. Clearly, one can replace each occurrence of an operation $\textsc{Union}_t$ or $\textsc{Join}_t$ in T^σ for some $t \geq 1$ by a sequence of $t-1$ operations $\textsc{Subst}_H$ with a pattern graph H consisting of two adjacent or non-adjacent vertices, i.e., $H \in \{I_2, K_2\}$. Thus, w.l.o.g. we can assume that σ does not consists of operations $\textsc{Union}_t$ nor $\textsc{Join}_t$. Furthermore, we can assume that each pattern H for a $\textsc{Subst}_H$ operation is of size at least two and that no argument of any $\textsc{Subst}_H$ operation is the empty graph. For a node $v \in V(T^\sigma)$, let T^σ_v denote the subtree of T^σ with root node v. With a slight abuse of notation, we denote by $val(T^\sigma_v)$ the subgraph of G corresponding to the subexpression tree T^σ_v.

To compute $f(G)$, we traverse the expression tree T^σ in a bottom-up manner and compute for each subexpression tree T^σ_v of T^σ the value $f(val(T^\sigma_v))$ for each $v \in V(T^\sigma)$. See Algorithm 4 for the divide-and-conquer algorithm that computes $f(G)$ by traversing T^σ. By induction over the length of the expression it is easy to see that Algorithm 4 computes the function $f(G)$ correctly.

Algorithm 4: Algorithm for computing $f(G)$

Input: Graph G with corresponding expression tree T^σ
Output: $f(G)$

if *G is the empty graph* **then**
  return $f(\circ)$
if *G is a single-vertex graph* **then**
  return $f(\bullet)$
Let r be the root node of T^σ and let $v_1, \ldots, v_c$ be the children of r in T^σ.
Let $G_1, \ldots, G_c$ be the graphs $val(T^\sigma_{v_1}), \ldots, val(T^\sigma_{v_c})$.
for $i \in [c]$ **do**
  $f(G_i) = Compute(G_i, T^\sigma_{v_i})$
Compute $f(G)$ via the algorithm corresponding to the root node r.
return $f(G)$

We are left to show the desired running time for Algorithm 4. Note that all leaves of T^σ correspond to either $\circ$ or $\bullet$ operations, while in our setting an operation $\circ$ is necessarily followed by an operation $\textsc{Inc}_{x,E_x}$. Thus, the number of leaves in T^σ is at most n, i.e., at most one for each vertex in $V(G)$; the remaining vertices come via additional $\textsc{Inc}_{x,E_x}$ operations. Since $f(\circ)$ and $f(\bullet)$ can be computed in constant time, the total time for processing all leaves is bounded by $\mathcal{O}(n)$.

All interior nodes of T^σ correspond to either to $\textsc{Inc}_{x,E_x}$ or $\textsc{Subst}_H$ operations. We denote by $V_{T^\sigma}^{\textsc{Inc}}$, $V_{T^\sigma}^{\textsc{Subst}}$, resp. $V_{T^\sigma}^{\textsc{SubstTd}}$ those nodes in $V(T^\sigma)$ that correspond to an operation $\textsc{Inc}_{x,E_x}$, $\textsc{Subst}_H$ with $|H| \leq h$, resp. $\textsc{Subst}_H$ with $\mathsf{td}(H) \leq \ell$. The total running time of the algorithm can now be bounded by the sum of the running times needed to process each node in T^σ.

For any node $v_H \in V_{T^\sigma}^{\textsc{Subst}} \cup V_{T^\sigma}^{\textsc{SubstTd}}$, let n_H and m_H denote the number of vertices resp. edges in the pattern graph H associated with v_H. Thus, a node $v_H \in V_{T^\sigma}^{\textsc{Subst}} \cup V_{T^\sigma}^{\textsc{SubstTd}}$ has exactly n_H children. Note that $n_H \geq 2$ for each pattern graph of a node v_H and since the number of leaves in T^σ is bounded by n, it holds that $|V_{T^\sigma}^{\textsc{Subst}} \cup V_{T^\sigma}^{\textsc{SubstTd}}| \leq n - 1$. Hence, the sum of the values n_H for all nodes $v_H \in V_{T^\sigma}^{\textsc{Subst}} \cup V_{T^\sigma}^{\textsc{SubstTd}}$ can be bounded by $2n$, i.e., the number of leaves plus $|V_{T^\sigma}^{\textsc{Subst}} \cup V_{T^\sigma}^{\textsc{SubstTd}}|$.

We can now bound the combined running time of all nodes in $V_{T^\sigma}^{\textsc{Subst}}$ corresponding to an operation $\textsc{Subst}_H$ with $|H| \leq h$ in a similar way as done in the proof of Theorem 3.15:

$$\begin{aligned}
\sum_{v_H \in V_{T^\sigma}^{\textsc{Subst}}} T_{\mathrm{Sub}}(n_H, m_H) &= \sum_{v_H \in V_{T^\sigma}^{\textsc{Subst}}} n_H \frac{T_{\mathrm{Sub}}(n_H, m_H)}{n_H} \\
&\leq \sum_{v_H \in V_{T^\sigma}^{\textsc{Subst}}} n_H \cdot \left(\max_{\substack{1 \leq n_H \leq h \\ 1 \leq m_H \leq m}} \frac{T_{\mathrm{Sub}}(n_H, m_H)}{n_H} \right) \\
&\leq 2n \cdot \frac{T_{\mathrm{Sub}}(h, m)}{h}
\end{aligned}$$

The final inequality holds due to Equation (6.1) and due to T_{Sub} being superhomogeneous. For all nodes in $V_{T^\sigma}^{\textsc{SubstTd}}$ corresponding to an operation $\textsc{Subst}_H$ with $\mathsf{td}(H) \leq \ell$ the running time can be bounded in a similar way:

$$\sum_{v_H \in V_{T^\sigma}^{\text{SubstTd}}} T_{\text{SubTd}}(n_H, m_H, \mathsf{td}(H)) \leq \sum_{v_H \in V_{T^\sigma}^{\text{SubstTd}}} n_H \frac{T_{\text{SubTd}}(n_H, m_H, \ell)}{n_H}$$
$$\leq \sum_{v_H \in V_{T^\sigma}^{\text{SubstTd}}} n_H \cdot \left(\max_{\substack{1 \leq n_H \leq n \\ 1 \leq m_H \leq m}} \frac{T_{\text{SubTd}}(n_H, m_H, \ell)}{n_H} \right)$$
$$\leq 2n \cdot \frac{T_{\text{SubTd}}(n, m, \ell)}{n} = 2 \cdot T_{\text{SubTd}}(n, m, \ell)$$

Finally, consider all nodes in $V_{T^\sigma}^{\text{Inc}}$. We define for each node $v \in V_{T^\sigma}^{\text{Inc}}$ a depth $d(v)$ equal to the maximum number of nodes in $V_{T^\sigma}^{\text{Inc}}$ on a path from v to a leaf in T_v^σ (including v), i.e., the nesting depth of the operation Inc_{x,E_x} in T_v^σ. It is easy to see that for two nodes $v, u \in V_{T^\sigma}^{\text{Inc}}$ with $d(v) = d(u)$ and $v \neq u$, the subtrees T_v^σ and T_u^σ are disjoint. Furthermore, the depth $d(v)$ of any node in $V_{T^\sigma}^{\text{Inc}}$ is per definition at most k. For a node $v \in V_{T^\sigma}^{\text{Inc}}$, let n_v resp. m_v denote the number of vertices resp. number of edges in $val(T_v^\sigma)$. We can now upper bound the running time of all nodes $v \in V_{T^\sigma}^{\text{Inc}}$:

$$\sum_{v \in V_{T^\sigma}^{\text{Inc}}} T_{\text{Inc}}(n_v, m_v) = \sum_{i=1}^{k} \sum_{\substack{v \in V_{T^\sigma}^{\text{Inc}} \\ d(v)=i}} T_{\text{Inc}}(n_v, m_v) \leq \sum_{i=1}^{k} T_{\text{Inc}}(n, m) \leq k \cdot T_{\text{Inc}}(n, m)$$

The penultimate inequality holds due to T_{Inc} being superhomogeneous (which implies that $T_{\text{Inc}} \in \Omega(n+m)$). In total, the running time sums up to $\mathcal{O}(kT_{\text{Inc}}(n, m) + \frac{n}{h} T_{\text{Sub}}(h, m) + T_{\text{SubTd}}(n, m, \ell))$ and we have proven Theorem 6.10. □

Theorem 6.10 describes the running time for graphs in the class $\text{TD}_k\text{MW}_h\text{MTD}_\ell$ if algorithms A_{Inc}, A_{Sub}, and A_{SubTd} are known. Similarly, one can state the running times for the other subclasses of $\text{TD}_k\text{MW}_h\text{MTD}_\ell$ that are defined in Section 6.2.

Corollary 6.11. *Let $G = (V, E)$ be a graph with $|V| = n$ and $|E| = m$. Let further f be a function defined on subgraphs of G and let T_{Inc}, T_{Sub}, resp. T_{SubTd} be as defined in Theorem 6.10. Then, depending on a given expression σ with $val(\sigma) = G$, one can compute $f(G)$ in the following time:*

	Total running time to compute $f(G)$	
$G \in \text{MW}_h$	$\mathcal{O}(\frac{n}{h} T_{\text{Sub}}(h, m))$	Theorem 3.15
$G \in \text{TD}_k$	$\mathcal{O}(kT_{\text{Inc}}(n, m))$	[IOO18]
$G \in \text{MTD}_\ell$	$\mathcal{O}(T_{\text{SubTd}}(n, m, \ell))$	
$G \in \text{TD}_k\text{MW}_h$	$\mathcal{O}(kT_{\text{Inc}}(n, m)) + \mathcal{O}(\frac{n}{h} T_{\text{Sub}}(h, m))$	
$G \in \text{TD}_k\text{MTD}_\ell$	$\mathcal{O}(kT_{\text{Inc}}(n, m)) + \mathcal{O}(T_{\text{SubTd}}(n, m, \ell))$	
$G \in \text{MW}_h\text{MTD}_\ell$	$\mathcal{O}(\frac{n}{h} T_{\text{Sub}}(h, m)) + \mathcal{O}(T_{\text{SubTd}}(n, m, \ell))$	

6.4 Applications

In this section, we give applications of Theorem 6.10. We present algorithms solving the problems TRIANGLE COUNTING, NEGATIVE CYCLE DETECTION, and VERTEX-WEIGHTED ALL-PAIRS

SHORTEST PATHS. For the latter two problems we restrict ourselves to vertex-weighted graphs, since for edge-weighted graphs the problems ALL-PAIRS SHORTEST PATHS and NEGATIVE CYCLE DETECTION are as hard as the general case already on cliques (as mentioned in Section 3.8), which are contained in $\text{TD}_0\text{MW}_0\text{MTD}_0$. For each problem, we will describe the algorithms A_{Inc}, A_{Sub}, and A_{SubTd} for some sensible function f, and prove the running time applying Theorem 6.10.

6.4.1 Triangle Counting

As a first and simple application of Theorem 6.10, we consider the problem TRIANGLE COUNTING. We will prove the following theorem:

Theorem 6.12. *Let G be an undirected graph such that $G \in \text{TD}_k\text{MW}_h\text{MTD}_\ell$ with a given expression for some $k, h, \ell \in \mathbb{N}$. Then one can solve* TRIANGLE COUNTING *in time $\mathcal{O}(h^{\omega-1}n + (k+\ell)m)$.*

For a graph G, we define $f(G)$ as the function that returns the values $n_G = |V(G)|$, $m_G = |E(G)|$, and the number of triangles, denoted by t_G.[3] To use Theorem 6.10, we will describe the algorithms A_{Inc}, A_{Sub}, and A_{SubTD} in the following three lemmata. The first lemma was already proven in Section 3.5.

Lemma 6.13. *Let H be a t-vertex graph, let $G_1, \ldots, G_t$ be graphs, and let $f(G_i)$ be given for each G_i. Then one can compute $f(\text{SUBST}_H(G_1, \ldots, G_t))$ in time $\mathcal{O}(t^\omega)$.*

Lemma 6.14. *Let G be a graph, let $x \notin V(G)$, let $E_x \subseteq \{\{x, u\} \mid u \in V(G)\}$, and let $f(G)$ be given. Then one can compute $f(\text{INC}_{x,E_x}(G))$ in time $\mathcal{O}(|E(G) \cup E_x|)$.*

Proof. Let n_G, m_G, and t_G be the values returned by $f(G)$. First, we compute the number of vertices and edges in linear time. In order to update the number of triangles, we initialize $t = t_G$, iterate over all edges in $\text{INC}_{x,E_x}(G)$, and check if both endpoints are adjacent to x. If so, we increment t by one. Since every triangle that is in $\text{INC}_{x,E_x}(G)$ but not in G needs to use the vertex x together with exactly one edge in $E(G)$, we have counted every triangle in time $\mathcal{O}(|E(G) \cup E_x|)$. □

Lemma 6.15. *Let H be a t-vertex graph, let $G_1, \ldots, G_t$ be graphs, and let $f(G_i)$ be given for each G_i. Then one can compute $f(\text{SUBST}_H(G_1, \ldots, G_t))$ in time $\mathcal{O}(\mathsf{td}(H) \cdot |E(H)|)$.*

Proof. Let $G = \text{SUBST}_H(G_1, \ldots, G_t)$ for a t-vertex graph H with $V(H) = \{v_1, \ldots, v_t\}$ and let n_i, m_i, and t_i be the values returned by $f(G_i)$. As shown in Section 3.5, for the number of vertices in G it holds that $n_G = \sum_{v_i \in V(H)} n_i$, for the number of edges in G it holds that $m_G = \sum_{v_i \in V(H)} m_i + \sum_{\{v_{i_1}, v_{i_2}\} \in E(H)} n_{i_1} \cdot n_{i_2}$, and finally $t_G = \sum_{v_i \in V(H)} t_i + \sum_{\{v_{i_1}, v_{i_2}\} \in E(H)} (m_{i_1} n_{i_2} + n_{i_1} m_{i_2}) + \sum_{\{v_{i_1}, v_{i_2}, v_{i_3}\} \in T(H)} n_{i_1} \cdot n_{i_2} \cdot n_{i_3}$, where $T(H)$ denotes the set of all triangles in H. Thus, n_G, m_G, and the first two summands of t_G can be computed in time $\mathcal{O}(|E(H)|)$. To compute the final summand of t_G, we use the tree-depth expression of H by adjusting the algorithm for INC_{x,E_x} (shown in Lemma 6.14), such that whenever we find a triangle $\{v_{i_1}, v_{i_2}, v_{i_3}\}$ we increment the number of triangles by $n_{i_1} \cdot n_{i_2} \cdot n_{i_3}$ (instead of by one). Using the tree-depth running-time framework by Iwata et al. [IOO18],[4] we have proven the lemma. □

[3] For the algorithm A_{Inc} the value t_G alone would suffice. The values n_G and m_G are only computed to potentially use the algorithms A_{Sub} and A_{SubTd}.

[4] The tree-depth running-time framework by Iwata et al. exactly corresponds to Theorem 6.10 applied only to a expression for TD_k.

Proof of Theorem 6.12. We use Theorem 6.10 to prove the claim. Lemma 6.14, Lemma 6.13, resp. Lemma 6.15 provide the algorithms A_{Inc}, A_{Sub}, resp. A_{SubTd}. Moreover, $f(\circ)$ and $f(\bullet)$ can be computed in constant time. Thus, by Theorem 6.10, the claim follows. □

6.4.2 Negative Cycle Detection

Before stating the algorithm for NEGATIVE CYCLE DETECTION and VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, we transfer the graph classes defined in Section 6.2 to directed graphs. To do so, we require each pattern graph H to be directed and we define the operation INC_{x,E_x} for an edge set $E_x \subseteq \{(x,u) \mid u \in V\} \cup \{(u,x) \mid u \in V\}$. In this section, we prove the following theorem.

Theorem 6.16. *Let $G = (V,E)$ be a directed graph such that $G \in \text{TD}_k\text{MW}_h\text{MTD}_\ell$ with a given expression for some $k,h,\ell \in \mathbb{N}$, and let $w\colon V \to \mathbb{R}$. Then one can solve* NEGATIVE CYCLE DETECTION *combinatorially in time $\mathcal{O}(h^2n + (k+\ell)(m + n\log n) + n^2)$ resp. in time $\mathcal{O}(h^{1.842}n + (k+\ell)(m + n\log n) + n^2)$ using fast matrix multiplication.*

First, we need to recall some notations about (feasible) potentials. For a directed graph $G = (V,E)$ with edge weights $c\colon E \to \mathbb{R}$ and a function $\pi\colon V \to \mathbb{R}$, we define $c_\pi\colon E \to \mathbb{R}$ with $c_\pi((x,y)) := c((x,y)) + \pi(x) - \pi(y)$ as the the *reduced costs* with respect to c and π.

Observation 6.17. *Let $G = (V,E)$ be a directed graph with edge weights $c : E \to \mathbb{R}$, let $\pi\colon V \to \mathbb{R}$ be a function defined on the vertices of G, and let P be an u-v path in G for $u,v \in V$. Then $c_\pi(P) = c(P) + \pi(u) - \pi(v)$.*

If for a function $\pi\colon V \to \mathbb{R}$ it holds that $c_\pi(e) \geq 0$ for all $e \in E$, we call π a *feasible potential* for c. It is well known that an edge-weighted graph G has no negative cycle if and only if G admits a feasible potential, see e.g. [Sch02]. To compute a feasible potential for a graph without a negative cycle, one can use the following lemma.

Lemma 6.18 ([KVKV11])**.** *Let $G = (V,E)$ be a directed graph that has no negative cycles relative to edge weights $c\colon E \to \mathbb{R}$. Let $x \notin V$ be a new vertex that is connected to all other vertices with edges of weight zero. Then $\pi\colon V \to \mathbb{R}$ with $\pi(v) = dist_{G,c}(x,v)$ is a feasible potential.*

We call a feasible potential computed as in Lemma 6.18 a *shortest-path feasible potential*. To define a feasible potential also for vertex-weighted graphs, we consider the edge-shifted weights instead, i.e., where the vertex weight of a vertex is shifted to all outgoing edges.

Definition 6.19. Let $G = (V,E)$ be a directed graph with vertex weights $w\colon V \to \mathbb{R}$. For any edge $(x,y) \in E$, we define the *edge-shifted weights* by $c^w((x,y)) := w(x)$.

Note that for a vertex-weighted graph G with vertex weights $w\colon V(G) \to \mathbb{R}$, a path P is a shortest path between two vertices $u,v \in V(G)$ w.r.t. the vertex weights w if and only if it is a shortest path w.r.t. the edge-shifted weights c^w.

Observation 6.20. *Let $G = (V,E)$ be a directed graph with vertex weights $w\colon V \to \mathbb{R}$, let P be a u-v path in G for $u,v \in V$, and let C be a cycle in G. Then $c^w(P) = w(P) - w(v)$ and $c^w(C) = w(C)$.*

In the following, if we speak about a potential for vertex-weighted graphs we implicitly consider the edge-shifted weights. Alongside to a feasible potential we will compute the value of a minimum shortest path of a subgraph in the algorithm, which we define next. Note that we consider a single vertex as a path.

Definition 6.21. Let $G = (V, E)$ be a graph and let $w\colon V \to \mathbb{R}$ be vertex weights. We define the *minimum shortest path* value as $\texttt{msp}(G) = \min_{P \in \mathcal{P}} w(P)$ where $\mathcal{P}$ denotes the set of all paths (including single vertices) in G.

In a slight abuse of notation we also use G to stand for a weighted, directed graph, consisting of a directed graph (V, E) and vertex weights $w\colon V \to \mathbb{R}$. For such a vertex-weighted, directed graph G, we define $f(G)$ as the function that either returns a negative cycle in G or that returns a shortest-path feasible potential and the minimum shortest path value $\texttt{msp}(G)$. (Formally $f(G) := f(G, w)$ and $\texttt{msp}(G) := \texttt{msp}(G, w)$.)

Increment. Let $G = (V, E)$ be a directed graph and let $w\colon V \to \mathbb{R}$ be vertex weights. Note that one cannot use Dijkstra's algorithm directly to compute a shortest-path feasible potential since there may be negative weights. However, in [IOO18] it was shown how to compute a feasible potential for $\textsc{Inc}_{x,E_x}(G)$ in time $\mathcal{O}(m + n \log n)$ that is not necessarily a shortest-path feasible potential. This can be used to compute $\texttt{msp}(\textsc{Inc}_{x,E_x}(G))$ and a shortest-path feasible potential in the same running time.

Lemma 6.22. *Let $G = (V, E)$ be a directed graph, let $x \notin V$, let $E_x \subseteq \{(x, u) \mid u \in V\} \cup \{(u, x) \mid u \in V\}$, let $w\colon V \cup \{x\} \to \mathbb{R}$, and let $f(G)$ be given. Then one can compute $f(\textsc{Inc}_{x,E_x}(G))$ in time $\mathcal{O}(m + n \log n)$ with $n = |V|$ and $m = |E \cup E_x|$.*

Proof. If $f(G)$ returns a negative cycle, we do so for $f(\textsc{Inc}_{x,E_x}(G))$. Let otherwise c^w be the edge-shifted weights in G. In [IOO18, Theorem 4], it was shown how to compute a feasible potential π w.r.t. c^w for $\textsc{Inc}_{x,E_x}(G)$ in time $\mathcal{O}(m + n \log n)$ if a feasible potential for G is known. Let $c^w_\pi\colon E \cup E_x \to \mathbb{R}$ be the reduced cost with respect to c^w and π. Using Dijkstra's algorithm, one can compute the values $dist_{c^w_\pi}(x, v)$ for all $v \in V$ in the same running time and, by reversing all edge directions, also the values $dist_{c^w_\pi}(v, x)$. By Observation 6.17 and Observation 6.20, we can then reconstruct the distances w.r.t. the edge weights c^w and finally w.r.t. the original vertex weights w in linear time.

A minimum shortest path in $\textsc{Inc}_{x,E_x}(G)$ either does use the new vertex x or it does not, thus, it holds that $\texttt{msp}(\textsc{Inc}_{x,E_x}(G))$ is the minimum of the two values $\min_{v \in V} dist_w(v, x) + \min_{u \in V} dist_w(x, u) - w(x)$ and $\texttt{msp}(G)$, and can be determined in linear time.

Finally, the feasible potential π that is computed in [IOO18] does not need to be a shortest-path feasible potential, however, using the corresponding non-negative edge weights c^w_π and Lemma 6.18, the problem of computing a shortest-path feasible potential reduces to the problem of computing a shortest path to all vertices from a new vertex that is connected to every vertex with an edge of weight zero in a graph with *non-negative* edge weights. This can be done, using Dijkstra's algorithm once again, in time $\mathcal{O}(m + n \log n)$. □

Substitution. Let H be an unweighted, directed t-vertex graph and let G_i be vertex-weighted, directed graphs for $i \in [t]$. If the vertex weights are non-negative, we know by Lemma 3.17 that for $u, v \in V(\textsc{Subst}_H(G_1, \ldots, G_t))$ with $\{u, v\} \nsubseteq V(G_i)$ for all $i \in [t]$ it holds that there exists always a shortest u-v path that visits each module at most once. The following lemma generalizes Lemma 3.17 to arbitrary vertex-weighted and directed graphs.

Lemma 6.23. *Let H be an unweighted, directed t-vertex graph and let G_i be vertex-weighted, directed graphs such that $\textsc{Subst}_H(G_1, \ldots, G_t)$ does not contain a negative cycle. Let further $u, v \in V(\textsc{Subst}_H(G_1, \ldots, G_t))$ such that there exists a shortest u-v path and $\{u, v\} \nsubseteq V(G_i)$ for all $i \in [t]$. Then there exists a shortest u-v path P in $\textsc{Subst}_H(G_1, \ldots, G_t)$ in which the occurrences of vertices in each G_i occur consecutively.*

Proof. Denote $G = \textsc{Subst}_H(G_1, \dots, G_t)$ and let $P = (u = p_1, p_2, \dots, p_n = v)$ be a shortest u-v path in G. Assume for contradiction that there exists indices $\alpha, \alpha', \beta, \gamma, \gamma' \in [n]$ with $\alpha \leq \alpha' < \beta < \gamma \leq \gamma'$ such that $p_j \in V(G_i)$ for all $\alpha \leq j \leq \alpha'$ and $\gamma \leq j \leq \gamma'$, but $p_\beta \notin V(G_i)$ for some $i \in [t]$. We distinguish between the two cases $\alpha \neq 1$ and $\alpha = 1$.

If $\alpha \neq 1$ it holds that $(p_{\alpha-1}, p_\alpha) \in E(G)$. Since $V(G_i)$ is a module in G it also holds that $(p_{\alpha-1}, p_\gamma) \in E(G)$. Thus, consider the path $P' = (p_1, \dots, p_{\alpha-1}, p_\gamma, p_{\gamma+1}, \dots, p_n)$ and denote by $C' = (p_\alpha, \dots, p_{\gamma-1})$ the skipped path. It holds that $\omega(P) = \omega(P') + \omega(C')$. Since $V(G_i)$ is a module in G with $p_\alpha, p_\gamma \in V(G_i)$ and $(p_{\gamma-1}, p_\gamma) \in E(G)$, it holds that also $(p_{\gamma-1}, p_\alpha) \in E(G)$. Thus, C' is a cycle and it holds that $\omega(C') \geq 0$, since G has no negative cycle. Hence, it follows that $\omega(P') \leq \omega(P)$.

If $\alpha = 1$ it holds that $\gamma' \neq n$, since u and v are in different modules, and therefore $(p_\gamma, p_{\gamma+1}) \in E(G)$. Since $V(G_i)$ is a module, it also holds that $(p_{\alpha'}, p_{\gamma+1}) \in E(G)$. Thus, consider the path $P'' = (p_1, \dots, p_{\alpha'}, p_{\gamma'+1}, \dots, p_n)$ and denote by $C'' = (p_{\alpha'+1}, \dots, p_\gamma)$ the skipped path. It holds thats $w(P) = \omega(P'') + \omega(C'')$. Since $V(G_i)$ is a module in G with $p_{\alpha'}, p_{\gamma'} \in V(G_i)$ and $(p_{\alpha'}, p_{\alpha'+1}) \in E(G)$, it holds that also $(p_\gamma, p_{\alpha'+1}) \in E(G)$. Thus, C'' is a cycle and it holds that $\omega(C'') \geq 0$, yielding again that $\omega(P'') \leq \omega(P)$.

We iterate this procedure for every pair of maximal sequences of vertices that are in a same module but interrupted by at least one vertex of a different module. Since the vertices in P' resp. P'' are a strict subset of the vertices in P, the number of vertices in the u-v path strictly reduces each time. □

Lemma 6.23 motivates the following definition.

Definition 6.24. Let H be an unweighted, directed graph with $V(H) = \{v_1, \dots, v_t\}$ and let G_i be vertex-weighted, directed graphs for $i \in [t]$. Define H_w as the graph H with vertex weights $w \colon V(H) \to \mathbb{R}$ defined by $w(v_i) = \mathtt{msp}(G_i)$.

Let H_w be the graph defined in Definition 6.24 for a t-vertex graph H and weighted graphs G_i for $i \in [t]$. The next lemma shows that there is a negative cycle in $\textsc{Subst}_H(G_1, \dots, G_t)$ if and only if there is a negative cycle in H_w and that the minimum shortest path value in $\textsc{Subst}_H(G_1, \dots, G_t)$ and H_w coincide.

Lemma 6.25. *Let H be an unweighted, directed t-vertex graph, let G_i be vertex-weighted, directed graphs without negative cycles, and let H_w be the graph as defined in Definition 6.24. Then there exists a negative cycle in $\textsc{Subst}_H(G_1, \dots, G_t)$ if and only if there exists a negative cycle in H_w. Moreover, if $\textsc{Subst}_H(G_1, \dots, G_t)$ does not admit a negative cycle, $\mathtt{msp}(H_w) = \mathtt{msp}(\textsc{Subst}_H(G_1, \dots, G_t))$.*

Proof. Assume that $\textsc{Subst}_H(G_1, \dots, G_t)$ has a negative cycle. Let C be a negative cycle in $\textsc{Subst}_H(G_1, \dots, G_t)$ of minimum weight. Since no G_i contains a negative cycle, C cannot be completely inside a single G_i. Moreover, because C is chosen as a cycle of minimum weight, each subpath of C is a shortest path and thus, by Lemma 6.23, C enters each G_i at most once. Denote by P_i the maximal subpaths of C with vertices in $V(G_i)$. Clearly, $w(P_i) \geq \mathtt{msp}(G_i)$, and in fact it even holds that $w(P_i) = \mathtt{msp}(G_i)$, since otherwise we could replace P_i by the path P_i' with $w(P_i') = \mathtt{msp}(G_i)$, because $V(G_i)$ is a module in $\textsc{Subst}_H(G_1, \dots, G_t)$, which contradicts the choice of C as a cycle of minimum weight. Thus, there is a corresponding cycle C' in H_w with $w(C) = w(C')$. Conversely, each cycle in H_w directly corresponds to a cycle in $\textsc{Subst}_H(G_1, \dots, G_t)$ of the same length by replacing each vertex by a corresponding minimum shortest path, hence there is a negative cycle in $\textsc{Subst}_H(G_1, \dots, G_t)$ if and only if there is one in H_w.

We are left to show $\texttt{msp}(H_w) = \texttt{msp}(\textsc{Subst}_H(G_1, \dots, G_t))$ if $\textsc{Subst}_H(G_1, \dots, G_t)$ does not contain a negative cycle. Clearly, $\texttt{msp}(\textsc{Subst}_H(G_1, \dots, G_t)) \leq \texttt{msp}(H_w)$, since every path in H_w corresponds to a path in $\textsc{Subst}_H(G_1, \dots, G_t)$ of the same length. For the other direction let P be a path in $\textsc{Subst}_H(G_1, \dots, G_t)$ with $w(P) = \texttt{msp}(\textsc{Subst}_H(G_1, \dots, G_t))$. We distinguish three cases: (1) $V(P) \subseteq V(G_i)$ for some $i \in [t]$. Then, the corresponding vertex $v_i \in V(H_w)$ has weight at most $w(P)$ and thus $w(P) \geq \texttt{msp}(G_i) \geq \texttt{msp}(H_w)$. (2) P starts and ends in the same vertex set $V(G_i)$ for some $i \in [t]$. Let P' be the first part of P that is completely in $V(G_i)$ and let P'' be the remainder of P. Then it holds that $w(P'') \geq 0$ since otherwise there would be a negative cycle in $\textsc{Subst}_H(G_1, \dots, G_t)$ on the same vertex set of P'' ($V(G_i)$ is a module). Thus, $w(P) = w(P') + w(P'') \geq w(P') \geq \texttt{msp}(G_i) \geq \texttt{msp}(H_w)$. (3) P starts in $V(G_i)$ and ends in $V(G_j)$ for $i, j \in [t]$ with $i \neq j$. Then, by Lemma 6.23, P visits each G_i at most once. Let P^* be the path that replaces each maximal subpath of P in some G_i by the minimum shortest path in G_i. Now, it holds that $w(P) \geq w(P^*)$ and since there is a corresponding path P' in H_w with $w(P') = w(P^*)$ we have shown that $w(P) \geq w(P^*) \geq \texttt{msp}(H_w)$. □

We can now prove the following lemma.

Lemma 6.26. *Let H be an unweighted, directed t-vertex graph, let G_i be vertex-weighted, directed graphs, and let $f(G_i)$ be given for each G_i. Then one can compute $f(\textsc{Subst}_H(G_1, \dots, G_t))$ combinatorially in time $\mathcal{O}(t^3 + n)$ or in time $\mathcal{O}(t^{2.842} + n)$ using fast matrix multiplication.*

Proof. If any $f(G_i)$ indicates a negative cycle, we report this negative cycle also for the graph $\textsc{Subst}_H(G_1, \dots, G_t)$. Let otherwise H_w be the vertex-weighted graph as defined in Definition 6.24, let π_i be a shortest-path feasible potential for G_i, and let $\texttt{msp}(G_i)$ be the minimum shortest path value of G_i for $i \in [t]$.

By Lemma 6.25, it suffices to solve VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS on H_w to check if there is a negative cycle in $\textsc{Subst}_H(G_1, \dots, G_t)$ and (if not) to compute the minimum shortest path value of $\textsc{Subst}_H(G_1, \dots, G_t)$. This can be done in time $\mathcal{O}(t^{2.842})$ by the algorithm of Yuster [Yus09] resp. combinatorially in time $\mathcal{O}(t^3)$ using a vertex-weighted variant of the algorithm by Floyd and Warshall [Flo62, War62].

We are left to compute a shortest-path feasible potential for $\textsc{Subst}_H(G_1, \dots, G_t)$. To do so, we add a new vertex x to H_w of weight zero with directed arcs to all other vertices of H_w. This corresponds to adding a new vertex x' to the graph $\textsc{Subst}_H(G_1, \dots, G_t)$ with directed arcs to all other vertices of $\textsc{Subst}_H(G_1, \dots, G_t)$. Now, we compute a shortest-path feasible potential (regarding the edge-shifted weights) π_H for H_w in time $\mathcal{O}(t^{2.842})$ resp. $\mathcal{O}(t^3)$. We define $\pi \colon V(\textsc{Subst}_H(G_1, \dots, G_t)) \to \mathbb{R}$ by $\pi(v) = \pi_i(v) + \pi_H(v_i)$ for $v \in G_i$, and claim that π is a shortest-path feasible potential for $\textsc{Subst}_H(G_1, \dots, G_t)$.

To prove the claim, we show that for any vertex $v \in G_i$ for some $i \in [t]$, a shortest x'-v path (regarding the edge-shifted weights) has weight $\pi_i(v) + \pi_H(v_i)$. Since x' is connected to every vertex in $V(\textsc{Subst}_H(G_1, \dots, G_t))$, the singleton set $\{x'\}$ forms a module and thus we know by Lemma 6.23 that there exists a shortest x'-v path that does not enter any G_i twice. Hence, a shortest path from x' to v can be split into a shortest path to reach $V(G_i)$ and a shortest path from some vertex in G_i to v. The latter part is exactly $\pi_i(v)$ and the former part can be constructed by using a minimum shortest path in each G_j for $j \in [t]$, i.e., is equal to $\pi_H(v_i)$. Consequently, after we have computed a shortest-path feasible potential π_H for H_w, we can compute a shortest-path feasible potential π for all vertices in $V(\textsc{Subst}_H(G_1, \dots, G_t))$ in total time $\mathcal{O}(n)$, which proves the lemma. □

Lemma 6.27. *Let H be an unweighted, directed t-vertex graph, let G_i be vertex-weighted, directed graphs, and let $f(G_i)$ be given for each G_i. Then one can compute $f(\textsc{Subst}_H(G_1,..,G_t))$ in time $\mathcal{O}(\mathtt{td}(H)\cdot(|E(H)|+t\log t)+n)$ with $n=|\textsc{Subst}_H(G_1,..,G_t))|$.*

Proof. If any $f(G_i)$ indicates a negative cycle, we report this negative cycle also for the graph $\textsc{Subst}_H(G_1,\dots,G_t)$. Otherwise, by Lemma 6.25, $\textsc{Subst}_H(G_1,\dots,G_t)$ has a negative cycle if and only if H_w does and $\mathtt{msp}(\textsc{Subst}_H(G_1,\dots,G_t))=\mathtt{msp}(H_w)$. Moreover, as seen in the proof of Lemma 6.26, one can compute a shortest-path feasible potential for $\textsc{Subst}_H(G_1,\dots,G_t)$ in linear time after we have computed a shortest-path feasible potential for H_w. Thus, we can exploit the tree-depth expression of H_w by using Lemma 6.22. Using the tree-depth running-time framework by Iwata et al. [IOO18], we have proven the lemma. □

Proof of Theorem 6.16. Let σ be the given expression with $val(\sigma)=G$. We use Theorem 6.10 to prove the claim. Lemma 6.22, Lemma 6.26, resp. Lemma 6.27 provide the algorithms A_{Inc}, A_{Sub}, resp. A_{SubTd}. Since $|T^\sigma|\leq n$, the linear portions of the running time of A_{Sub} and A_{SubTd} sum up to $\mathcal{O}(n^2)$ time. Moreover, $f(\circ)$ and $f(\bullet)$ can be computed in constant time. Thus, by Theorem 6.10, the claim follows. □

We emphasize that for the algorithm A_{Inc} only a feasible potential is needed and for the algorithm A_{Sub} the value of a minimum shortest path alone would suffice, but in the heterogeneous case we need to compute both in all algorithms. We consider a *shortest-path* feasible potential to ease the computation of a (shortest-path) feasible potential in A_{Sub} and A_{SubTD}.

6.4.3 Vertex-Weighted All-Pairs Shortest Paths

In this section we will extend the algorithm of Section 6.4.2 and compute for all pairs of vertices the shortest-path distance. We will prove the following theorem:

Theorem 6.28. *Let $G=(V,E)$ be a directed graph such that $G\in \textsc{Td}_k\textsc{Mw}_h\textsc{Mtd}_\ell$ with a given expression for some $k,h,\ell\in\mathbb{N}$, and let $w\colon V\to\mathbb{R}$. Then one can either conclude that G contains a negative cycle or one can solve* VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS combinatorially in time $\mathcal{O}(h^2n+(k+\ell)n^2)$ resp. in time $\mathcal{O}(h^{1.842}n+(k+\ell)n^2)$ using fast matrix multiplication.

We will compute slightly different values $f(G)$ depending on the operation. Most notably, we will compute the shortest-path values between all pair of vertices only before and after an operation $\textsc{Inc}_{x,E_x}$, whereas for operations $\textsc{Subst}_H$ we restrict the computation to auxiliary values between modules and single vertices, until the next $\textsc{Inc}_{x,E_x}$ operation appears or the expression is fully processed.

Substitution. For a directed, unweighted t-vertex graph H with a vertex set $V(H)=\{v_1,\dots,v_t\}$ and directed, vertex-weighted graphs G_i, let H_w be the graph defined in Definition 6.24 and let $G=\textsc{Subst}_H(G_1,\dots,G_t)$.

For $u\in V(G_i)$ and $v\in V(G_j)$ with $i\neq j$, a shortest u-v path in G will consists of a shortest path from u to some vertex in $V(G_i)$ (possibly only u), followed by a shortest path from v_i to v_j in H_w (i.e., using the minimum shortest path in all intermediate modules[5]), completed by a shortest path from some vertex in $V(G_j)$ to v (possibly only v). This motivates the following definition: We define the function $f_S(G)$ as the function that returns the same values

[5]Hence, we do not consider the weights of the start- and endvertex.

Values returned by $f_S(G)$	Values returned by $f_I(G)$
• shortest-path feasible potential π	• shortest-path feasible potential π
• $\texttt{msp}(G) = \min_{u,v \in V(G)} dist_G(u,v)$	• $\texttt{msp}(G) = \min_{u,v \in V(G)} dist_G(u,v)$
• $\min_{v \in V(G)} dist_G(u,v)$ for all $u \in V(G)$	• $\min_{v \in V(G)} dist_G(u,v)$ for all $u \in V(G)$
• $\min_{v \in V(G)} dist_G(v,u)$ for all $u \in V(G)$	• $\min_{v \in V(G)} dist_G(v,u)$ for all $u \in V(G)$
• $dist_{H_w}(v_i, v_j)$ for all $v_i, v_j \in V(H_w)$	• $dist_G(u,v)$ for all $u, v \in V(G)$

Table 6.2: An overview of the values returned by the functions f_S resp. f_I for the problem VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS. All but the last bullet are identical. For the problem NEGATIVE CYCLE DETECTION the first two bullets are sufficient.

that we have computed in Section 6.4.2, i.e., a shortest-path feasible potential and the value $\texttt{msp}(G)$ of a minimum shortest path in G. Additionally, $f_S(G)$ returns for each pair of vertices $v_i, v_j \in V(H_w)$ the length of a shortest v_i-v_j path in H_w for $i, j \in [t]$, and for each vertex $u \in V(G)$ the values $\min_{v \in V(G)} dist_G(u,v)$ and $\min_{v \in V(G)} dist_G(v,u)$. See also the left side of Table 6.2 for the list of values returned by the function f_S.

Lemma 6.29. *Let H be an unweighted, directed t-vertex graph, let G_i be vertex-weighted, directed graphs, and let $f_S(G_i)$ be given for each G_i. Then one can compute $f_S(\textsc{Subst}_H(G_1, .., G_t))$ in $\mathcal{O}(t^3 + n)$ combinatorially time or $\mathcal{O}(t^{2.842} + n)$ time using fast matrix multiplication.*

Proof. If any $f(G_i)$ indicates a negative cycle, we report this negative cycle also for the graph $\textsc{Subst}_H(G_1, .., G_t)$. Otherwise, let H_w be as defined in Definition 6.24 and denote $G = \textsc{Subst}_H(G_1, \ldots, G_t)$. First of all, we solve the vertex-weighted all-pairs shortest paths problem on H_w, check for a negative cycle, compute a shortest-path feasible potential, and compute the minimum shortest path value $\texttt{msp}(G)$ as done in Section 6.4.2 for NEGATIVE CYCLE DETECTION. We are left to compute for each vertex $u \in V(G)$ the values $\min_{v \in V(G)} dist_G(u,v)$ and $\min_{v \in V(G)} dist_G(v,u)$.

Let $i \in [t]$ such that $u \in V(G_i)$. If $\arg\min_{v \in V(G)} dist_G(u,v) \notin V(G_i)$ we know due to Lemma 6.23 that there exists a shortest u-v path in G in which the occurrences of vertices in each G_i occur consecutively. Thus, in this case it holds that

$$\min_{v \in V(G)} dist_G(u,v) = \min_{u' \in V(G_i)} dist(u,u') + \min_{v_j \in V(H_w)} dist_{H_w}(v_i, v_j) - w(v_i). \tag{6.2}$$

If $\arg\min_{v \in V(G)} dist_G(u,v) \in V(G_i)$, we observe that in this case it holds that the length of a shortest u-v path is equal to $\min_{u' \in V(G_i)} dist(u,u')$; otherwise there would be a negative cycle in G, since w.l.o.g. one can assume that each shortest u-v path does start with a path in $V(G_i)$ of length $\min_{u' \in V(G_i)} dist(u,u')$. Thus, Equation (6.2) does hold in general. To compute the value in Equation (6.2) for all $u \in V(G)$, we first determine for each $v_i \in V(H_w)$ the value $\min_{v_j \in V(H_w)} dist_{H_w}(v_i, v_j)$ in time $\mathcal{O}(t^2)$ and store them. Since the value of $\min_{u' \in V(G_i)} dist(u,u')$ is known by $f(G_i)$, we can compute the value in Equation (6.2) for each $u \in V(G)$ in constant time.

The value $\min_{v \in V(G)} dist_G(v,u)$ can be computed analogously by executing the whole algorithm at any time also for the edge-flipped graph. □

Increment. For the operation $\textsc{Inc}_{x,E_x}$, we define $f_I(G)$ as a function that returns the same values as $f_S(G)$ but instead of the pairwise distances in the graph H_w it returns the pairwise

distance for all pairs $u, v \in V(G)$. See also Table 6.2 for an summary of the values returned by the function f_I. We compute $f_I(G)$ before and after the INC-operation.

Recall that for an expression σ, as defined in Definition 6.7, we denote by T^σ the corresponding expression tree and for a node $r \in V(T^\sigma)$ we denote by T^σ_r the subexpression tree of T^σ with root r. Further, we will denote by $G^\sigma_r = val(T^\sigma_r)$ the resulting graph after evaluating T^σ_r. Consider a node $r \in V(T^\sigma)$ labeled according to an operation SUBST_H whose parent node is labeled according to an operation INC_{x,E_x}.

To compute $f_I(G^\sigma_r)$ we will consider the maximal subtree of T^σ_r that only admits nodes labeled according to an operation SUBST_H. Considering this subexpression tree, one can prove in a similar way as done in Section 3.4 the following lemma:

Lemma 6.30. *Let T^σ be an expression tree of an expression σ as defined in Definition 6.7, let $r \in V(T^\sigma)$ labeled according to an operation SUBST_H whose parent node is labeled according to an operation INC_{x,E_x}, and let G^σ_r be the resulting graph after evaluating T^σ_r. Let further the function f_S resp. f_I be known for all graphs corresponding to nodes in $V(T^\sigma_r)$ labeled according to an operation SUBST_H resp. INC_{x,E_x}. Then, one can compute $f_I(G^\sigma_r)$ in time $\mathcal{O}(n^2)$ with $n = |V(G^\sigma_r)|$.*

Proof. Since $f_S(G^\sigma_r)$ is known, we are left to compute $dist_{G^\sigma_r}(u, v)$ for all $u, v \in V(G^\sigma_r)$ in order to compute $f_I(G^\sigma_r)$. For this, we traverse the expression tree of T^σ_r downwards as long as one encounters a node that is not labeled according to an operation SUBST_H. W.l.o.g. we can assume that such a node is either labeled according to an operation INC_{x,E_x} or $\bullet$ (a single vertex), since one can assume that no argument of an operation SUBST_H is the empty graph and one can replace the operations UNION and JOIN by operations SUBST_H with pattern graphs of size two, as seen in the proof of Theorem 6.10. We denote the vertex set of this part of the expression tree (including the first nodes that are not labeled according to an operation SUBST_H) by S_r. Note that for a vertex $x \in S_r$ the corresponding vertex set $V(G^\sigma_x)$ forms a module in G^σ_r. We will determine for each such G^σ_x the value of a shortest u-v path in G^σ_r only using vertices in $V(G^\sigma_r) \setminus V(G^\sigma_x)$ with the property that u and v are adjacent to (all vertices of) $V(G^\sigma_x)$. We denote this value by c_x and we set $c_r = \infty$.

Assume for now that one has computed the values c_x for all $x \in S_r$ in time $\mathcal{O}(n^2)$ for $n = |V(G^\sigma_r)|$. Consider a node $x \in S_r$ labeled according to an operation SUBST_H and let $G^\sigma_x = \text{SUBST}_H(G_1, \ldots, G_t)$ be the corresponding graph with each G_i corresponds to a child of x. Let H_w be the graph defined in Definition 6.24.

For two vertices u, v with $u \in V(G_i)$ and $v \in V(G_j)$ for $i \neq j$, it holds that a shortest u-v path in G^σ_r is either completely in G^σ_x or it does use vertices in $V(G^\sigma_r) \setminus V(G^\sigma_x)$. Since there is no negative cycle in G^σ_r and due to Lemma 6.23, a shortest u-v path completely in G^σ_x has length $\min_{u' \in V(G_i)} dist_{G_i}(u, u') + dist_{H_w}(v_i, v_j) - w(v_i) - w(v_j) + \min_{v' \in V(G_j)} dist_{G_j}(v', v)$. If a shortest u-v path does use vertices in $V(G^\sigma_r) \setminus V(G^\sigma_x)$, we can determine the length by $\min_{u' \in V(G^\sigma_x)} dist_{G^\sigma_x}(u, u') + c_x + \min_{v' \in V(G^\sigma_x)} dist_{G^\sigma_x}(v', v)$. Thus, for a node $x \in S_r$ labeled according to an operation SUBST_H, we can compute the shortest-path distance of two vertices that are in different modules by taking the minimum of those two values.

For a node $x \in S_r$ that is not labeled according to an operation SUBST_H, we have already computed $f_I(G^\sigma_x)$ and thus, $dist_{G^\sigma_x}(u, v)$ is known for all $u, v \in V(G^\sigma_x)$ and it holds that $dist_{G^\sigma_r}(u, v) = \min\{dist_{G^\sigma_x}(u, v), \min_{u' \in V(G^\sigma_x)} dist_{G^\sigma_x}(u, u') + c_x + \min_{v' \in V(G^\sigma_x)} dist_{G^\sigma_x}(v', v)\}$.

Note that for a pair of vertices $u, v \in V(G^\sigma_r)$ it either holds that $u, v \in V(G^\sigma_x)$ for some $x \in S_r$ that is not labeled according to an operation SUBST_H or it holds that $u, v \in V(G^\sigma_x)$ for some $x \in S_r$ that is labeled according to an operation SUBST_H and u and v are in different modules.

Since all the considered values are known, we can compute $dist_{G_r}(u, v)$ for all $u, v \in V(G_r)$ in time $\mathcal{O}(n^2)$.

We are left to compute the values c_x for each node $x \in S_r$. We do this in a top-down traversal of the nodes in S_r. For the root $r \in S_r$ it holds that $c_r = \infty$. Let $x \in S_r$ be a node labeled according to an operation $\textsc{Subst}_H$ and let $v_1, \ldots, v_t \in T_r^\sigma$ be the children of x, i.e., $G_x^\sigma = \textsc{Subst}_H(G_{v_1}^\sigma, \ldots, G_{v_t}^\sigma)$. Further, let H_w as defined in Definition 6.24 with vertex set $\{v_1, \ldots, v_t\}$ and vertex weights ω. Inductively, we can assume that c_x is know. Now, c_{v_i} corresponds to a cycle that either only uses vertices in $V(G_x^\sigma)$ or it uses vertices in $V(G_r^\sigma) \backslash V(G_x^\sigma)$. Thus, c_{v_i} is the minimum of the two values:

- $\min_{C \in \mathcal{C}_{v_i}} \omega(C) - \omega(v_i)$ with $\mathcal{C}_{v_i}$ is the set of all cycles from v_i to v_i in H_w
- $\min_{v_j \in V(H_w)} dist_{H_w}(v_i, v_j) - \omega(v_i) + c_x + \min_{v_j \in V(H_w)}(v_j, v_i) - \omega(v_i)$.

Since all considered values are known, we can compute the values c_x for each $x \in U_r$ by a top-down traversal in time $\mathcal{O}(|S_r|) \subseteq \mathcal{O}(n)$. □

Lemma 6.31. *Let $G' = (V', E')$ be a directed graph, let $x \notin V(G')$, let $E_x \subseteq \{(x, u) \mid u \in V'\} \cup \{(u, x) \mid u \in V'\}$, let $w \colon V' \cup \{x\} \to \mathbb{R}$, and let $f_I(G')$ be given. Then one can compute $f_I(\textsc{Inc}_{x,E_x}(G'))$ in time $\mathcal{O}(n^2)$, with $n = |V'|$.*

Proof. Let $G = \textsc{Inc}_{x,E_x}(G')$. By Lemma 6.22, we can compute the value $\texttt{msp}(G)$ of a minimum shortest path and a feasible potential for G in time $\mathcal{O}(m + n \log n)$. Apply Dijkstra's algorithm twice to compute the values $dist_G(x, v)$ and $dist_G(v, x)$ for all $v \in V'$ in the same running time. For $u, v \in V'$, a shortest u-v path in G does either use the vertex x, or it does not use the vertex x. Thus, we can update the shortest-path distance for each pair $u, v \in V'$ by $dist_G(u, v) = \min\{dist_{G'}(u, v), dist_G(u, x) + dist_G(x, u)\}$, which can be done for each pair in constant time. Additionally, for each $v \in V(G)$, the values $\min_{u \in V(G)} dist(v, u)$ and $\min_{u \in V(G)} dist(u, v)$ can be looked up in linear time, yielding a total running time of $\mathcal{O}(n^2)$ to compute $f_I(G)$. □

Lemma 6.32. *Let H be an unweighted, directed t-vertex graph, G_i be vertex-weighted, directed graphs, and $f_S(G_i)$ be given for each G_i. Then one can compute $f_S(\textsc{Subst}_H(G_1, \ldots, G_t))$ in time $\mathcal{O}(\texttt{td}(H) \cdot t^2 + n)$ with $n = |V(\textsc{Subst}_H(G_1, \ldots, G_t))|$.*

Proof. Let $G = \textsc{Subst}_H(G_1, \ldots, G_t)$. Consider the graph H_w as defined in Definition 6.24 with $V(H_w) = \{v_1, \ldots, v_t\}$. Note that $\texttt{td}(H_w) = \texttt{td}(H)$, i.e., $H_w \in \textsc{Td}_{\texttt{td}(H)}$, thus, by Lemma 6.31 and Corollary 6.11, one can compute $f_I(H_w)$, especially VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, in H_w in time $\mathcal{O}(\texttt{td}(H) \cdot t^2)$. Since $f_S(G_i)$ is known, one can compute the values $\min_{v \in V(G)} dist_G(u, v)$ resp. $\min_{v \in V(G)} dist_G(v, u)$ as done in the proof of Lemma 6.29 in time $\mathcal{O}(n)$. Moreover, a shortest-path feasible potential for G and the value $\texttt{msp}(G)$ can also be computed as done in Section 6.4.2 for NEGATIVE CYCLE DETECTION in $\mathcal{O}(t^2)$. □

Proof of Theorem 6.28. Let σ be the given expression after replacing each occurrence of an operation JOIN or UNION by (possible multiple) operations $\textsc{Subst}_H$ with a pattern graph H of size two. We traverse the expression tree T^σ from bottom to top. The values $f_I(\bullet)$,$f_S(\bullet)$, $f_I(\circ)$, and $f_I(\circ)$ can be trivially computed in constant time. For a node $x \in T^\sigma$ labeled according to an operation $\textsc{Subst}_H$, the algorithms A_{Sub} resp. A_{SubTd} are as described in Lemma 6.29 resp. Lemma 6.32 and compute $f_S(T_x^\sigma)$. For each node $x \in T^\sigma$ labeled according to an operation $\textsc{Inc}_{x,E_x}$, let $y \in T^\sigma$ be the unique child of x. If y is labeled according to an operation $\textsc{Subst}_H$, the algorithm A_{Inc} first computes $f_I(T_y^\sigma)$ using Lemma 6.30. Then, the algorithm computes $f_I(T_x^\sigma)$ using Lemma 6.31. The linear portions of the running time of A_{Sub} and A_{SubTd} sum up to $\mathcal{O}(n^2)$ total time. By Theorem 6.10, we have proven the theorem. □

6.5 Comparing the Graph Classes

In this section, we compare the graph classes defined in Section 6.2. It is well known that the graph classes TD_k and MW_h are incomparable: For any $k, h \in \mathbb{N}$ it holds that $K_{k+1} \in \text{MW}_0$, but $K_{k+1} \notin \text{TD}_k$, for K_{k+1} being the complete graph of size $k+1$. Conversely, let G be a subdivided star of degree h, i.e, the graph $K_{1,h}$ with a pendant vertex attached to each vertex of degree one, cf. Figure 6.1. Then it holds that $G \in \text{TD}_3$, but $G \notin \text{MW}_h$.

Naturally it holds that $\text{MW}_h \subseteq \text{TD}_0\text{MW}_h$ and $\text{TD}_k \subseteq \text{TD}_k\text{MW}_0$ for any $k, h \geq 0$; in fact, it can be observed that $\text{MW}_h = \text{TD}_0\text{MW}_h$ and $\text{TD}_k \subsetneq \text{TD}_k\text{MW}_0$.[6] Conversely, the following lemma shows that for any $k, h \in \mathbb{N}$ there exist graphs that are neither in TD_k nor in MW_h, but that are contained in $\text{TD}_{k'}\text{MW}_{h'}$ even for constant k' and h'.

Lemma 6.33. *For any $k, h \in \mathbb{N}$, there exists a graph G such that $G \notin \text{TD}_k$ and $G \notin \text{MW}_h$, but $G \in \text{TD}_1\text{MW}_0$.*

Proof. Let $p = \max\{k, h, 2\}$. We construct a graph G with $2p+1$ vertices as follows: Consider p pairs of non-adjacent vertices $v_{i,1}$ and $v_{i,2}$ for $i \in [p]$ and one additional vertex called x, i.e., $V = \{v_{1,1}, v_{1,2}, v_{2,1}, v_{2,2}, \ldots, v_{p,1}, v_{p,2}, x\}$. There is an edge between any $v_{i,r}$ and $v_{j,s}$ if and only if $i \neq j$ for $r, s \in \{1, 2\}$. The vertex x is connected to each $v_{i,1}$ for $i \in [p]$. See also Figure 6.1. Now, in this constructed graph G the vertex set $\{v_{i,1} \mid i \in [p]\}$ (as the set $\{v_{i,2} \mid i \in [p]\}$) is a clique of size p, thus, $\mathsf{td}(G) \geq p$. On the other side, the graph G does not admit any non-trivial module, since it is easy to see that every minimal extension of a two-vertex set results in the whole vertex set $V(G)$. Thus, $\mathsf{h}(G) = |V(G)| = 2p+1$.

At the same time, $G \setminus \{x\} \in \text{TD}_0\text{MW}_0$ since $G \setminus \{x\}$ is a cograph, implying that $G \in \text{TD}_1\text{MW}_0$. □

Clearly, also the notion of modular tree-depth generalizes both modular-width and tree-depth: For a graph G, the modular tree-depth matches the tree-depth for the special case of G being prime and since it holds that $\mathsf{td}(G) \leq |V(G)|$ for any graph G, it also holds that $\mathsf{mtd}(G) \leq \mathsf{mw}(G)$. The next lemma shows that in general neither the modular-width nor the tree-depth can be bounded by a function depending on the modular-tree-depth of a graph.

Lemma 6.34. *For any $k, h \in \mathbb{N}$, there exists a graph G such that $G \notin \text{TD}_k$ and $G \notin \text{MW}_h$, but $G \in \text{MTD}_3$.*

Proof. Let $p \in \mathbb{N}$ such that $2p+1 > h$. We construct a pattern graph H with $2p+1$ vertices as follows: Let H be a subdivided star of degree p with $2p+1$ vertices, cf. Figure 6.1. Denote $V(H) = \{v_1, v_2, \ldots, v_{2p+1}\}$, and let $G = \text{SUBST}_H(K_{k+1}, \ldots, K_{k+1})$. Now, $G \notin \text{TD}_k$ since G contains an clique of size $k+1$ and $G \notin \text{MW}_h$, since G contains an induced subdivided star with more than h vertices.

At the same time, since $\mathsf{td}(H) = 3$ and $K_{k+1} \in \text{MTD}_0$, it holds that $G \in \text{MTD}_3$. □

As a consequence of Lemma 6.33 and Lemma 6.34 it follows that also the classes $\text{MW}_h\text{MTD}_\ell$ and $\text{TD}_k\text{MTD}_\ell$ are strictly more general then the classes MW_h and TD_k.

Next, we will compare the class MTD_ℓ from Definition 6.6 with the class TD_kMW_h from Definition 6.3. We will see that they are indeed incomparable. A simple extension of the proof of Lemma 6.33 yields the following lemma.

Lemma 6.35. *For any $\ell \in \mathbb{N}$, there exists a graph G such that $G \notin \text{MTD}_\ell$, but $G \in \text{TD}_1\text{MW}_0$.*

[6] Similar trivial relations are true for other classes, we do not state them explicitly.

Figure 6.1: Left: The graph constructed in the proof of Lemma 6.33 for $p = 4$. The bold edges indicate a full join. Right: A subdivided star of degree six.

Proof. Let G be the graph constructed in the proof of Lemma 6.33 with $p = \ell$, cf. left side of Figure 6.1. As mentioned in the proof of Lemma 6.33, it holds that $G \in \text{TD}_1\text{MW}_0$. However, the constructed graph G is prime and $\mathsf{td}(G) \geq p$. Thus, $\mathsf{mtd}(G) \geq p$. □

Conversely, for each $k, h \in \mathbb{N}$, already the graph class MTD_3 contains graphs that are not in TD_kMW_h.

Lemma 6.36. *For any $k, h \in \mathbb{N}$, there exists a graph G such that $G \notin \text{TD}_k\text{MW}_h$, but $G \in \text{MTD}_3$.*

Proof. Let $p \in \mathbb{N}$ such that $2p + 1 > h$. We construct G as done in Lemma 6.34, i.e., let H be a subdivided star with $2p + 1$ vertices with $V(H) = \{v_1, v_2, \ldots, v_{2p+1}\}$, and let $G = \text{SUBST}_H(K_{k+1}, \ldots, K_{k+1})$. Denote by M_i, for $i \in [2p + 1]$, the $2p + 1$ many modules in G. It was already shown that $G \in \text{MTD}_3$.

We claim that $G \notin \text{TD}_k\text{MW}_h$: Due to the structure of G, for any two vertices $u \in M_i$, $v \in M_j$, with $i \neq j$, it holds that there is no module in G containing u and v. Thus, any modular partition of G consists of at least $2p + 1$ many modules, which implies that G cannot be in TD_0MW_j for $j \leq h < 2p + 1$. Moreover, this statement holds even after deleting any k vertices in G. Thus, G cannot be in TD_iMW_j for any $i \leq k$ and $j \leq h$. □

Although the modular tree-depth is always upper bounded by the modular-width, it is possible that these two parameters only differ by a constant factor. Thus, depending on the application and the input, it may be beneficial to consider the modular-width instead of the more general modular tree-depth, since its (asymptotic) impact on the running time may be much lower.

Lemma 6.37. *For any $\ell \in \mathbb{N}$ with $\ell \geq 2$, there exists a graph G such that $G \notin \text{MTD}_\ell$, but $G \in \text{MW}_{2\ell}$.*

Proof. Consider a graph G that is a clique K_ℓ of size ℓ with a pendant vertex for each vertex in K_ℓ. Now, there is no non-trivial module in G and since G contains a clique of size ℓ it holds that $\mathsf{td}(G) > \ell$, thus, $G \notin \text{MTD}_\ell$. However, since $|V(G)| = 2\ell$ it trivially holds that $\mathsf{h}(G) = 2\ell$. □

Next, we show that the classes TD_kMW_h and $\text{MW}_h\text{MTD}_\ell$ are incomparable.

Lemma 6.38. *For any $h, \ell \in \mathbb{N}$, there exists a graph G such that $G \notin \text{MW}_h\text{MTD}_\ell$, but $G \in \text{TD}_1\text{MW}_0$.*

Proof. Consider the graph constructed in the proof of Lemma 6.33 for $p = \ell$, cf. the left side of Figure 6.1. Since G does not admit any non-trivial module and contains a clique of size ℓ it holds that $G \notin \text{MTD}_\ell$, but $G \in \text{TD}_1\text{MW}_0$ as shown in the proof of Lemma 6.36. □

Corollary 6.39. *For any $k, h \in \mathbb{N}$, there exists a graph G such that $G \notin \text{TD}_k\text{MW}_h$, but $G \in \text{MW}_0\text{MTD}_3$.*

Proof. This claim follows directly from Lemma 6.36. □

Finally, we compare the graph classes with the class $\text{TD}_k\text{MTD}_\ell$ and obtain the following relationships.

Corollary 6.40. *For any $h, \ell \in \mathbb{N}$, there exists a graph G such that $G \notin \text{MW}_h\text{MTD}_\ell$, but $G \in \text{TD}_1\text{MTD}_0$.*

Proof. This claim follows directly from Lemma 6.35. □

Clearly, for any graph $G \in \text{MW}_h\text{MTD}_\ell$ it holds that $G \in \text{TD}_0\text{MTD}_{\max\{h,\ell\}}$. Thus, it is not possible that for arbitrary $k, \ell \geq 0$ there exists graphs $G \notin \text{TD}_k\text{MTD}_\ell$, but $G \in \text{MW}_{h'}\text{MTD}_{\ell'}$ for constants h' and ℓ'. However, it is possible to make such a claim with ℓ' being a constant.

Lemma 6.41. *For any $k, \ell \in \mathbb{N}$ with $k, \ell \geq 2$, there exists a graph G such that $G \notin \text{TD}_k\text{MTD}_\ell$, but $G \in \text{MW}_{2\ell}\text{MTD}_0$.*

Proof. Let H be a clique K_ℓ of size ℓ with a pendant vertex for each vertex in K_ℓ and let $G = \text{SUBST}_H(K_{k+1}, \ldots, K_{k+1})$. Now, it holds that $G \notin \text{TD}_k\text{MTD}_\ell$ but $\mathsf{h}(G) = 2\ell$. □

Corollary 6.42. *For any $k, h \in \mathbb{N}$, there exists a graph G such that $G \notin \text{TD}_k\text{MW}_h$, but $G \in \text{TD}_0\text{MTD}_3$.*

Proof. This claim follows directly from Lemma 6.36. □

Clearly, for any graph $G \in \text{TD}_k\text{MW}_h$ it holds that $G \in \text{TD}_k\text{MTD}_\ell$ with $\ell = h$. Thus, it is again not possible that for arbitrary $k, \ell \geq 1$ there exists graphs $G \notin \text{TD}_k\text{MTD}_\ell$, but $G \in \text{TD}_{k'}\text{MW}_{h'}$ for constants k' and h'. However, it is possible to make such a claim with only k' being a constant.

Corollary 6.43. *For any $k, \ell \in \mathbb{N}$ with $k, \ell \geq 2$, there exists a graph G such that $G \notin \text{TD}_k\text{MTD}_\ell$, but $G \in \text{TD}_0\text{MW}_{2\ell}$.*

Proof. This claim follows directly from Lemma 6.41. □

Relationships to known parameters. A different way of combining tree-depth and modular-width lies in the notion of shrub-depth, introduced by Ganian et al. [GHN+19], which, in fact, is a tree-depth-like variation of clique-width. Graph classes of bounded tree-depth also have bounded shrub-depth while also dense graph classes, e.g. the class of all cliques, have bounded shrub-depth. Since cographs do not have bounded shrub-depth, all graph classes with heterogeneous structure introduced in this work are incomparable to shrub-depth. Moreover, to our best knowledge, there are not yet any dedicated efficient parameterized algorithms for graphs of small shrub-depth.

Let us also point out that for any fixed values of k, ℓ, and h all classes introduced in this chapter have bounded clique-width. In particilar, it holds that $\mathsf{cw}(G) \leq \mathsf{cw}(\text{INC}_{x,E_x}(G)) \leq 2 \cdot \mathsf{cw}(G)$, i.e., the addition of a vertex at most doubles the clique-width of a graph (this also

holds for the NLC-width) and does not shrink [Gur17]. That said, the clique-width may be much larger than these parameters defined in this chapter and the best running times relative to clique-width can be worse (and they are for the examples provided in this work, cf. [CDP19, Duc22b]). In other words, we obtain faster algorithms by making use of the specific (heterogeneous) structure of the introduced graph classes, rather than considering them as graphs of bounded clique-width.

Regarding tree-width and branch-width, we observe that the presented graph classes remain incomparable to these (as modular-width and modular tree-depth alone), as there are no constants $k, h, \ell \in \mathbb{N}$ such that the class of all paths are contained in $\text{TD}_k\text{MW}_h\text{MTD}_\ell$. Trivially, there are always constants $k, h, \ell \in N$ such that a given H-minor free graph class is not contained in $\text{TD}_k\text{MW}_h\text{MTD}_\ell$ since one can build any graph H with $|V(H)|$ many increment operations.

6.6 Conclusion

Coping with the limited generality of graph structures for which we know fast algorithms, we have presented in this chapter a clean and robust way of defining more general, heterogeneous structure via graph operations and algebraic expressions. We gave a generic framework that allows to design algorithms with competitive running times by designing routines for any subset of the considered operations. As applications we showed algorithms for TRIANGLE COUNTING, NEGATIVE CYCLE DETECTION, and VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS.

The heterogeneous merge of the parameter tree-depth and modular-width can here only be seen as an example as similar approaches for other parameters that can be defined via operations are inevitable. Thus, in future work we aim to extend this approach so that further operations and problems can be studied. We refer also to the discussion on combining further parameters in a heterogeneous way in Section 7.2. This often touches upon concrete algorithmic questions for well-studied parameters. For example, it is an open problem to design a non-trivial algorithm for MAXIMUM MATCHING relative to modular tree-depth or, equivalently, for MAXIMUM b-MATCHING relative to tree-depth. Such an algorithm would yield a fast algorithm for MAXIMUM MATCHING for all graphs in $\text{TD}_k\text{MW}_h\text{MTD}_\ell$. As it stands, known algorithms for MAXIMUM MATCHING parameterized by the tree-depth [IOO18] and modular-width (Section 3.2) yield an algorithm with time complexity $\mathcal{O}((h^2 \log h)n + km)$ for graphs in TD_kMW_h via Theorem 6.10 (with a function $f(G)$ that just returns a maximum matching in G). We want to highlight at this point that such results does not hold inevitably. Even if there are separate algorithms A_{Inc}, A_{Sub}, resp. A_{SubTd} for the homogeneous case, each algorithm might use a different set of auxiliary values that needs to be computed, defined by a function f. In the heterogeneous case, it does not hold automatically that one can provide all those auxiliary values for all the different algorithms, which often gives rise to interesting new algorithmic questions (as also seen in this chapter for the algorithm for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS).

Another important spin-off question is to determine the complexity of finding for a given graph, a set of operations, and suitable parameters, e.g., $k, h, \ell \in \mathbb{N}$, an equivalent algebraic expression, i.e., a decomposition of the heterogeneous structure of the graph. Nevertheless, because algebraic expressions over some set of graph operations are also graph constructions, we already learn what processes create graphs of beneficial heterogeneous structure.

Part IV

Conclusion

7

Concluding Remarks and Open Problems

In this thesis, we carried forward the study of several popular problems in the "FPT in P" framework for well-studied parameters (and the much older programs of adaptive algorithms and faster algorithms for restricted settings). In the first main part of this thesis, we obtained efficient parameterized algorithm with respect to the parameters modular-width, clique-width, and twin-width for several fundamental problems as MAXIMUM MATCHING, VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, or TRIANGLE COUNTING.

The second main part of this thesis was motivated by the great success of parameterized algorithms and the wealth of algorithms benefiting from different structures in the input, both for problems in P and NP. While each of the parameterized algorithms utilize a different specific structure, we discussed how to combine two or more parameters to much more general and heterogeneous graph classes, while still being able to solve a problem efficiently. We have shown how to define such heterogeneous graph classes in a clean and robust way and presented a general running-time framework for such classes.

In this chapter, we will give concluding remarks to both parts of this thesis and present various interesting open questions and directions for further research.

7.1 Algorithms Parameterized by a Single Parameter

For the parameter modular-width, we achieved in Chapter 3 several algorithms that are adaptive, i.e., they match the trivial lower bound of the problem for constant parameter size, and match the running time of an optimal unparameterized algorithm for worst-case parameter value $\mathsf{mw} \in \Theta(n)$. Thus, already for slightly non-trivial parameter value $\mathsf{mw} \in o(n)$ our algorithm outperforms the optimal unparameterized algorithm. Broadly speaking, each algorithm parameterized by the modular-width traversed the modular-decomposition tree, simplified and compressed the current graph, and encoded all necessary information in the quotient graph; either by using weights resp. capacities in the quotient graph, by slightly extending the quotient

graph, or both. Solving a weighted variant of the considered problem on the quotient graphs then yielded the desired result. E.g., for MAXIMUM MATCHING we could compress all important information into a vertex-capacitated graph with at most $3\,\mathsf{mw}$ many vertices and could compute the size of a maximum matching by solving a weighted variant of MAXIMUM MATCHING, namely MAXIMUM b-MATCHING, on this smaller graph; or we have reduced the problem of finding a maximum number of edge-disjoint s-t in an unweighted graph to the computation of an edge-capacitated maximum s-t flow on a graph closely related to the quotient graph. While for most problems, the weighted variant require a higher running time as the unweighted variant,[1], we could not achieve adaptive running times for MAXIMUM MATCHING, MAXIMUM EDGE-DISJOINT s-t PATHS, and GLOBAL MINIMUM EDGE CUT, while the achieved running times still yield a large regime for mw where we achieve faster algorithm.

Considering the vertex-weighted variant in the first place, we achieved adaptive running times for MAXIMUM b-MATCHING, VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS, MAXIMUM s-t VERTEX FLOW, WEIGHTED GLOBAL VERTEX MIN CUT, and TRIANGLE COUNTING. This immediately motivates the following question.

Open Problem. *For which other combinations of problems and parameters exist adaptive algorithms?*

We have partially answered this question for the problem TRIANGLE COUNTING by giving adaptive algorithms for TRIANGLE COUNTING with regard to the parameters clique-width, twin-width, tree-depth, and modular-tree-width, if we restrict ourselves to combinatorial algorithms on dense graphs. In this context, it would be interesting to see if one could leverage fast matrix multiplication in those parameterized algorithm to speed up the running time, such that one gets adaptive algorithms against the classic $\mathcal{O}(n^\omega)$-time algorithm for TRIANGLE COUNTING. We stress that adaptive algorithms are particularly appealing for graph parameters for which one can compute the beneficial structure faster than the optimal unparameterized running time of the problem. However, such structures tend to be not very general.

On the other side, one always strives to consider parameters that are as general as possible, i.e., to capture a wider class of graphs for which we get improved algorithms. In this respect, we have considered in Chapter 4 the parameter clique-width which is much less restrictive than the parameter modular-width. Our main result here was the $\mathcal{O}(\mathsf{cw}^2\, n^2)$-time algorithm for the problem VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS when parameterized by the clique-width cw of the input. This is the first algorithm parameterized by the clique-width for this problem. For this we considered the closely related parameter NLC-width and the corresponding expression tree. In three phases, we have first computed some local auxiliary values, extended those with more global auxiliary values in the second phase, and eventually used those values to compute for each pair of vertices in a final traversal the actual shortest distance. A natural further-reaching question regarding VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS is whether one can improve the result towards an *adaptive* algorithm parameterized by the clique-width of the input.

Open Problem. *Is there an adaptive parameterized algorithm for* VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS when parameterized by the clique-width of a graph.

In the presented algorithm, we computed edge-weighted ALL-PAIRS SHORTEST PATHS on the auxiliary graph H_w using simply the algorithm due to Floyd and Warshall. However, this graph

[1] The problem TRIANGLE COUNTING is an exception here, as one can compute the capacitated variant of TRIANGLE COUNTING also in time $\mathcal{O}(n^\omega)$, as seen in Section 3.5

does itself admit quite some structure due to the circular structure, e.g., H_w is bipartite. It is not known if one can solve ALL-PAIRS SHORTEST PATHS faster on bipartite graph; Torgasin and Zimmermann [TZ13] presented an algorithm for ALL-PAIRS SHORTEST PATHS on directed and bipartite graphs with real-weighted weights without negative cycles that reduces the size of matrices used in the computations to 1/4 of the size of the original adjacency matrix, while the running time becomes at least seven times faster for large graphs when compared to the standard algorithm; while also other heuristic algorithms for ALL-PAIRS SHORTEST PATHS on bipartite graphs exists, e.g., [HZA22]. However,in the presented algorithm there would still be the bottleneck of computing the $d^x_{v,i,y}$-values in the second phase. After the present work, Ducoffe [Duc22b] advanced in this direction by giving an almost adaptive algorithm of time $\mathcal{O}(\mathsf{cw}(n \log n)^2)$ for VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS. A caveat here is that this algorithm considers only undirected graphs with non-negative vertex weights.

As a second result, we presented an $\mathcal{O}(cw^2n+\mathsf{cw}\, m)$-time algorithm for the problem TRIANGLE COUNTING when parameterized by the clique-width of the input, which is a slight improvement over the previous algorithm with a time complexity of $\mathcal{O}(\mathsf{cw}^2(n+m))$. This algorithm is only adaptive, if we restrict ourselves to combinatorial algorithms on dense graphs.

Open Problem. *Is there a non-combinatorial adaptive algorithm for* TC when parameterized by the clique-width of a graph.

Concerning further tractable problems, very recently Ducoffe [Duc22a] was able to solve MAXIMUM MATCHING when parameterized by the clique-width cw of the graph in time $\mathcal{O}(f(\mathsf{cw}) \cdot (n+m)^{1+o(1)})$ using among other techniques Courcelle's theorem [CMR00], where f is a computable function.[2] While this algorithm yields a almost linear time for MAXIMUM MATCHING on graphs of bounded clique-width, the actual dependency on the clique-width is non-elementarily. It would be interesting to know, if there exists also a combinatorial algorithm for MAXIMUM MATCHING parameterized by the clique-width, e.g., an algorithm where the dependency on the clique-width is only exponential.

Open Problem. *Is there an* combinatorial *algorithm for* MAXIMUM MATCHING when parameterized by the clique-width of a graph.

Finally, we have turned our focus to the parameter twin-width, the most general parameter considered in this thesis. We presented an algorithm based on dynamic programming to compute the number of triangles in a graph in time $\mathcal{O}(d^2n + m)$, assuming that a d-contraction sequence is given in a compact way. For several NP-hard problems, e.g., k-INDEPENDENT SET, k-CLIQUE, k-VERTEX COVER, k-DOMINATING SET, or k-SUBGRAPH ISOMORPHISM it was shown that one can solve them in fpt-time $\mathcal{O}(f(k,d) \cdot n)$, assuming that a d-contraction sequence is provided [BKTW20].

In the realm of "FPT in P", Bonnet et al. [BGK+21b] solved unweighted unweighted ALL-PAIRS SHORTEST PATHS in time $\mathcal{O}(dn^2 \log n)$ on graphs of twin-width at most d assuming a given d-contraction sequence in a compact way. For this, they used a so-called interval biclique partition and showed that one compute a interval biclique partition of size $\mathcal{O}(dn)$ in time $\mathcal{O}(dn)$ using the d-contraction sequence. Apart from this, the (recently introduced) parameter twin-width got little attention regarding tractable problems. Next to generalizing the result of Bonnet

[2]Note that one cannot express a matching in MSO_1 logic and thus, Courcelle's theorem cannot be applied directly. However, using a *Counting* MSO_1 formula one can evaluate via the Tutte-Berge formula [Ber01] the size of a maximum matching. Evaluating this formula on various subgraphs then also lead to the actual maximum matching [Duc22a].

et al. to vertex-weighted graphs, it is an open problem if one can solve MAXIMUM MATCHING faster on graphs of bounded twin-width.

Open Problem. *Is there a parameterized algorithm for* MAXIMUM MATCHING when parameterized by the twin-width of a graph.

7.2 Heterogeneous Structure

Considering the wealth of parameterized algorithms for many computational problems and many (incomparable) parameters, we have formalized in Chapter 6 how to combine two or more different parameters into much more general graph classes of heterogeneous structure. We presented a framework for designing operation-based algorithms and extended our running-time framework from Theorem 3.15 to cope with all presented heterogeneous graph classes. We show-cased this approach combining the parameters tree-depth, modular-width, and the natural notion of modular tree-depth. We presented algorithms for TRIANGLE COUNTING, NEGATIVE CYCLE DETECTION, and VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS relative to the heterogeneous graph classes with running times that match the running times for the homogeneous cases.

The applicability of this framework to other parameters is evident. Consider as an example the parameter path-width (cf. Section 2.3 for a standard definition). One can define graphs of path-width at most k via operations in the following way: Similarly to the definition of clique-width, we consider labeled graph where each vertex is either labeled "active" or "inactive". We then define an operation $deact_v(G)$ that sets the label of a vertex $v \in V(G)$ to "inactive", and an operation $\textsc{Inc}_{x,E_x}(G)$ with $lab(x) =$ "active" and $E_x \subseteq \{\{x, v\} \mid v \in V \wedge lab(v) =$ "active"$\}$ that adds a new vertex with a neighborhood only consisting of "active" vertices. If we now restrict the number of "active" vertices at each time to be at most $k + 1$, this exactly defines the graphs of path-width at most k. The heterogeneous combination with, e.g., modular-width can be done with the same approach presented in this chapter. If we denote the resulting graph class by $\textsc{Pw}_p\textsc{Mw}_h$ where $p \in N$ denote the maximum number of simultaneous "active" vertices and h denotes the maximum size of an operation $\textsc{Subst}_H$, one can show that the problem TRIANGLE COUNTING can be solved in time $\mathcal{O}(p^2n + h^{\omega-1}n)$.

Open Problem. *Which combinations of parameters allow efficient parameterized algorithms for the problems* MAXIMUM MATCHING or VERTEX-WEIGHTED ALL-PAIRS SHORTEST PATHS.

The set of operations used the presented framework was motivated by the operations needed to define specific graph parameters. However, it is also possible to simply define a set of operations, some of them are limited by a parameter, e.g., defining the maximum nesting depth of an operation of the size of a pattern. By considering all algebraic expressions over the set of operations, we can simply define the corresponding graph class.

As a simple example, we can extend the set of operations in the definition of clique-width (cf. Chapter 4) by an operation that adds a pendant vertex and an operation that adds a dominating vertex (each with an arbitrary label). Interestingly, the graph class consisting of all graphs that get created using only the latter two operations, is of unbounded clique-width [Rao08a]. However, it is easy to extend the algorithm for TRIANGLE COUNTING parameterized by the NLC-width to cope with the latter two operations, immediately extending the class of graph for which this algorithm yield a faster algorithm for TRIANGLE COUNTING.

Such *one-vertex extensions* were also considered by Ducoffe and Popa [DP21b] for MAXIMUM MATCHING on modular-decomposition trees. With the cost of an additionally sum of $\mathcal{O}m \log n$

in the running time, they showed that one can prune every quotient graph by the one-vertex extensions pendant, anti-pendant, twin, universal and isolated vertex (while universal and isolated vertices can only occur after the removal of other one-vertex extensions), i.e., that parameter value is defined as the maximum size of any quotient graph *after* the removal of all one-vertex extensions.

to define a heterogeneous graph class parameterized by several

Open Problem. *What is a maximum set of operations such that there exists an efficient parameterized algorithm for* TRIANGLE COUNTING.

Generalize $\textsc{Inc}_{x,E_x}$ to IncMod

We have generalized modular-width to modular tree-depth by allowing pattern graphs for which the tree-depth of the pattern is bounded instead of the size. In principle, as long as the pattern graph H might be "easy", there is hope that one can also solve problems in $\textsc{Subst}_H(G_1, \ldots, G_t)$, assuming that the graphs G_i are also capable of being solved efficiently. Since we postulate that every algebraic expression over the given set should be "easy", one interesting idea would be to create the pattern graph H for an operation $\textsc{Subst}_H$ by using the same set of operations, i.e., we generalize $\textsc{Subst}_H(G_1, \ldots, G_t)$ to $\textsc{Subst}(G_1, \ldots, G_t, H)$ that still returns $H[v_1 \leftarrow G_1, \ldots, v_t \leftarrow G_t]$, however the last argument defines the pattern graph H. I.e., in a suitable expression tree, the node corresponding to such an operation admits $t + 1$ children, where the subexpression tree of the first t children define G_i for $i \in [t]$ and the subexpression tree of the last child defines H.

Doing this, the number of parameters would shrink to only the maximum number of

One can show that the size of such an expression tree would still be $\mathcal{O}(n)$, thus the increment trick still works

Open Problem. *What is the complexity of fining for a given set of operations and a given graph, a suitable expression*

Ideally, each problem then might have their own set of operations that are efficiently processable, which can be seen as a first step of what structure might be beneficial, and what structure not.

For tree-depth, one can decide in linear time for a fixed k, if a graph has tree-depth k, actual time is $2^{\mathcal{O}(t^2)} \cdot n$ [RRVS14] (NP-hard in general [Pot88])

Bibliography

[ABW15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015. (Cited on page 8.)

[ACL95] Mikhail J. Atallah, Danny Z. Chen, and D. T. Lee. An optimal algorithm for shortest paths on weighted interval and circular-arc graphs, with applications. *Algorithmica*, 14(5):429–441, 1995. (Cited on pages 8 and 26.)

[Ans87] Richard P. Anstee. A polynomial algorithm for b-matchings: An alternative approach. *Inf. Process. Lett.*, 24(3):153–157, 1987. (Cited on page 25.)

[AWY18] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM J. Comput.*, 47(3):1098–1122, 2018. (Cited on page 7.)

[AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. (Cited on pages 27, 33, 47, and 92.)

[BBCG10] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient algorithms for large-scale local triangle counting. *ACM Trans. Knowl. Discov. Data*, 4(3):13:1–13:28, 2010. (Cited on page 12.)

[BBD22] Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is np-complete. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. (Cited on page 84.)

[Ber88] Alan A. Bertossi. Parallel circle-cover algorithms. *Inf. Process. Lett.*, 27(3):133–139, 1988. (Cited on page 8.)

[Ber01] Claude Berge. *The theory of graphs*. Courier Corporation, 2001. (Cited on page 121.)

[BFNN19] Matthias Bentert, Till Fluschnik, André Nichterlein, and Rolf Niedermeier. Parameterized aspects of triangle enumeration. *J. Comput. Syst. Sci.*, 103:61–77, 2019. (Cited on pages 8 and 27.)

[BGHK95] Hans L. Bodlaender, John R. Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, 1995. (Cited on page 24.)

[BGK+21a] Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1977–1996. SIAM, 2021. (Cited on page 84.)

[BGK+21b] Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set, min dominating set, and coloring. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on pages 26, 84, 92, and 121.)

[BK15] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97, 2015. (Cited on page 8.)

[BK18] Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1216–1235, 2018. (Cited on pages 8 and 58.)

[BKTW20] Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 601–612. IEEE, 2020. (Cited on pages 22, 83, 84, 85, 92, and 121.)

[Blu90] Norbert Blum. A new approach to maximum matching in general graphs. In Mike Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 586–597. Springer, 1990. (Cited on page 25.)

[BN18] Matthias Bentert and André Nichterlein. Parameterized complexity of diameter. *CoRR*, abs/1802.10048, 2018. (Cited on pages 26 and 95.)

[BR96] V. Balachandhran and C. Pandu Rangan. All-pairs-shortest-length on strongly chordal graphs. *Discret. Appl. Math.*, 69(1-2):169–182, 1996. (Cited on page 26.)

[Bri14] Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670. IEEE Computer Society, 2014. (Cited on page 7.)

[BS05] Pál Burai and Arpád Száz. Relationships between homogeneity, subadditivity and convexity properties. *Publikacije Elektrotehničkog fakulteta. Serija Matematika*, pages 77–87, 2005. (Cited on pages 42 and 100.)

[BTV11] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-width of graphs. *Theor. Comput. Sci.*, 412(39):5187–5204, 2011. (Cited on pages 21, 22, and 84.)

[CDP19] David Coudert, Guillaume Ducoffe, and Alexandru Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. *ACM Trans. Algorithms*, 15(3):33:1–33:57, 2019. (Cited on pages 8, 24, 27, 31, 32, 33, 37, 38, 40, 52, 62, 79, 82, 95, and 115.)

[CEN12] Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Homology flows, cohomology cuts. *SIAM J. Comput.*, 41(6):1605–1634, 2012. (Cited on page 8.)

[CFK+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. (Cited on page 8.)

[Cha96] Maw-Shang Chang. Algorithms for maximum matching and minimum fill-in on chordal bipartite graphs. In Tetsuo Asano, Yoshihide Igarashi, Hiroshi Nagamochi, Satoru Miyano, and Subhash Suri, editors, *Algorithms and Computation, 7th International Symposium, ISAAC '96, Osaka, Japan, December 16-18, 1996, Proceedings*, volume 1178 of *Lecture Notes in Computer Science*, pages 146–155. Springer, 1996. (Cited on page 24.)

[Cha10] Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010. (Cited on page 25.)

[CHL+12] Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce A. Reed, and Udi Rotics. Polynomial-time recognition of clique-width $\leq$3 graphs. *Discret. Appl. Math.*, 160(6):834–865, 2012. (Cited on page 62.)

[CHM81] M. Chein, Michel Habib, and M. C. Maurer. Partitive hypergraphs. *Discret. Math.*, 37(1):35–50, 1981. (Cited on page 37.)

[CKL+22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *CoRR*, abs/2203.00671, 2022. (Cited on pages 27 and 58.)

[CLB81] Derek G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discret. Appl. Math.*, 3(3):163–174, 1981. (Cited on page 9.)

[CMR00] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. (Cited on pages 24, 61, and 121.)

[CN85] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985. (Cited on page 27.)

[CO00] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000. (Cited on pages 62 and 63.)

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. (Cited on page 20.)

[CR05] Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005. (Cited on pages 21, 61, and 62.)

[CWC99] Ho Chin-Wen and Jou-Ming Chang. Solving the all-pairs-shortest-length problem on chordal bipartite graphs. *Information processing letters*, 69(2):87–93, 1999. (Cited on page 26.)

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. (Cited on page 8.)

[Die12] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. (Cited on page 15.)

[DK98] Elias Dahlhaus and Marek Karpinski. Matching and multidimensional matching in chordal and strongly chordal graphs. *Discret. Appl. Math.*, 84(1-3):79–91, 1998. (Cited on page 24.)

[DP21a] Guillaume Ducoffe and Alexandru Popa. The b-matching problem in distance-hereditary graphs and beyond. *Discret. Appl. Math.*, 305:233–246, 2021. (Cited on page 25.)

[DP21b] Guillaume Ducoffe and Alexandru Popa. The use of a pruned modular decomposition for maximum matching algorithms on some graph classes. *Discret. Appl. Math.*, 291:201–222, 2021. (Cited on pages 95 and 122.)

[Duc22a] Guillaume Ducoffe. Maximum Matching in Almost Linear Time on Graphs of Bounded Clique-Width. *Algorithmica*, 2022. (Cited on pages 8, 24, and 121.)

[Duc22b] Guillaume Ducoffe. Optimal Centrality Computations Within Bounded Clique-Width Graphs. *Algorithmica*, 2022. (Cited on pages 8, 26, 95, 115, and 121.)

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965. (Cited on pages 11 and 24.)

[EKM+04] Stephen G. Eubank, V. S. Anil Kumar, Madhav V. Marathe, Aravind Srinivasan, and Nan Wang. Structural and algorithmic aspects of massive social networks. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 718–727. SIAM, 2004. (Cited on page 12.)

[ER90] Andrzej Ehrenfeucht and Grzegorz Rozenberg. Theory of 2-structures, part II: representation through labeled tree families. *Theor. Comput. Sci.*, 70(3):305–342, 1990. (Cited on page 37.)

[EW92] Vladimir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Comput. Surv.*, 24(4):441–476, 1992. (Cited on page 9.)

[FG04] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Log.*, 130(1-3):3–31, 2004. (Cited on page 61.)

[FKM+19] Till Fluschnik, Christian Komusiewicz, George B. Mertzios, André Nichterlein, Rolf Niedermeier, and Nimrod Talmon. When can graph hyperbolicity be computed in linear time? *Algorithmica*, 81(5):2016–2045, 2019. (Cited on pages 8 and 95.)

[FKN69] M Fujii, T Kasami, and K Ninomiya. Optimal sequencing of two equivalent processors. *SIAM Journal on Applied Mathematics*, 17(4):784–789, 1969. (Cited on page 11.)

[Flo62] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962. (Cited on pages 25, 33, and 107.)

[FLS+18] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Trans. Algorithms*, 14(3):34:1–34:45, 2018. (Cited on pages 8, 24, and 95.)

[Fre76] Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):83–89, 1976. (Cited on page 25.)

[Fre77] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977. (Cited on page 11.)

[Fre87] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987. (Cited on page 26.)

[FRRS09] Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is np-complete. *SIAM J. Discret. Math.*, 23(2):909–939, 2009. (Cited on page 62.)

[Gab95] Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *J. Comput. Syst. Sci.*, 50(2):259–273, 1995. (Cited on page 27.)

[Gab18] Harold N. Gabow. Data structures for weighted matching and extensions to b-matching and f-factors. *ACM Trans. Algorithms*, 14(3):39:1–39:80, 2018. (Cited on pages 25, 31, 33, 38, 41, and 42.)

[Gal67] Tibor Gallai. Transitiv orientierbare graphen. *Acta Mathematica Hungarica*, 18(1-2):25–66, 1967. (Cited on page 35.)

[Gal14] François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. (Cited on page 24.)

[Gan11] Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In Dániel Marx and Peter Rossmanith, editors, *Parameterized and Exact Computation - 6th International Symposium, IPEC 2011, Saarbrücken, Germany, September 6-8, 2011. Revised Selected Papers*, volume 7112 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2011. (Cited on page 19.)

[Gar03] Frédéric Gardi. Efficient algorithms for disjoint matchings among intervals and related problems. In Cristian Calude, Michael J. Dinneen, and Vincent Vajnovszki,

editors, *Discrete Mathematics and Theoretical Computer Science, 4th International Conference, DMTCS 2003, Dijon, France, July 7-12, 2003. Proceedings*, volume 2731 of *Lecture Notes in Computer Science*, pages 168–180. Springer, 2003. (Cited on page 24.)

[GB13] Oded Green and David A. Bader. Faster clustering coefficient using vertex covers. In *International Conference on Social Computing, SocialCom 2013, SocialCom/PASSAT/BigData/EconCom/BioMedCom 2013, Washington, DC, USA, 8-14 September, 2013*, pages 321–330. IEEE Computer Society, 2013. (Cited on page 27.)

[GHN+12] Robert Ganian, Petr Hlinený, Jaroslav Nesetril, Jan Obdrzálek, Patrice Ossona de Mendez, and Reshma Ramadurai. When trees grow low: Shrubs and fast MSO1. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, volume 7464 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 2012. (Cited on pages 22 and 61.)

[GHN+19] Robert Ganian, Petr Hlinený, Jaroslav Nesetril, Jan Obdrzálek, and Patrice Ossona de Mendez. Shrub-depth: Capturing height of dense graphs. *Log. Methods Comput. Sci.*, 15(1), 2019. (Cited on pages 22 and 114.)

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. (Cited on page 7.)

[GK04] Andrew V. Goldberg and Alexander V. Karzanov. Maximum skew-symmetric flows and matchings. *Math. Program.*, 100(3):537–568, 2004. (Cited on pages 24 and 25.)

[GLO13] Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Z. Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013. (Cited on pages 20, 37, and 98.)

[GMN17] Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theor. Comput. Sci.*, 689:67–95, 2017. (Cited on pages 8 and 32.)

[GR99a] Andrew V. Goldberg and Satish Rao. Flows in undirected unit capacity networks. *SIAM J. Discret. Math.*, 12(1):1–5, 1999. (Cited on pages 27, 33, and 49.)

[GR99b] Martin Charles Golumbic and Udi Rotics. The clique-width of unit interval graphs is unbounded. *CONGRESSUS NUMERANTIUM*, pages 5–18, 1999. (Cited on page 22.)

[GR00] Martin Charles Golumbic and Udi Rotics. On the clique-width of some perfect graph classes. *Int. J. Found. Comput. Sci.*, 11(3):423–443, 2000. (Cited on pages 26 and 62.)

[Gue07] Frank Guerin. Tractable combinatorial auctions via graph matching. 2007. (Cited on page 11.)

[Gur17] Frank Gurski. The behavior of clique-width under graph operations and graph transformations. *Theory Comput. Syst.*, 60(2):346–376, 2017. (Cited on page 115.)

[GWY16] Frank Gurski, Egon Wanke, and Eda Yilmaz. Directed nlc-width. *Theor. Comput. Sci.*, 616:1–17, 2016. (Cited on pages 62, 66, 67, and 78.)

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. (Cited on page 24.)

[HK79] Frank Harary and Helene J Kommel. Matrix measures for transitivity and balance. *Journal of Mathematical Sociology*, 6(2):199–210, 1979. (Cited on page 12.)

[HK19] Falko Hegerfeld and Stefan Kratsch. On adaptive algorithms for maximum matching. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 71:1–71:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (Cited on page 24.)

[HP10] Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Comput. Sci. Rev.*, 4(1):41–59, 2010. (Cited on page 37.)

[HRG00] Monika Rauch Henzinger, Satish Rao, and Harold N. Gabow. Computing vertex connectivity: New bounds from old techniques. *J. Algorithms*, 34(2):222–250, 2000. (Cited on pages 27, 33, 56, and 57.)

[HT16] Yijie Han and Tadao Takaoka. An o(n3loglog n / log2 n) time algorithm for all pairs shortest paths. *J. Discrete Algorithms*, 38-41:9–19, 2016. (Cited on page 25.)

[HW07] Jan M. Hochstein and Karsten Weihe. Maximum s-t-flow with k crossings in $O(k^3 n \log n)$ time. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 843–847. SIAM, 2007. (Cited on page 8.)

[HZA22] Muhammad Kashif Hanif, Karl-Heinz Zimmermann, and Asad Anees. Accelerating all-pairs shortest path algorithms for bipartite graphs on graphics processing units. *Multim. Tools Appl.*, 81(7):9549–9566, 2022. (Cited on page 121.)

[IOO18] Yoichi Iwata, Tomoaki Ogasawara, and Naoto Ohsaka. On the power of tree-depth for fully polynomial FPT algorithms. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 41:1–41:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. (Cited on pages 8, 9, 13, 24, 26, 32, 95, 96, 99, 100, 102, 103, 105, 108, and 115.)

[IR78] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978. (Cited on page 26.)

[Joh77] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977. (Cited on page 33.)

[Joh98] Öjvind Johansson. Clique-decomposition, NLC-decomposition, and modular decomposition - relationships and results for random graphs. In *Congr. Numer.* Citeseer, 1998. (Cited on pages 66 and 79.)

[Kar93] David R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 21–30. ACM/SIAM, 1993. (Cited on page 27.)

[Kar00] David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. (Cited on pages 27 and 33.)

[KN18] Stefan Kratsch and Florian Nelles. Efficient and adaptive parameterized algorithms on modular decompositions. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 55:1–55:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. (Cited on page 12.)

[KN20] Stefan Kratsch and Florian Nelles. Efficient parameterized algorithms for computing all-pairs shortest paths. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 38:1–38:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. (Cited on page 12.)

[KN22] Stefan Kratsch and Florian Nelles. Efficient parameterized algorithms on graphs with heterogeneous structure: Combining tree-depth and modular-width. *CoRR*, abs/2209.14429, 2022. (Cited on page 13.)

[KNS22] Stefan Kratsch, Florian Nelles, and Alexandre Simon. On triangle counting parameterized by twin-width. *CoRR*, abs/2202.06708, 2022. (Cited on page 13.)

[Kön31] Dénes König. Graphen und matrizen, mat. *Lapok*, 38:116–119, 1931. (Cited on page 11.)

[KPP15] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Dynamic set intersection. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, volume 9214 of *Lecture Notes in Computer Science*, pages 470–481. Springer, 2015. (Cited on page 27.)

[KPP16] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1272–1287. SIAM, 2016. (Cited on page 27.)

[KS96] David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, 1996. (Cited on page 27.)

[KVKV11] Bernhard H Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*, volume 1. Springer, 2011. (Cited on pages 15 and 104.)

[Lam12] Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. (Cited on page 19.)

[Lam20] Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. (Cited on page 98.)

[LM17] Michael Lampis and Valia Mitsou. Treewidth with a quantifier alternation revisited. In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6-8, 2017, Vienna, Austria*, volume 89 of *LIPIcs*, pages 26:1–26:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. (Cited on page 98.)

[LNP+21] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Vertex connectivity in poly-logarithmic max-flows. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 317–329. ACM, 2021. (Cited on page 59.)

[LP20] Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 85–92. IEEE, 2020. (Cited on page 58.)

[LR93] Y. Daniel Liang and Chongkye Rhee. Finding a maximum matching in a circular-arc graph. *Inf. Process. Lett.*, 45(4):185–190, 1993. (Cited on page 24.)

[LS12] Andrzej Lingas and Dzmitry Sledneu. A combinatorial algorithm for all-pairs shortest paths in directed vertex-weighted graphs with applications to disc graphs. In Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser, and György Turán, editors, *SOFSEM 2012: Theory and Practice of Computer Science - 38th Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 21-27, 2012. Proceedings*, volume 7147 of *Lecture Notes in Computer Science*, pages 373–384. Springer, 2012. (Cited on page 26.)

[MdM05] Ross M. McConnell and Fabien de Montgolfier. Linear-time modular decomposition of directed graphs. *Discret. Appl. Math.*, 145(2):198–209, 2005. (Cited on page 37.)

[Men27] Karl Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927. (Cited on page 54.)

[Men16] Stefan Mengel. Parameterized compilation lower bounds for restricted cnf-formulas. *CoRR*, abs/1604.06715, 2016. (Cited on page 98.)

[MNN16] George B. Mertzios, André Nichterlein, and Rolf Niedermeier. Fine-grained algorithm design for matching. *CoRR*, abs/1609.08879, 2016. (Cited on pages 8, 24, and 95.)

[MNN20] George B. Mertzios, André Nichterlein, and Rolf Niedermeier. The power of linear-time data reduction for maximum matching. *Algorithmica*, 82(12):3521–3565, 2020. (Cited on page 24.)

[MR84] Rolf H Möhring and Franz J Radermacher. Substitution decomposition for discrete structures and connections with combinatorial optimization. In *North-Holland mathematics studies*, volume 95, pages 257–355. Elsevier, 1984. (Cited on page 37.)

[MS04] Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 248–255. IEEE Computer Society, 2004. (Cited on page 24.)

[MS06] Marcin Mucha and Piotr Sankowski. Maximum matchings in planar graphs via gaussian elimination. *Algorithmica*, 45(1):3–20, 2006. (Cited on page 24.)

[MT16] Ali Mohammadian and Vilmar Trevisan. Some spectral properties of cographs. *Discret. Math.*, 339(4):1261–1264, 2016. (Cited on page 83.)

[MV80] Silvio Micali and Vijay V. Vazirani. An o(sqrt(|v|) |e|) algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27. IEEE Computer Society, 1980. (Cited on pages 24, 25, 32, 33, and 38.)

[NdM06] Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. (Cited on page 20.)

[NdM12] Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. (Cited on page 21.)

[OMSW10] James B. Orlin, Kamesh Madduri, K. Subramani, and Matthew D. Williamson. A faster algorithm for the single source shortest path problem with few distinct positive lengths. *J. Discrete Algorithms*, 8(2):189–198, 2010. (Cited on page 8.)

[Orl13] James B. Orlin. Max flows in o(nm) time, or better. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 765–774. ACM, 2013. (Cited on pages 27, 33, 51, 54, and 56.)

[OS06] Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006. (Cited on page 21.)

[Oum08] Sang-il Oum. Rank-width is less than or equal to branch-width. *J. Graph Theory*, 57(3):239–244, 2008. (Cited on pages 21 and 22.)

[Pet04] Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theor. Comput. Sci.*, 312(1):47–74, 2004. (Cited on page 25.)

[Pot88] Alex Pothen. The complexity of optimal elimination trees. *Technical Report*, 1988. (Cited on page 123.)

[PR05] Seth Pettie and Vijaya Ramachandran. A shortest path algorithm for real-weighted undirected graphs. *SIAM J. Comput.*, 34(6):1398–1431, 2005. (Cited on pages 10, 25, and 82.)

[PSS16] Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for CNF formulas of bounded modular treewidth. *Algorithmica*, 76(1):168–194, 2016. (Cited on pages 96 and 98.)

[PW10] Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1065–1075. SIAM, 2010. (Cited on page 7.)

[Rao08a] Michaël Rao. Clique-width of graphs defined by one-vertex extensions. *Discret. Math.*, 308(24):6157–6165, 2008. (Cited on page 122.)

[Rao08b] Michaël Rao. Solving some np-complete problems using split decomposition. *Discret. Appl. Math.*, 156(14):2768–2780, 2008. (Cited on page 62.)

[Ren15] Yihui Ren. *Betweenness Centrality and Its Applications from Modeling Traffic Flows to Network Community Detection.* PhD thesis, University of Notre Dame, Mar 2015. (Cited on page 11.)

[RRVS14] Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014. (Cited on page 123.)

[RS91] Neil Robertson and Paul D. Seymour. Graph minors. x. obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2):153–190, 1991. (Cited on page 21.)

[RSÜ04] Alvin E Roth, Tayfun Sönmez, and M Utku Ünver. Kidney exchange. *The Quarterly journal of economics*, 119(2):457–488, 2004. (Cited on page 11.)

[RSÜ05] Alvin E Roth, Tayfun Sönmez, and M Utku Ünver. Pairwise kidney exchange. *Journal of Economic theory*, 125(2):151–188, 2005. (Cited on page 11.)

[Sch02] Alexander Schrijver. Combinatorial optimization. *preprint*, page 58, 2002. (Cited on page 104.)

[Sei95] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995. (Cited on page 25.)

[SGDJ04] Jonatan Schroeder, André Guedes, and Elias P Duarte Jr. Computing the minimum cut and maximum flow of undirected graphs. *Technical report, Federal University of Paraná*, 2004. (Cited on page 18.)

[SP15] Susmita Susmita and Manish Pandey. Algorithms of all pair shortest path problem. *International Journal of Computer Applications*, 120(15):1–6, 2015. (Cited on page 11.)

[SW97] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *J. ACM*, 44(4):585–591, 1997. (Cited on pages 27 and 53.)

[SW05] Thomas Schank and Dorothea Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In Sotiris E. Nikoletseas, editor, *Experimental and Efficient Algorithms, 4th InternationalWorkshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005, Proceedings*, volume 3503 of *Lecture Notes in Computer Science*, pages 606–609. Springer, 2005. (Cited on pages 33 and 47.)

[SYZ11] Asaf Shapira, Raphael Yuster, and Uri Zwick. All-pairs bottleneck paths in vertex weighted graphs. *Algorithmica*, 59(4):621–633, 2011. (Cited on page 26.)

[TCHP08] Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2008. (Cited on pages 31, 37, and 43.)

[TZ13] Svetlana Torgasin and Karl-Heinz Zimmermann. An all-pairs shortest path algorithm for bipartite graphs. *Central Eur. J. Comput. Sci.*, 3(4):149–157, 2013. (Cited on page 121.)

[Vaz20] Vijay V. Vazirani. A proof of the MV matching algorithm. *CoRR*, abs/2012.03582, 2020. (Cited on page 24.)

[Wan94] Egon Wanke. k-NLC graphs and polynomial algorithms. *Discrete Applied Mathematics*, 54(2-3):251–266, 1994. (Cited on pages 62, 64, and 65.)

[War62] Stephen Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, 1962. (Cited on pages 25, 33, and 107.)

[Wil18a] R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. (Cited on page 25.)

[Wil18b] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3447–3487. World Scientific, 2018. (Cited on page 25.)

[WS98] Duncan J Watts and Steven H Strogatz. Collective dynamics of small-world networks. *nature*, 393(6684):440–442, 1998. (Cited on page 12.)

[WW18] Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. (Cited on pages 25, 26, and 32.)

[Yus09] Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 950–957. SIAM, 2009. (Cited on pages 25, 33, 46, and 107.)

[YY93] Ming-Shing Yu and Cheng-Hsing Yang. A linear time algorithm for the maximum matching problem on cographs. *BIT*, 33(3):420–432, 1993. (Cited on page 24.)

[YZ07] Raphael Yuster and Uri Zwick. Maximum matching in graphs with an excluded minor. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 108–117. SIAM, 2007. (Cited on page 24.)

[Zwi02] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. (Cited on page 25.)

Selbstständigkeitserklärung

Ich erkläre, dass ich die Dissertation selbständig und nur unter Verwendung der von mir gemäß § 7 Abs. 3 der Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät, veröffentlicht im Amtlichen Mitteilungsblatt der Humboldt-Universität zu Berlin Nr. 42/2018 am 11.07.2018 angegebenen Hilfsmittel angefertigt habe.

Berlin, 2023

Florian Nelles